# Parametrising Neural Feedback Policies with Stability **and** Robustness Guarantees

Nicholas Habib Barbara

A thesis submitted in fulfilment of the
requirements for the degree of
Doctor of Philosophy

Australian Centre for Robotics
School of Aerospace, Mechanical, and Mechatronic Engineering
The University of Sydney

November 1, 2025

# Statement of Originality

This is to certify that the content of this thesis is my own work. This thesis has not been submitted for any other degree or purpose. I certify that the intellectual content of this thesis is the product of my own work, and that all assistance received in preparing this thesis and all sources have been acknowledged.

*Nicholas H. Barbara*
November 1, 2025

# Authorship Attribution Statement

The following publications make up the main content of this thesis.

[15] Nicholas H. Barbara, Ruigang Wang, Alexandre Megretski, and Ian R. Manchester, "React to Surprises: Stable-by-Design Neural Feedback Control and the Youla-REN," *Submitted to the IEEE Transactions on Automatic Control,* 2025.

[14] Nicholas H. Barbara, Ruigang Wang, and Ian R. Manchester, "R2DN: Scalable Parametrization of Contracting and Lipschitz Recurrent Deep Networks," *In Preparation for the IEEE Control Systems Letters,* 2025.

[11] Nicholas H. Barbara, Max Revay, Ruigang Wang, Jing Cheng, and Ian R. Manchester, "RobustNeuralNetworks.jl: a Package for Machine Learning and Data-Driven Control with Certified Robustness," *Proceedings of the JuliaCon Conferences,* 2025.

[13] Nicholas H. Barbara, Ruigang Wang, and Ian R. Manchester, "On Robust Reinforcement Learning with Lipschitz-Bounded Policy Networks," *Symposium on Systems Theory in Data and Optimization,* 2024.

[12] Nicholas H. Barbara, Ruigang Wang, and Ian R. Manchester, "Learning Over Contracting and Lipschitz Closed-Loops for Partially-Observed Nonlinear Systems," *Proceedings of the IEEE Conference on Decision and Control,* 2023.

[178] Ruigang Wang, Nicholas H. Barbara, Max Revay, and Ian R. Manchester, "Learning Over All Stabilizing Nonlinear Controllers for a Partially-Observed Linear System," *IEEE Control Systems Letters,* 2022.

Chapter 3 is based on [13]. Chapters 4 and 5 are based on [15, 12, 178]. Chapter 6 is based on [14]. Appendix B is based on [11]. Note that [15] is currently under review.

For each paper where I am listed as the first author, I designed the study, analysed the data, and wrote the drafts of the manuscript. For [178], I collaborated with the lead author, Dr. Ruigang Wang, to set up the numerical experiments, analyse the data, and write drafts of the manuscript.

In addition to the authorship attribution statements above, in cases where I am not the corresponding author of a published item, permission to include the published material has been granted by the corresponding author.

---

*Nicholas H. Barbara*
November 1, 2025

As supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements above are correct.

---

*Ian R. Manchester*
November 1, 2025

*For Maddie and Papa.*

*"All we have to decide is what to do with the time that is given us."*

J.R.R. Tolkien,
The Fellowship of the Ring.

# Abstract

Learning-based control is a powerful tool for nonlinear control design for complex dynamical systems. Driven by the rise of deep Reinforcement Learning (RL), most approaches parametrise control policies with black-box neural networks, which are universal approximators for nonlinear systems. These can easily be trained in simulation with simple, gradient-based optimisation schemes. However, black-box approaches such as deep RL suffer a fundamental limitation: they lack certifiable guarantees of closed-loop stability, robustness to disturbances, and sensitivity to model error.

This thesis introduces novel parametrisations of neural feedback policies with built-in stability and robustness guarantees. Instead of relying on black-box neural networks, we parametrise our policies with state-of-the-art robust neural networks that are designed to automatically satisfy stability and robustness constraints of their own.

Our main contribution combines robust neural networks with a nonlinear version of the Youla-Kučera parametrisation. We propose a theoretically-motivated framework that fuses classical and learning-based control with model information under a single policy architecture. The resulting policy parametrisation: (1) automatically guarantees closed-loop stability and robustness; (2) allows for plug-and-play optimisation with standard training pipelines based on gradient descent; and (3) is not restrictive in the controllers it covers (for certain classes of nonlinear systems). We derive rigorous theoretical certificates for our parametrisation in partially-observed, nonlinear systems with incremental stability requirements (contraction and Lipschitzness), and demonstrate its capability for stability-certified deep RL via numerical experiments.

We extend our study of robust learning-based control with two further contributions. In the first, we demonstrate via an empirical study that to improve robustness in existing deep RL architectures, one can directly replace the black-box neural network with a Lipschitz-bounded network. However, the structure of the network parametrisation is important. We show that only expressive parametrisations with tight Lipschitz bounds can boost closed-loop robustness without sacrificing nominal performance. In the second, we derive a new, scalable parametrisation of stable (contracting) and robust (Lipschitz) networks as a computationally-efficient alternative to existing robust neural networks. Our new parametrisation is up to an order of magnitude faster in training and inference than existing tools, and opens the possibility of learning large-scale stabilising controllers for high-dimensional systems in future work.

# Acknowledgements

This thesis would not have been possible without the guidance and support of some brilliant minds. First and foremost, my thanks go to my supervisor, Ian Manchester. Your persistence, patience, and ability to cut right to the heart of a problem are inspiring and have shaped the way I think. This extends to Ruigang (Ray) Wang, to whom I owe my sincere thanks for countless hours of discussion and collaboration. It has been a pleasure working with you both these past few years, and I am excited to see what comes next.

Thanks also go to Alexandre Megretski, for a single conversation one Friday afternoon in Boston that fundamentally changed my approach to research, and set me on a path to discovering the material presented in this thesis.

Thanks, to all the members of the Planning and Control group, the J04 lunch crew, and the ACFR administrative staff for making it fun and enjoyable to come to work each day. Special thanks go to Jack, Jesse, Em, Priyanka, and Luci[1] for your many years of enduring friendship (not to mention the impromptu lunches, dinners, discussions, and collaboration) without which this thesis would never have reached its conclusion.

Thanks, to the Lindfield District Cricket Club, for giving me a Saturday outlet and for a club culture that professional teams around the world can only envy.

Thanks, to my parents, for your never-ending support, love, and care, and for never giving up in your attempts to understand what I actually do.

And finally, my heartfelt thanks to Zoe and Luci[2]. You were there for me every step of the way, and I could not have done it without either of you.

[1] Lucinda.
[2] Jean Luc.

*This research was supported by an Australian Government Research Training Program (RTP) Scholarship. Generative AI was used not used in any way to write the contents of this thesis, other than as a code translation tool in part of Chapter 3.*

# Contents

# Nomenclature

| | |
|---|---|
| **ACFR** | Australian Centre For Robotics |
| **AOL** | Almost Orthogonal Lipschitz |
| **APG** | Analytic Policy Gradients |
| **CNN** | Convolutional Neural Network |
| **CPU** | Central Processing Unit |
| **DNN** | Deep Neural Network |
| **GAE** | Generalised Advantage Estimation |
| **GPU** | Graphics Processing Unit |
| **IMC** | Internal Model Control |
| **IQC** | Integral Quadratic Constraint |
| **LBDN** | Lipschitz-Bounded Deep Network |
| **LMI** | Linear Matrix Inequality |
| **LQG** | Linear Quadratic Gaussian |
| **LQR** | Linear Quadratic Regulator |
| **LSTM** | Long-Short Term Memory |
| **LTI** | Linear Time Invariant |
| **MLP** | Multi-Layer Perceptron |
| **MPC** | Model Predictive Control |
| **NRMSE** | Normalised Root-Mean-Square Error |
| **ODE** | Ordinary Differential Equation |
| **PDE** | Partial Differential Equation |

**PGD** Projected Gradient Descent

**PPO** Proximal Policy Optimisation

**R2DN** Robust Recurrent Deep Network

**REN** Recurrent Equilibrium Network

**RNN** Recurrent Neural Network

**RL** Reinforcement Learning

**SGD** Stochastic Gradient Descent

**SLS** System-Level Synthesis

**SN** Spectral Normalisation

# Chapter 1

# Introduction

## 1.1 Learning Robustly-Stabilising Neural Policies

The motivation for this thesis comes from the field of deep Reinforcement Learning (RL). Deep RL is an extremely powerful tool for general-purpose nonlinear control design via simulation [7, 159, 18]. It has been the driving force behind many recent successes in learning-based control, including: teaching computers to play arcade [113] and strategy games [149, 150]; enabling highly-dynamic robotic locomotion [97, 148, 138] and manipulation [62, 81]; reaching super-human performance in high-speed autonomous drone racing [83, 156]; and even in nuclear fusion [35]. The power of deep RL comes from its use of simple, gradient-based optimisation methods [107, 143] to train control policies parametrised by Deep Neural Networks (DNNs), which we call policy networks. Together with modern automatic differentiation tools [19, 22] and hardware-accelerated physics simulators [46, 111], it is now possible to train deep RL policies to control real-world robotic systems in less than an hour on a standard desktop Graphics Processing Unit (GPU) [138, 25].

The standard approach to training a policy network with deep RL is to minimise empirical estimates of an expected cost over a distribution of different initial system

states, disturbances, and physical properties [159, 5, 166]. It is here, however, that we encounter one of the key limitations of deep RL – there are no guarantees on the stability, bounded response to disturbances, or sensitivity to model error of the resulting closed-loop system. While the risk of instability can be reduced by using long training horizons with regulation- or tracking-style cost functions, closed-loop stability and robustness are not explicitly part of the standard RL problem formulation. This makes deep RL policies susceptible to unexpected failures in new environments [27, 36], even in the presence of very small disturbances such as those generated by targeted adversarial attacks [70, 139, 144].

Readers with a background in control theory may find the lack of closed-loop stability guarantees in deep RL rather disturbing. Indeed, classical control is built on a deep and rigorous understanding of stability theory for dynamical systems [186, 196, 175, 130], and one of the primary objectives of control engineering is to design (robustly) stabilising controllers. The performance of these controllers should of course be optimised, but always subject to the fundamental constraint that the closed-loop system be *provably* stable in all operating conditions. This raises the question of whether a similar design philosophy can be brought to learning-based control without sacrificing the power and flexibility of methods such as deep RL. Specifically, we consider the following question:

*Can we learn robustly-stabilising neural feedback policies with closed-loop guarantees for nonlinear systems using simple, gradient-based optimisation tools?*

How best to approach this question remains an open challenge [69]. Verification tools such as [33, 190, 123] have been developed to test the closed-loop stability and robustness of learned policy networks after they have been trained. However, training a policy network often takes tens of minutes (with many GPUs or on small problems) or hours to days (otherwise), making it infeasible to iteratively train a network, verify its closed-loop stability, adjust the training setup, and train it again. Instead, other works such as [60, 80, 69] have proposed projected gradient descent methods, where the parameters of a neural network are projected into a known set of robustly-stabilising

policies at each step during training. While this approach guarantees closed-loop stability, projection steps are computationally expensive and scale poorly with both the size of the dynamical system and the complexity of the policy network. The growth in the scale of systems and policies that can be trained by RL methods is significantly outpacing the scalability of such methods. An alternative approach which avoids projections is to apply the classical small-gain theorem [2]. That is, if an upper bound on the input-output gain of the dynamical system to be controlled is known, we can guarantee closed-loop stability by also limiting the input-output gain of the policy network [145, 78]. However, the small-gain theorem is known to be a conservative design tool. The trade-off for computational efficiency is therefore a limit on the expressive power of the policy network, which in turn limits performance. Note that there is also a wealth of related literature called safe RL, where the objective is to learn policy networks that constrain a dynamical system within some "safe" region of its state-space [54, 65, 23, 61]. Even in safe RL, however, the question of learning robustly-stabilising policies *within* the safe region remains an open problem [23, Sec. 3.3.1.1].

The approach we take in this thesis is to construct neural policy parametrisations that are *stable by design* – i.e., closed-loop stability and robustness constraints are embedded directly within the policy parametrisation itself. Specifically, we study policy parametrisations which automatically satisfy the following high-level requirements.

1. **Robustly-stabilising**: the policy parametrisation only includes controllers which stabilise the closed-loop system according to particular notions of stability and robustness.

2. **Differentiable**: the policy parametrisation is compatible with standard automatic differentiation tools to facilitate plug-and-play learning with existing optimisation pipelines, such as those based on Stochastic Gradient Descent (SGD).

3. **Expressive**: the policy parametrisation is expressive and contains a large set of (preferably all) robustly-stabilising controllers for a given system.

These three requirements allow us to decouple closed-loop stability and robustness certificates from the training setup, which includes the optimisation algorithms, cost functions, and training data. Any policy parametrisation satisfying these requirements is guaranteed to be robustly-stabilising by construction, and can easily be trained via existing data-driven control pipelines such as deep RL without worrying about closed-loop stability or robustness during training – that is, there is no need for additional computationally-expensive projections or stability analysis procedures during training. Moreover, the more expressive the parametrisation, the greater the likelihood that it will contain high-performing policies. In a sense, the goal of this thesis is to design policy parametrisations which maximise the third requirement subject to the constraints of the first two.

The fundamental tool that will allow us to construct stable-by-design policy parametrisations is a robust neural network. The current state-of-the-art approach to parametrising neural policies is to use black-box neural networks, which are widely known to be sensitive to small perturbations and have no constraints on their input-output robustness or internal stability [160, 55, 1]. This makes it challenging to certify stability and robustness of the resulting closed-loop system, and is the root cause of the difficulties with the existing methods for learning stabilising policies outlined above. In contrast, robust neural networks are designed to automatically satisfy a set of user-defined behavioural constraints including certified bounds on their sensitivity to perturbations [133, 181] and guarantees on their internal stability [135, 136].

## 1.2 Contributions and Outline

The contributions of this thesis are as follows. In Chapters 3 to 5, we address the problem of parametrising robustly-stabilising neural policies using robust neural networks. Roughly speaking, we replace the black-box policy network in a typical neural policy architecture with either a robust neural network (Chapter 3) or a more sophisticated policy parametrisation built around a robust neural network (Chapters 4 and 5). We

then use the certified robustness properties of the network to regulate the stability and robustness of the closed-loop system via the policy parametrisation itself, which is not possible with black-box networks. The main theoretical contribution is the study of a stable-by-design policy parametrisation based on the classical Youla-Kučera parametrisation [193, 92] in Chapters 4 and 5, where we find that the closed-loop system automatically inherits the same robust stability properties as the policy network under assumptions.

In Chapter 6, we shift our focus to the robust neural networks themselves. We propose a new method to improve the scalability of the robust network parametrisation from [136] (which we use as the basis of our policy class in Chapter 5) so that our policies can be scaled to large neural models for high-dimensional problems in future work.

A detailed overview of the contributions of each chapter is provided below, including the publications on which they are based (see Section 1.3).

- Chapter 3 is based on [13]. It presents a study of robust policy networks satisfying a particular constraint on their input-output sensitivity called a Lipschitz bound. We investigate the benefits of different parametrisations for Lipschitz-bounded policy networks in deep RL by analysing their empirical performance and robustness on two representative problems: pendulum swing-up and Atari Pong. We illustrate that policy networks with smaller Lipschitz bounds are more robust to disturbances, random noise, and targeted adversarial attacks than unconstrained policies composed of vanilla Multi-Layer Perceptrons (MLPs) or Convolutional Neural Networks (CNNs). However, the structure of the Lipschitz layer is important. We find that the widely-used method of spectral normalisation [112] is too conservative and severely impacts clean performance, whereas more expressive Lipschitz layers such as the recently-proposed Sandwich layer [181] can achieve improved robustness without sacrificing clean performance.

- Chapters 4 and 5 are based on [15, 12, 178]. We study stable-by-design parametrisations of nonlinear policies for learning-based control. We propose a structure

based on a nonlinear version of the Youla-Kučera parametrisation combined with robust neural networks such as the Recurrent Equilibrium Network (REN) [136]. The resulting parametrisations are unconstrained, and hence can be searched over with first-order optimization methods, while always ensuring closed-loop stability by construction. We study the combination of (a) nonlinear dynamics, (b) partial observation, and (c) incremental closed-loop stability requirements (contraction and Lipschitzness). We find that with any two of these three features, a contracting and Lipschitz Youla parameter always leads to contracting and Lipschitz closed loops. However, if all three hold, then incremental stability can be lost with exogenous disturbances. Instead, a weaker condition is maintained, which we call d-tube contraction and Lipschitzness. We further obtain converse results showing that the proposed parametrisation covers all contracting and Lipschitz closed loops for certain classes of nonlinear systems. Numerical experiments illustrate the utility of our parametrisation when learning controllers with built-in stability certificates for: (i) cost functions without stabilizing effects; (ii) short training horizons; and (iii) uncertain systems.

- Chapter 6 is based on [14]. It focuses on improving the scalability of robust neural network parametrisations so that they can be applied to higher-dimensional learning tasks in future work. Specifically, we present the Robust Recurrent Deep Network (R2DN), which is a feedback interconnection of an Linear Time Invariant (LTI) system and a 1-Lipschitz deep feedforward network. We directly parametrise the weights of the R2DN so that the models are stable (contracting) and robust to small input perturbations (Lipschitz) by design. Our parametrisation uses a similar structure to the REN architecture [136], but without the requirement to iteratively solve an equilibrium layer at each time-step. This speeds up model evaluation and backpropagation on GPUs, and makes it computationally feasible to scale up the network size, batch size, and input sequence length in comparison to RENs. We compare R2DNs to RENs on three representative problems in nonlinear system identification, observer design, and deep

RL and find that training and inference are both up to an order of magnitude faster with similar test set performance, and that training and inference times scale more favourably with respect to model expressivity.

Another significant contribution of this thesis is the development of two software packages[1,2] open-sourcing the robust neural networks developed at the Australian Centre For Robotics (ACFR) during the author's PhD candidature. Partial documentation for one of these packages is included in this thesis.

- Appendix B is based on [11]. It documents `RobustNeuralNetworks.jl`, an open-source software package written in the Julia programming language [19] which contains implementations of Lipschitz-bounded Sandwich networks from [181] and RENs from [136]. The package was designed to make these recently-developed robust network architectures accessible to the community, and allows users to interface directly with Julia's most widely-used machine learning package, `Flux.jl` [76]. We discuss how the parametrisation of robust neural networks can be elegantly represented in Julia code, give an overview of the package, and provide a tutorial demonstrating its use in image classification, RL, and nonlinear observer design. The package is registered on the Julia general package registry.

The remainder of this thesis is structured as follows. Chapter 2 provides an overview of key concepts and tools used in the main body of this thesis, including: neural policy parametrisations and their lack of closed-loop guarantees; incremental stability and robustness in nonlinear systems; a review of the robust neural network architectures from [181, 136]; and numerical methods for solving deep RL problems. The main contributions are in Chapters 3 to 6. Chapter 7 presents a summary of key findings and concluding remarks, including suggested directions for future work, and Appendix A contains additional proofs not included in the main body of this thesis. We conclude with Appendix B on `RobustNeuralNetworks.jl`.

[1] https://github.com/acfr/RobustNeuralNetworks.jl
[2] https://github.com/acfr/RobustNeuralNetworks

## 1.3 Publications

The following publications make up the main content of this thesis. Note that [15] is currently under review, and an updated version of [14] is currently in preparation.

[15] Nicholas H. Barbara, Ruigang Wang, Alexandre Megretski, and Ian R. Manchester, "React to Surprises: Stable-by-Design Neural Feedback Control and the Youla-REN," *Submitted to the IEEE Transactions on Automatic Control,* 2025.

[14] Nicholas H. Barbara, Ruigang Wang, and Ian R. Manchester, "R2DN: Scalable Parametrization of Contracting and Lipschitz Recurrent Deep Networks," *In Preparation for the IEEE Control Systems Letters,* 2025.

[11] Nicholas H. Barbara, Max Revay, Ruigang Wang, Jing Cheng, and Ian R. Manchester, "RobustNeuralNetworks.jl: a Package for Machine Learning and Data-Driven Control with Certified Robustness," *Proceedings of the JuliaCon Conferences,* 2025.

[13] Nicholas H. Barbara, Ruigang Wang, and Ian R. Manchester, "On Robust Reinforcement Learning with Lipschitz-Bounded Policy Networks," *Symposium on Systems Theory in Data and Optimization,* 2024.

[12] Nicholas H. Barbara, Ruigang Wang, and Ian R. Manchester, "Learning Over Contracting and Lipschitz Closed-Loops for Partially-Observed Nonlinear Systems," *Proceedings of the IEEE Conference on Decision and Control,* 2023.

[178] Ruigang Wang, Nicholas H. Barbara, Max Revay, and Ian R. Manchester, "Learning Over All Stabilizing Nonlinear Controllers for a Partially-Observed Linear System," *IEEE Control Systems Letters,* 2022.

## 1.4 Notation

Let $\mathbb{N}$ and $(\mathbb{R}^+)$ $\mathbb{R}$ be the set of natural and (positive) real numbers, respectively.

We denote the set of sequences $x : \mathbb{N} \to \mathbb{R}^n$ by $\ell^n$, where the superscript $n$ is omitted if it is clear from context. We write $\ell_p^n \subset \ell^n$ for the set of sequences with finite $\ell_p$ norm for $p \in [1, \infty)$, where the $\ell_p$ norm is defined by

$$\|x\|_p := \left( \sum_{t=0}^{\infty} |x_t|^p \right)^{\frac{1}{p}}$$

for any $x \in \ell^n$, and $|\cdot|$ is the Euclidean norm. Letting $x_{i:j} := (x_i, x_{i+1}, \ldots, x_j)$ be the sequence truncation over $[i, j]$ with $i \leq j$, we use $\|x\|_{p,T}$ to denote the $\ell_p$ norm of $x_{0:T}$ with $T \in \mathbb{N}$. For the majority of this thesis, we are interested in the $\ell_2$ norm of signals, hence we use the short-hand $\|\cdot\| := \|\cdot\|_2$ and $\|\cdot\|_T := \|\cdot\|_{2,T}$ where appropriate.

We write $A \succ 0, A \succeq 0, A \prec 0, A \preceq 0$ for positive-definite, positive semi-definite, negative-definite, and negative semi-definite matrices $A \in \mathbb{R}^{n \times n}$, respectively.

A function $f : \mathbb{R}^m \to \mathbb{R}^n$ is said to be Lipschitz continuous if $\exists\, \gamma^\star \in \mathbb{R}^+$ such that

$$|f(a) - f(b)| \leq \gamma^\star |a - b|$$

for all $a, b \in \mathbb{R}^n$. We call the function $\gamma$-Lipschitz if the condition holds for a particular $\gamma \in \mathbb{R}^+$ where $\gamma \geq \gamma^\star$.

We denote the concatenation of vectors $u \in \mathbb{R}^n$, $v \in \mathbb{R}^m$ as $[u; v]$.

# Chapter 2

# Background

The work presented in this thesis combines ideas from classical feedback control and modern machine learning. In this chapter, we present an overview of the necessary background to understand our main contributions in Chapters 3 to 6. We assume that the reader has a background in control theory and some familiarity with standard methods in machine learning and learning-based control.

We start by reviewing the problem of learning neural feedback policies through the lens of deep RL, and introduce the concept of a neural policy parametrisation. We then discuss limitations of existing black-box neural policy parametrisations, in particular their failure to guarantee stability and robustness in the closed-loop system. This is followed by an introduction to two specific notions of nonlinear stability and robustness (respectively) – contraction [98] and incremental Integral Quadratic Constraints (IQCs) [108]. Having defined these notions, we introduce the concept of a robust or behaviourally-constrained neural network and discuss two important network architectures which we ultimately use in place of black-box policy networks in Chapters 3 to 6. We conclude with a brief review of numerical methods in deep RL, which we use for training policy networks later in this thesis.

## 2.1 Learning Neural Feedback Policies

There are many strategies with which to learn feedback controllers from data [32, 7, 45]. The most powerful methods for nonlinear systems all share a common trait – they use a high-dimensional neural network to parametrise the control policy, and learn the weights of the network via first-order optimisation schemes. This means, however, that the most powerful methods are also those that are affected by fundamental limitations on the robustness of black-box neural networks.

Before reviewing black-box neural networks and their relevant advantages and limitations, we first formally introduce the problem of learning feedback control policies, which will form the basis of the main contributions in this thesis.

### 2.1.1 Problem Formulation

Consider the block-diagram in Figure 2.1. We are interested in controlling discrete-time, nonlinear dynamical systems $\mathcal{G}$ with state-space representations

$$\begin{aligned} x_{t+1} &= f(x_t, u_t, w_t) \\ y_t &= h(x_t, u_t, v_t), \end{aligned} \tag{2.1}$$

internal states $x_t \in \mathbb{R}^{n_x}$, controlled inputs $u_t \in \mathbb{R}^{n_u}$, measured outputs $y_t \in \mathbb{R}^{n_y}$, and (unknown) disturbances $d = [w_t; v_t] \in \mathbb{R}^{n_d}$.

Our main objective is to design controllers $u = \mathcal{K}(y)$ which stabilise the closed-loop system in the presence of disturbances, and which minimise arbitrary cost functions $J(\mathcal{K})$ depending on the states and controls. A typical cost structure is

$$J(\mathcal{K}) = \mathop{\mathbb{E}}_{x_0,d}\left[\sum_{t=0}^{T-1} g(x_t, u_t) + g_f(x_T)\right] \quad \forall T \in \mathbb{N}, \tag{2.2}$$

where the expectation $\mathbb{E}[\cdot]$ is taken over some distribution of disturbances $d$ and initial states $x_0$, and $g, g_f$ are piecewise differentiable stage and terminal costs, respectively.

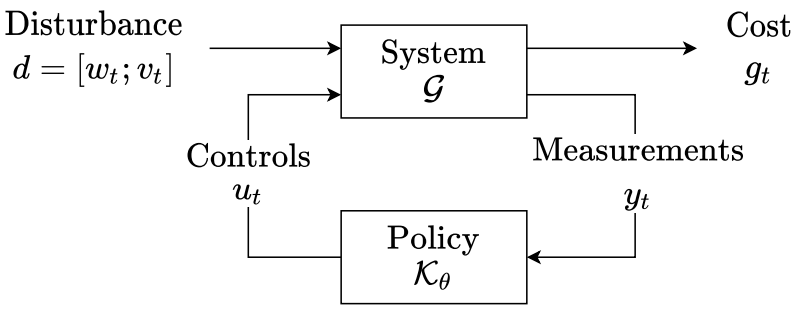



**Figure 2.1:** Typical block diagram for feedback control and reinforcement learning.

The typical workflow is to choose a controller structure or *policy class* $\mathcal{K}_\theta$ parametrised by a vector $\theta \in \Theta \subseteq \mathbb{R}^N$, and to optimise over controller parameters $\theta \in \Theta$ until some design specifications are met. We can therefore write the control design problem as

$$\min_{\theta \in \Theta} J(\theta) \quad \text{s.t.} \quad y = \mathcal{G}(u, d), \; u = \mathcal{K}_\theta(y). \tag{2.3}$$

There are some simplified settings in which (2.3) can be solved exactly. For example, if $\mathcal{G}$ is an LTI system and $J(\theta)$ a quadratic cost on the states and controls, it is well-known that the optimal structure for $\mathcal{K}_\theta$ is the series interconnection of a Kalman filter and state-feedback controller, and several constructive methods exist for choosing the optimal controller and observer gains $\theta$ [41, 64]. In the nonlinear setting, however, solving (2.3) is extremely challenging. In general, it is not clear what structure a controller $\mathcal{K}_\theta$ should have for it to be a "good" controller, and even if a particular structure is chosen, finding controller parameters $\theta$ which even approximately minimise the cost function (2.2) can be tricky.

The approach taken in many learning-based control methods is to choose a high-dimensional DNN for the controller $\mathcal{K}_\theta$. In this case, $\theta$ is a vector of the network's learnable weights and biases. How best to find the minimising $\theta$ then becomes a matter of preference. For example, the common approach in deep RL is as follows. Rather than trying to solve (2.3) exactly, the cost function is approximated by sampling over batches of $M$ trajectories of the closed-loop system with different initial conditions and

disturbances to estimate

$$\hat{J}(\theta) := \frac{1}{M} \sum_{m=0}^{M-1} \left[ \sum_{t=0}^{T-1} g(x_t^m, u_t^m) + g_f(x_T^m) \right] \approx J(\theta). \tag{2.4}$$

Typically $M$ is chosen to be very large (hundreds or thousands) to give a good estimate of the true cost for a particular choice of controller parameters $\theta$. The optimal choice of $\theta$ is then estimated by running variants of the classic gradient descent algorithm

$$\theta^{k+1} \leftarrow \theta^k - \alpha \nabla_\theta \hat{J}(\theta) \tag{2.5}$$

on the approximate cost function (2.4) until convergence, where $\alpha > 0$ is the learning rate (a hyperparameter to be chosen). A significant topic of research in deep RL is how to approximate the cost gradient $\nabla_\theta \hat{J}(\theta)$ in (2.5) in a computationally-efficient manner. The most successful approaches leverage automatic differentiation tools and massive parallelism with GPUs to approximate the cost gradient – see for example [113, 141, 143, 63, 49]. We defer our discussion of specific deep RL algorithms to Section 2.4.

### 2.1.2 Neural Policy Parametrisations

When a neural network is used to parametrise the feedback controller $\mathcal{K}_\theta$ in Figure 2.1, we call it a *neural policy parametrisation*. The strength of a neural policy parametrisation lies in the fact that sufficiently large neural networks universally approximate all continuous nonlinear functions (for static, feedforward networks) and all nonlinear state-space models (for recurrent networks) [68, 147]. Moreover, deep neural networks are directly compatible with automatic differentiation tools and can easily be trained with gradient descent methods based on (2.5). Neural policy parametrisations therefore offer a straightforward method for learning over a (very) large class of feedback controllers to optimise arbitrary cost functions (2.4).

The most common neural network structure used to parametrise feedback controllers

is a multi-layer feedforward network, which is a static nonlinear function defined by

$$\begin{aligned} w_0 &= u, \\ w_{k+1} &= \sigma(W_k w_k + b_k), \quad k = 0, \dots, L-1 \\ y &= W_L w_L + b_L, \end{aligned} \tag{2.6}$$

where $u \in \mathbb{R}^{n_u}, w_k \in \mathbb{R}^{n_k}, y \in \mathbb{R}^{n_y}$ are the input, hidden unit of the $k^{\text{th}}$ layer, and output of the network, respectively, and $L$ is the number of hidden layers. The terms $W_k \in \mathbb{R}^{n_{k+1} \times n_k}$, and $b_k \in \mathbb{R}^{n_k}$ are the learnable weight matrix and bias vector of the $k^{\text{th}}$ layer, respectively, and $\sigma$ is a scalar, nonlinear activation function (e.g., ReLU, tanh).

The two most common examples of feedforward networks (2.6) are MLPs and CNNs. Both are widely used as neural policy parametrisations, with practitioners typically relying on small MLPs for problems with low-dimensional measurement spaces (e.g., encoder readings in a robotic system [138, 83]) and CNNs for problems with high-dimensional measurements (e.g., images and video streams in vision-based feedback control [113, 81]). In the case of an MLP, the weight matrices $W_k$ are (in general) dense, whereas a CNN uses highly-structured weights and is implemented in practice by convolving the layer inputs with learnable convolution kernels [90].

A common assumption when learning neural policies is that all states of the system $\mathcal{G}$ in Figure 2.1 are directly measurable – i.e., $y_t = x_t$ exactly in (2.1). In this setting (under standard stochastic optimal control formulations) the optimal policy is a static, state-feedback controller [8], and it is common to use a memoryless network architecture like (2.6) to parametrise the policy [159]. In many practical scenarios, however, only partial state information is available. For example, one may have encoders reading the angular positions of joints in a robotic system, but no direct measurements of angular velocities or vibrational modes in flexible appendages. In classical feedback control, the solution in these situations is to add internal dynamics to the controller structure $\mathcal{K}_\theta$, often in the form of a state observer. A similar strategy in learning-based control is to use a dynamic or recurrent policy network which has an internal state – i.e., a

Recurrent Neural Network (RNN) [185]. The classic structure of a single-layer RNN is

$$\begin{aligned} x_{t+1} &= \sigma(C_1 x_t + D u_t + b_x) \\ y_t &= C_2 x_t + b_y \end{aligned} \tag{2.7}$$

where $x_t$ is the internal state at time $t \in \mathbb{N}$, $C_1 \in \mathbb{R}^{n_x \times n_x}, C_2 \in \mathbb{R}^{n_y \times n_x}, D \in \mathbb{R}^{n_x \times n_u}$ are the layer's weight matrices, and $b_x \in \mathbb{R}^{n_x}, b_y \in \mathbb{R}^{n_y}$ are the layer's bias vectors. Multi-layer RNNs can be constructed in much the same way as (2.6) by connecting multiple copies of (2.7) in series. Note, however, that recurrent policy networks parametrised by classic RNNs can be difficult to train due to internal model instability, which is widely known in the literature and is often referred to as the exploding gradients problem [17, 134]. In practice, Long-Short Term Memory (LSTM) networks [67] are preferred to vanilla RNNs due to their superior numerical performance [10, 185, 148].

It is important to reflect here that, unlike many controller parametrisations in classical optimal or robust control (e.g., [196, 155, 130]), no model information is explicitly included in a standard neural policy parametrisation. This is often seen as an advantage of learning-based methods such as deep RL and the term "model-free" is typically applied (e.g., [10, 97, 107, 151]). We note, however, that most deep RL policy networks, particularly those for robotics applications, are trained in high-fidelity physics simulators (e.g., [167, 46, 111]). While no explicit mathematical model is written into the controller parametrisation, the simulator is itself a model of the robotic system and its environment, hence a model is present and the neural network learns some internal representation of the physics during training. We will introduce an example of a policy parametrisation that explicitly includes model information later in this thesis (Chapters 4 and 5).

### 2.1.3 Stability and Robustness Are Not Guaranteed

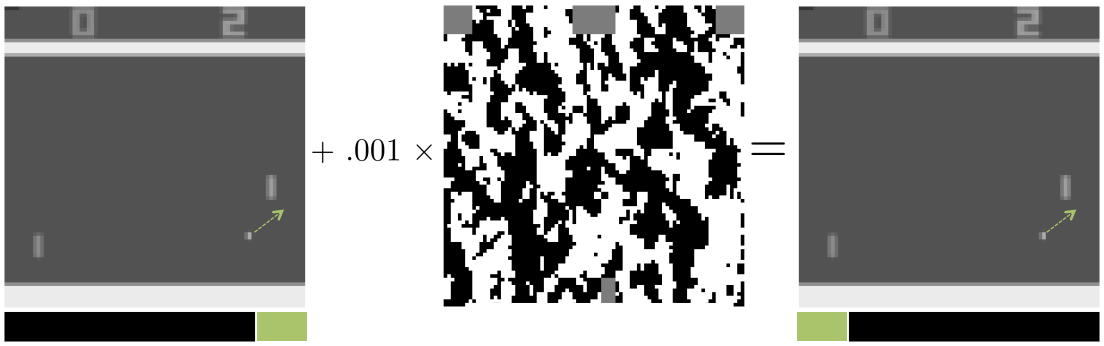
Most neural network architectures, such as (2.6) and (2.7), are simple function approximators. We refer to these architectures as "black-box" since they take an input vector,

return an output vector, and we have no control over the inner workings of the network other than by training them to minimise high-level cost functions. This very general structure leads to a fundamental limitation of black-box neural networks – they have no constraints on their input-output behaviour or internal stability.

In fact, black-box policy networks are widely known to be highly sensitive to small perturbations to their inputs [160], which in turn can de-stabilise the closed-loop system. The sensitivity of feedforward networks (2.6) was first observed in the context of image classification, where small amounts of optimised adversarial noise (adversarial attacks) were found to severely degrade the performance of CNN classifiers in [160, 55]. Similarly-styled adversarial attacks were also observed to negatively effect performance in recurrent network architectures (e.g., [26]). This vulnerability extends to the closed-loop setting too, where adversarial attacks have been shown to de-stabilise even state-of-the-art policy networks trained via deep RL [70, 139, 144].

We show two examples of this phenomenon in Figure 2.2. In Figure 2.2a [70], the policy network was trained to win the game of Atari Pong. The adversarial noise added to the gameplay image is entirely imperceptible to the human eye, and yet it is enough to cause the policy to make a sub-optimal decision and hold its paddle still instead of moving it towards the puck. The sensitivity of policy networks to adversarial attacks is also a problem outside of toy examples. Figure 2.2b shows a recent example from [144] on the ANYmal quadrupedal robot [73], where the state-of-the-art locomotion policy network from [109] was rendered closed-loop unstable by small, optimised adversarial noise added to the policy's measurements.

It is common to improve the robustness of neural policy networks by augmenting the training data with adversarial examples or realistic perturbations to the closed-loop system during training [122, 30, 166, 139]. Alternatively, one can train the policy network with specialised loss functions which encourage smooth responses that are less sensitive to perturbations [116, 114]. The key issue with these approaches, however, is that they still rely on black-box neural networks. This means that they do not




**(a)** Example on Atari Pong from [70].

**(b)** Example on ANYmal quadruped from [144].

**Figure 2.2:** Black-box policy networks are not robust to perturbations: (a) a policy networked trained to play Atari Pong is tricked into not moving the controlled paddle (right) instead of moving it down to meet the puck (left) by adding very small amounts of adversarial noise (middle) to the image; (b) a policy network trained to control the ANYmal quadruped is rendered closed-loop unstable when subject to adversarial measurement noise.

provide any certifiable guarantees of closed-loop stability and robustness, and that any improvement in empirical robustness is entirely dependent on the data used to train the policy. For the remainder of this thesis, we will be interested in policy networks which come equipped with certificates of internal stability and reduced input-output sensitivity so that we can derive guarantees of stability and robustness independently of the training data.

## 2.2 Incremental Stability and Robustness

Having established a need for learning neural feedback policies which provably and robustly stabilise nonlinear dynamical systems, we now need clear definitions of what is meant by closed-loop stability and robustness in this thesis. This section introduces contraction [98] and incremental IQCs [108] as our definitions of internal stability and robustness for nonlinear systems, respectively. We provide precise definitions of contracting systems admitting incremental IQCs, and highlight useful properties and special cases that motivate our study of such systems.

### 2.2.1 Contraction

In contrast to stability theory for LTI systems, common definitions of stability for nonlinear systems come in varying degrees of strength [87, 155]. For example, asymptotic stability is a relatively weak notion of nonlinear stability. It has minimal restrictions on the transient behaviour or rate at which a system eventually converges to some equilibrium state or trajectory. A stronger notion is exponential stability, which enforces that the state trajectories of a system exponentially converge to a particular equilibrium state or trajectory. The stability notion we are interested in achieving in this thesis is one step stronger – contraction, also known as incremental exponential stability [98].

A contracting dynamical system is one in which all the state trajectories exponentially converge together when given the same input sequence (see Figure 2.3). It is a strong, *incremental* form of nonlinear stability in that it describes the stability of the differences between trajectories. This makes it completely independent of any particular equilibrium point or trajectory, unlike other stability notions such as asymptotic or exponential stability. Incremental stability is well-suited to machine-learning applications because it allows for the stabilisation of all trajectories and smooth closed-loop responses. This is particularly useful: when a trained policy network is required to generalise to unseen data; for providing intuitive responses when hand-tuning; when

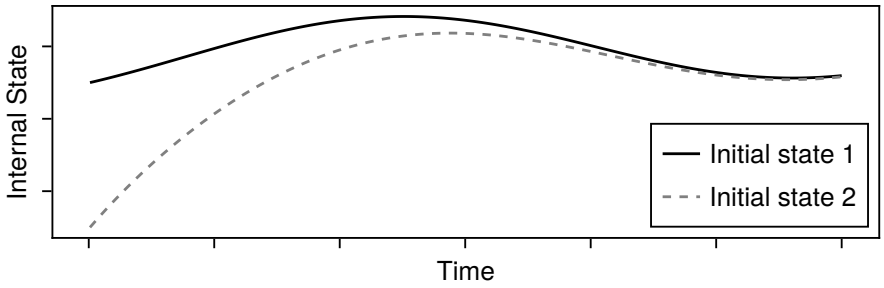



**Figure 2.3:** Contracting systems exponentially forget their initial states.

including the policy as a certified component in a bigger system; and for smooth behaviour in end-to-end optimisation pipelines.

Formally, we define a contracting system as follows. Consider a discrete-time, nonlinear dynamical system

$$x_{t+1} = F(x_t, u_t) \tag{2.8}$$

with internal state $x_t \in \mathbb{R}^{n_x}$, inputs $u_t \in \mathbb{R}^{n_u}$, and $F$ locally Lipschitz.

**Definition 2.1** (Contraction)**.** *The system* (2.8) *is said to be* contracting *(incrementally exponentially stable) with contraction rate* $\alpha \in [0, 1)$ *and overshoot* $\beta \in \mathbb{R}^+$ *if for any two initial states* $x_0^a, x_0^b \in \mathbb{R}^{n_x}$ *given the same input sequence* $u \in \ell^{n_u}$*, the corresponding state trajectories* $x^a, x^b \in \ell^{n_x}$ *satisfy*

$$|x_t^a - x_t^b| \leq \beta\alpha^t |x_0^a - x_0^b| \quad \forall\, t \in \mathbb{N}. \tag{2.9}$$

The above definition implies that contracting systems forget their initial states exponentially (Figure 2.3). This is a useful property in the context of feedback control. For example, if we can design a controller which (provably) achieves contracting closed loops with (2.8), then we can be sure that the system will be stable from *any* initial state. This is just one of the many useful properties of contracting systems. The following are all properties that naturally occur for contracting systems with a given (bounded) input sequence $u$ [98]:

- Under the mild assumption that there exists a single bounded trajectory of the system, then all state trajectories of the system are bounded.
- A convex contraction region contains at most one equilibrium point. Hence, if the input $u_t$ is constant for all $t \in \mathbb{N}$, all trajectories of the system exponentially converge to a unique equilibrium point.
- If $u$ is periodic, the state trajectories of (2.8) exponentially converge to a periodic sequence with the same fundamental frequency as the input.

Moreover, contracting systems are naturally robust to changes in their inputs $u$. Specifically, one can show that if $F$ is Lipschitz with respect to the inputs $u$, then bounded changes to the inputs of (2.8) result in bounded changes to the state trajectories, and exponentially converging inputs result in exponentially converging state trajectories. We formalise these properties in the following lemmas.

**Lemma 2.1.** *Suppose* (2.8) *is contracting with rate* $\alpha$ *and overshoot* $\beta$*, assume* $F$ *is Lipschitz continuous w.r.t.* $u$ *with Lipschitz constant* $\gamma$*, and consider any two input sequences* $u, u^\star \in \ell$*. Then the corresponding state trajectories* $x, x^\star \in \ell$ *satisfy*

$$\|x - x^\star\|_p \leq \beta C|x_0 - x_0^\star| + \frac{\gamma_u \beta}{1-\alpha}\|u - u^\star\|_p \tag{2.10}$$

*for* $p \in [1, \infty)$*, where* $C$ *is a constant that depends on the contraction rate* $\alpha$*.*

**Lemma 2.2.** *Suppose* (2.8) *is contracting with rate* $\alpha$ *and overshoot* $\beta$*, assume* $F$ *is Lipschitz continuous w.r.t.* $u$ *with Lipschitz constant* $\gamma$*, and consider any two exponentially converging input sequences* $u, u^\star \in \ell$ *such that* $|u_t - u_t^\star| \leq \bar{u}\epsilon^t$ *for some* $\bar{u} \in \mathbb{R}^+$ *and* $\epsilon \in [0, 1)$*. Then the corresponding state trajectories* $x, x^\star \in \ell$ *satisfy*

$$|x_t - x_t^\star| \leq \beta(|x_0 - x_0^\star| + \bar{u}\bar{C})\rho^t, \quad \forall t \in \mathbb{N} \tag{2.11}$$

*for some* $\rho \in [0, 1)$ *and* $\bar{C} \in \mathbb{R}^+$ *depending on* $\alpha, \varepsilon, \gamma$*.*

These two lemmas describe well-known properties of contracting systems. Similar statements appear in [98, Sec. 3.7, property (vii)], [173, Thm. 2.8], and [157, Thm. 4], respectively. We provide self-contained proofs in Appendix A.1 for completeness, and use the specific statements of Lemmas 2.1 and 2.2 to prove later results in Chapter 5.

Over the last two decades, contraction has seen a rise in popularity due to its many attractive properties. Some notable examples include: nonlinear control design via convex optimisation [104, 105, 182]; learning-based stability analysis and control [173, 172]; robot motion planning [153, 171, 191]; and as an analysis tool in neuroscience [89, 28]. For a comprehensive overview of contraction theory and its applications, we refer the reader to: the seminal paper on contraction theory [98]; a recent tutorial paper [173]; lecture notes introducing the topic for simplified settings [24]; connections between contraction and other definitions of nonlinear stability [169]; and an opinion paper on the benefits of contraction theory for control, optimisation, and learning [34]. We further provide a geometric interpretation of contraction theory in Appendix A.1.

### 2.2.2 Incremental IQCs

Contraction describes the internal stability of a dynamical system given a particular input sequence. We also need to define the robustness of a system to changes in its inputs. For this, we use incremental IQCs [108].

Let us first add an output map to the discrete-time nonlinear system (2.8) such that

$$\begin{aligned} x_{t+1} &= F(x_t, u_t) \\ y_t &= H(x_t, u_t) \end{aligned} \tag{2.12}$$

where $y_t \in \mathbb{R}^{n_y}$ is the system output and $F$, $H$ are locally Lipschitz. For the same reasons as discussed with regards to contraction analysis, we are interested in *incremental* definitions of robustness (i.e., concerning differences between trajectories) so that our robustness metrics are suitable for machine-learning applications. We therefore introduce the following definition [108].

**Definition 2.2** (Incremental IQC)**.** *The system* (2.12) *is said to admit the* incremental Integral Quadratic Constraint (IQC) *defined by* $(Q, S, R)$ *where* $Q = Q^\top \in \mathbb{R}^{n_y \times n_y}$, $S \in \mathbb{R}^{n_u \times n_y}$, $R = R^\top \in \mathbb{R}^{n_u \times n_u}$ *if for any two initial states* $x_0^a, x_0^b \in \mathbb{R}^{n_x}$ *and input sequences* $u^a, u^b \in \ell$*, the corresponding output sequences* $y^a, y^b \in \ell$ *satisfy*

$$\sum_{t=0}^{T} \begin{bmatrix} \Delta y_t \\ \Delta u_t \end{bmatrix}^\top \begin{bmatrix} Q & S^\top \\ S & R \end{bmatrix} \begin{bmatrix} \Delta y_t \\ \Delta u_t \end{bmatrix} \geq -\kappa(x_0^a, x_0^b) \quad \forall T \in \mathbb{N} \tag{2.13}$$

*for some function* $\kappa(x_0^a, x_0^b) \geq 0$ *with* $\kappa(x_0, x_0) = 0$*, and where* $\Delta y_t = y_t^b - y_t^a$, $\Delta u_t = u_t^b - u_t^a$*. We call such a system* $(Q, S, R)$*-dissipative.*

Incremental IQCs provide a very general and powerful framework for the analysis of nonlinear systems [108, 29], optimisation algorithms [96, 95], neural networks [133, 123], and many other complex dynamical systems. Definition 2.2 also includes many common definitions of incremental input-output stability and robustness as special cases (e.g., incremental passivity [175]). One particularly important special case for this thesis is a Lipschitz system, which is a $(Q, S, R)$-dissipative system with $Q = -\frac{1}{\gamma}$, $S = 0$, $R = \gamma I$ for some $\gamma \in \mathbb{R}^+$.

**Definition 2.3** (Lipschitz system)**.** *The system* (2.12) *is said to be* Lipschitz *(i.e., has a finite incremental* $\ell_2$*-gain bound) if there exists a* $\gamma^\star \in \mathbb{R}^+$ *such that*

$$\|\Delta y\|_T \leq \gamma^\star \|\Delta u\|_T + \kappa(x_0^a, x_0^b) \quad \forall T \in \mathbb{N} \tag{2.14}$$

*with* $\kappa(\cdot, \cdot), \Delta y, \Delta u$ *as defined in Definition 2.2. We call a system* $\gamma$-Lipschitz *if* (2.14) *holds for a particular* $\gamma \in \mathbb{R}^+$ *with* $\gamma \geq \gamma^\star$*.*

The Lipschitz bound plays an important role in quantifying the robustness of dynamical systems to input perturbations. Systems with small Lipschitz bounds are "smooth" in the sense that small changes to their inputs do not induce large variations in their outputs. The smaller we make $\gamma$, the smoother the response to input perturbations. The Lipschitz bound is a rather natural measure of smoothness for static functions,

since it is exactly the maximum slope of the function (for scalar functions) or the largest singular value of the Jacobian (for multi-variate functions). For dynamical systems, the connection between smoothness and robustness is perhaps most intuitive for LTI systems. In this case, the Lipschitz bound is exactly the $\mathcal{H}_\infty$ norm of the system, which is so often used as a measure of robustness in classical control theory [42, 196]. We therefore use the Lipschitz bound as a generalised notion of robustness to input perturbations for nonlinear dynamical systems throughout this thesis.

Note that Definition 2.2 only describes static incremental IQCs. We will extend it to the more general case of frequency-weighted IQCs [108] in Chapter 5.

## 2.3 Robust Neural Networks

We saw in Section 2.1.3 that using black-box neural networks to parametrise feedback policies can cause unstable closed-loop behaviour. Having defined our specific notions of stability and robustness in Section 2.2, we are now ready to introduce the main tool used in thesis to combat this limitation – stable and robust neural networks (*a.k.a* behaviourally-constrained neural networks [132]). We will see in Chapters 4 and 5 that, with certain policy architectures and assumptions on the system to be controlled, the closed-loop system can inherit the stability and robustness properties of a behaviourally-constrained neural network.

This section presents an overview of the two robust neural network architectures which we will use for numerical experiments (Chapters 3 to 5) and as a point of comparison (Chapter 6) throughout this thesis:

1. Lipschitz-bounded Sandwich networks [181].

2. Robust Recurrent Equilibrium Networks (RENs) [136].

Sandwich networks and RENs are designed to automatically satisfy user-defined stability (contraction) and robustness (incremental IQC) constraints by construction –

i.e., there is no need to impose these properties as constraints during training via computationally-expensive projection methods or with specialised cost functions. This is achieved via *direct parametrisations*, which are surjective, differentiable mappings from a set of learnable parameters to the network weights. Both network architectures are also highly expressive (i.e., not restrictive) in that they contain many existing neural network architectures as special cases.

### 2.3.1 Direct Parametrisations of Neural Networks

The key feature in the construction of the robust neural networks in [181, 136] is a direct parametrisation of the learnable parameters. Put simply, we say a neural network is directly parametrised if its learnable parameters live in $\mathbb{R}^N$. This is the case for standard, unconstrained neural networks. For example, an $L$-layer feedforward DNN (2.6) has learnable parameters $\varphi := \{(W_k, b_k)\}_{0 \leq k \leq L} \in \mathbb{R}^{n_\varphi}$. However, imposing constraints on the stability and robustness of a neural network restricts the set of possible weights and biases. Continuing with the example above, we would have

$$\varphi \in \{\varphi \mid \text{constraints are satisfied}\} \subseteq \mathbb{R}^{n_\varphi}$$

which may be a (highly) non-convex set.

Training models whose learnable parameters need to be restricted to a non-convex set can be computationally challenging, and in some cases infeasible. Instead, we can introduce a new set of learnable parameters $\theta \in \Theta$ and a *parametrisation* $\varphi = \mathcal{M}(\theta)$ which maps $\theta \mapsto \varphi$ such that the weights and biases $\varphi$ satisfy the constraints, but $\Theta$ is easier to work with. We formalise the notion of a neural-network parametrisation in the following definition [181, Def. 2.3].

**Definition 2.4.** *A parametrisation of a neural network is a differentiable mapping $\varphi = \mathcal{M}(\theta)$ from the learnable parameters $\theta \in \Theta \subseteq \mathbb{R}^N$ to the weights and biases $\varphi$. It is a* convex parametrisation *if $\Theta$ is convex, and a* direct parametrisation *if $\Theta = \mathbb{R}^N$.*

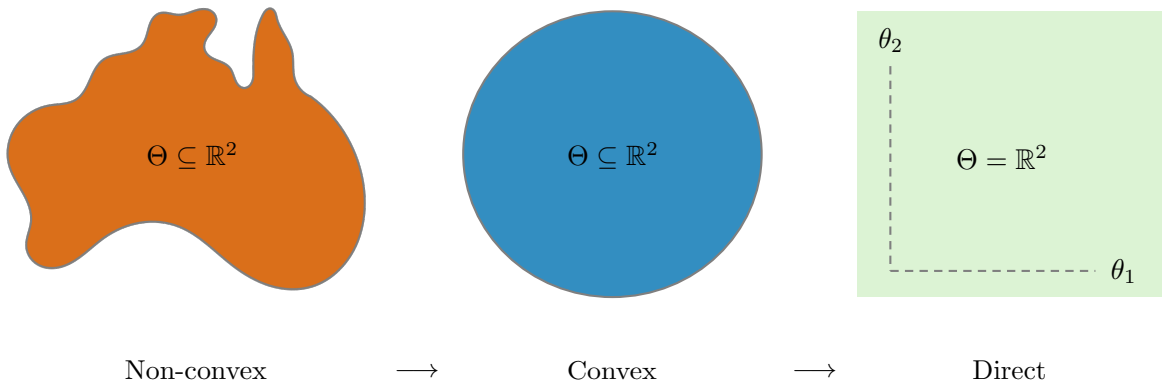



**Figure 2.4:** Illustration of different types of parametrisation for neural networks. The learnable parameters $\theta \in \Theta \subseteq \mathbb{R}^N$ of a neural network can live in a non-convex set (left), a convex set (middle), or $\mathbb{R}^N$ itself (right). When $\Theta = \mathbb{R}^N$, we call this a direct parametrisation. We use $N = 2$ here for illustrative purposes.

Different classes of the parameter set $\Theta$ are illustrated in Figure 2.4. It is argued in [181, 136] that direct parametrisations should be the gold standard for neural-network parametrisations due to their compatibility with common optimisation tools in machine learning. Most practitioners train neural networks with variants of unconstrained gradient descent which assume $\Theta = \mathbb{R}^N$ [88, 159, 143]. When $\Theta \subset \mathbb{R}^N$, even if $\Theta$ is convex, we cannot directly apply standard optimisation tools. Instead, we must use a restricted class of methods which are either tailored to particular structures of $\Theta$ (such as optimising over particular manifolds [21]), or involve computationally-expensive projection steps (e.g., [123, 134, 80]). In contrast, if we have a direct parametrisation such that $\Theta = \mathbb{R}^N$, we can immediately use standard optimisation tools, making the parametrisation "plug-and-play" with existing learning pipelines which do not include behavioural constraints. Note that Definition 2.4 specifies that a direct parametrisation should be differentiable. This is important for it to be compatible with smooth optimisation tools and automatic differentiation.

In the following subsections, we will look at two cases where direct parametrisations make it feasible to design plug-and-play neural network architectures with stability and robustness constraints.

### 2.3.2 Lipschitz-Bounded Sandwich Networks

The Lipschitz bound, introduced in Definition 2.3, is a natural robustness metric for static neural networks since it describes the sensitivity of the model to small changes in its inputs. However, computing the exact Lipschitz constant of a DNN (2.6) is NP-hard [177], which makes parametrising Lipschitz-Bounded Deep Networks (LBDNs) a challenging problem. Instead, the best we can do is to compute a tight upper bound on the Lipschitz constant.

The main contribution of [181] was deriving a direct parametrisation for LBDNs called the *Sandwich network* which satisfies the tightest-known bounds on the Lipschitz constant of a DNN, proposed in [44]. The bounds in [44] were derived by leveraging the incremental IQC framework [108] and by assuming that the activation function $\sigma$ satisfies the following assumption.

**Assumption 2.1.** *The activation function $\sigma$ is piecewise differentiable and slope-restricted in $[0, 1]$. That is, it satisfies the incremental sector bound*

$$0 \leq \frac{\sigma(x) - \sigma(y)}{x - y} \leq 1 \quad \forall\, x, y \in \mathbb{R} \text{ with } x \neq y. \tag{2.15}$$

This assumption holds for many common activation functions including the ReLU, tanh, and sigmoid functions, as depicted in Figure 2.5.

The Sandwich network [181] is a $\gamma$-Lipschitz, $L$-layer feedforward DNN with the par-

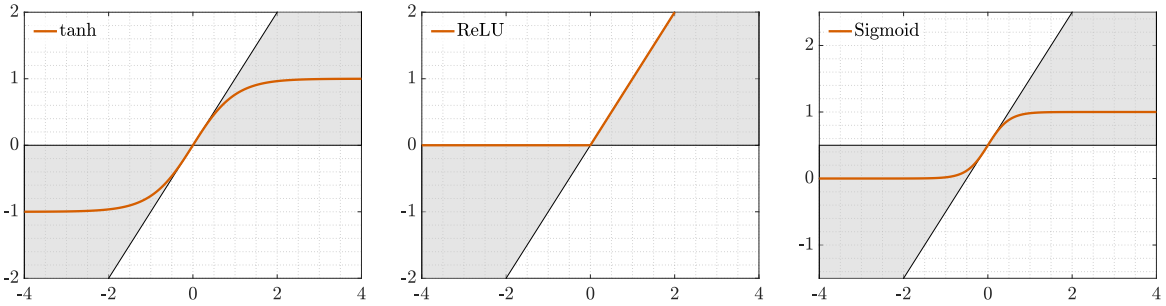



**Figure 2.5:** Common activation functions satisfying assumption 2.1.

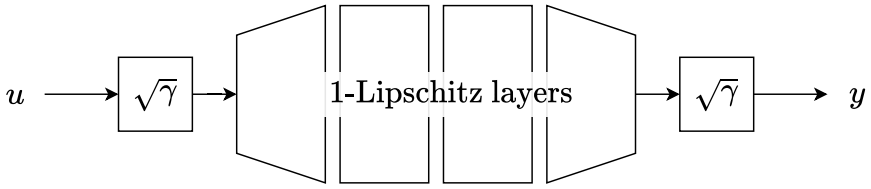



**Figure 2.6:** $\gamma$-Lipschitz deep networks like the Sandwich network are constructed by concatenating 1-Lipschitz layers and multiplying by $\sqrt{\gamma}$ at the network inputs and outputs.

ticular structure

$$h_0 = \sqrt{\gamma} u \tag{2.16a}$$

$$h_{k+1} = \sqrt{2} A_k^\top \Psi_k \sigma(\sqrt{2} \Psi_k^{-1} B_k h_k + b_k), \qquad k = 0, \ldots, L-1 \tag{2.16b}$$

$$y = \sqrt{\gamma} B_L h_L + b_L, \tag{2.16c}$$

where $h \in \mathbb{R}^{n_k}$ is the hidden unit of the $k^{\text{th}}$ layer, the weight matrices are $A_k \in \mathbb{R}^{n_{k+1} \times n_{k+1}}, B_k \in \mathbb{R}^{n_{k+1} \times n_k}, \Psi_k \in \mathbb{R}^{n_{k+1} \times n_{k+1}}$, the bias vector for the $k^{\text{th}}$ layer is $b_k \in \mathbb{R}^{n_{k+1}}$, and the activation function $\sigma$ satisfies assumption 2.1. The main component is the so-called *Sandwich layer* (2.16b), which is designed to be 1-Lipschitz. One can think of a $\gamma$-Lipschitz Sandwich network (2.16) as a series interconnection of Sandwich layers with a multiplier of $\sqrt{\gamma}$ on the inputs and outputs such that the total upper bound on the Lipschitz constant is $\gamma$. This is illustrated in Figure 2.6.

The Sandwich layer was directly parametrised in [181] to be 1-Lipschitz by construction. We review the parametrisation below. The learnable (direct) parameters in a Sandwich network are

$$\theta = \{d_j \in \mathbb{R}^{n_j},\ X_k \in \mathbb{R}^{n_{k+1} \times n_{k+1}},\ Y_k \in \mathbb{R}^{n_k \times n_{k+1}}, b_k \in \mathbb{R}^{n_{k+1}}\}_{0 \le j < L, 0 \le k \le L} \tag{2.17}$$

hence $\theta \in \Theta = \mathbb{R}^N$. The direct parametrisation mapping $\theta$ to the weights and biases in (2.16) is described by the following transformations:

$$\Psi_k = \operatorname{diag}\left(e^{d_k}\right), \qquad \begin{bmatrix} A_k^\top \\ B_k^\top \end{bmatrix} = \operatorname{Cayley}\left(\begin{bmatrix} X_k \\ Y_k \end{bmatrix}\right), \tag{2.18}$$

where the Cayley transform is defined such that for any $X \in \mathbb{R}^{m\times m}, Y \in \mathbb{R}^{n\times m}$ we have $Q^\top Q = I$ if

$$Q = \text{Cayley}\left(\begin{bmatrix} X \\ Y \end{bmatrix}\right) := \begin{bmatrix} (I+Z)^{-1}(I-Z) \\ -2Y(I+Z)^{-1} \end{bmatrix} \tag{2.19}$$

with $Z = X - X^\top + Y^\top Y$. This parametrisation of 1-Lipschitz Sandwich layers is a highly expressive parametrisation. Indeed, [181] show that it contains all 1-Lipschitz linear layers as a special case. We will explore the benefits of expressive parametrisations of LBDNs further in Chapter 3.

The definition of the Sandwich network in (2.16) has a similar structure to that of a feedforward DNN in (2.6). In fact, (2.16) is a particular parametrisation of (2.6) if we define the hidden units and weight matrices as

$$\begin{aligned} h_k &= \sqrt{2}A_{k-1}^\top \Psi_{k-1} w_k, \\ W_k &= 2\Psi_k^{-1} B_k A_{k-1}^\top \Psi_{k-1}, \end{aligned}$$

respectively. Just like with classic DNNs (2.6), the Sandwich network (2.16) includes both MLPs as well as other feedforward architectures like CNNs. From a mathematical perspective, their treatment is the same. From an implementation perspective, see [181, Algorithm 1] for details on convolutional Sandwich layers, and recent work in [125] for a new and more efficient parametrisation of Lipschitz-bounded CNNs. We will study the empirical benefits of both Lipschitz-bounded MLPs and CNNs constructed from Sandwich layers in Chapter 3.

### 2.3.3 Robust Recurrent Equilibrium Networks

Sandwich networks are a special class of robust neural network with a static, feedforward structure and a prescribed upper bound on their Lipschitz constant. We now introduce a more general model class that allows for recurrent networks with guarantees of contraction (internal stability) and $(Q, S, R)$-dissipativity (input-output robustness)

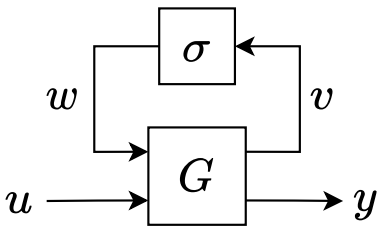



**Figure 2.7:** Block diagram for a REN. The system $\boldsymbol{G}$ is described by (2.20a).

called the Recurrent Equilibrium Network (REN) [135, 136].

RENs take the form of a classical Lur'e system, which is the feedback interconnection of an LTI (in this case affine) system and a static nonlinearity. This is illustrated in Figure 2.7. We can express a REN as a discrete-time dynamical system of the form

$$\begin{bmatrix} x_{t+1} \\ v_t \\ y_t \end{bmatrix} = \overbrace{\left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right]}^{W} \begin{bmatrix} x_t \\ w_t \\ u_t \end{bmatrix} + \overbrace{\begin{bmatrix} b_x \\ b_v \\ b_y \end{bmatrix}}^{b} \tag{2.20a}$$

$$w_t = \sigma(v_t), \tag{2.20b}$$

where $u_t \in \mathbb{R}^{n_u}, x_t \in \mathbb{R}^{n_x}, y_t \in \mathbb{R}^{n_y}$ are the input, internal state, and output of the REN at some time $t \in \mathbb{N}$, $\sigma$ is a scalar activation function satisfying assumption 2.1, and $W, b$ are the weights and biases of the network, respectively.

### Expressivity of the REN Model Class

The main component of a REN is its *equilibrium layer* – i.e., (2.20b) together with the middle line of (2.20a) – which is described by

$$w_t = \sigma(D_{11}w_t + b_{w_t}) \tag{2.21}$$

where $w \in \mathbb{R}^{n_v}$ and $b_{w_t} := C_1 x_t + D_{12}u_t + b_v$. Equilibrium networks built from layers (2.21) are themselves a widely-studied form of recurrent neural architecture [188, 133, 43]. Their name comes from the fact that at each time $t \in \mathbb{N}$, solutions to

(2.21) are equilibrium solutions of the discrete-time system $w_t^{k+1} = \sigma(D_{11}w_t^k + b_{w_t})$, or alternatively the Ordinary Differential Equation (ODE)

$$\frac{d}{ds}w_t(s) = -w_t(s) + \sigma(D_{11}w_t(s) + b_{w_t}).$$

If $D_{11} = 0$, the equilibrium layer reduces to $w_t = \sigma(b_{w_t})$ and the REN (2.20) describes a single-layer RNN. If, however, $D_{11} \neq 0$, then the equilibrium layer is non-trivial and the REN (2.20) describes a rich class of multi-layer networks, making the REN quite an expressive model class.

We illustrate the expressive power of RENs with non-zero $D_{11}$ by showing that the feedforward DNN structure (2.6) is contained within the REN model class. We repeat (2.6) below for convenience, and illustrate the process in Figure 2.8.

$$\begin{aligned} w_0 &= u, \\ w_{k+1} &= \sigma(W_k w_k + b_k), \quad k = 0, \dots, L-1 \\ y &= W_L w_L + b_L. \end{aligned} \tag{2.6}$$

Take the DNN (2.6), then pull apart the weights and activations to group the linear and nonlinear terms, respectively. We can then write it in the form of a Lur'e system with static nonlinearity $w = \sigma(v)$ and linear component

$$\begin{aligned} v &= D_{11}w + D_{12}u + b_v \\ y &= D_{21}w + b_y \end{aligned} \tag{2.22}$$

where $b_y = b_L$ and

$$b_v = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{L-1} \end{bmatrix}, \; D_{11} = \begin{bmatrix} 0 & & & \\ W_1 & \ddots & & \\ \vdots & \ddots & 0 & \\ 0 & \cdots & W_{L-1} & 0 \end{bmatrix}, \; D_{12} = \begin{bmatrix} W_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \; D_{21} = \begin{bmatrix} 0 & \cdots & 0 & W_L \end{bmatrix}. \tag{2.23}$$

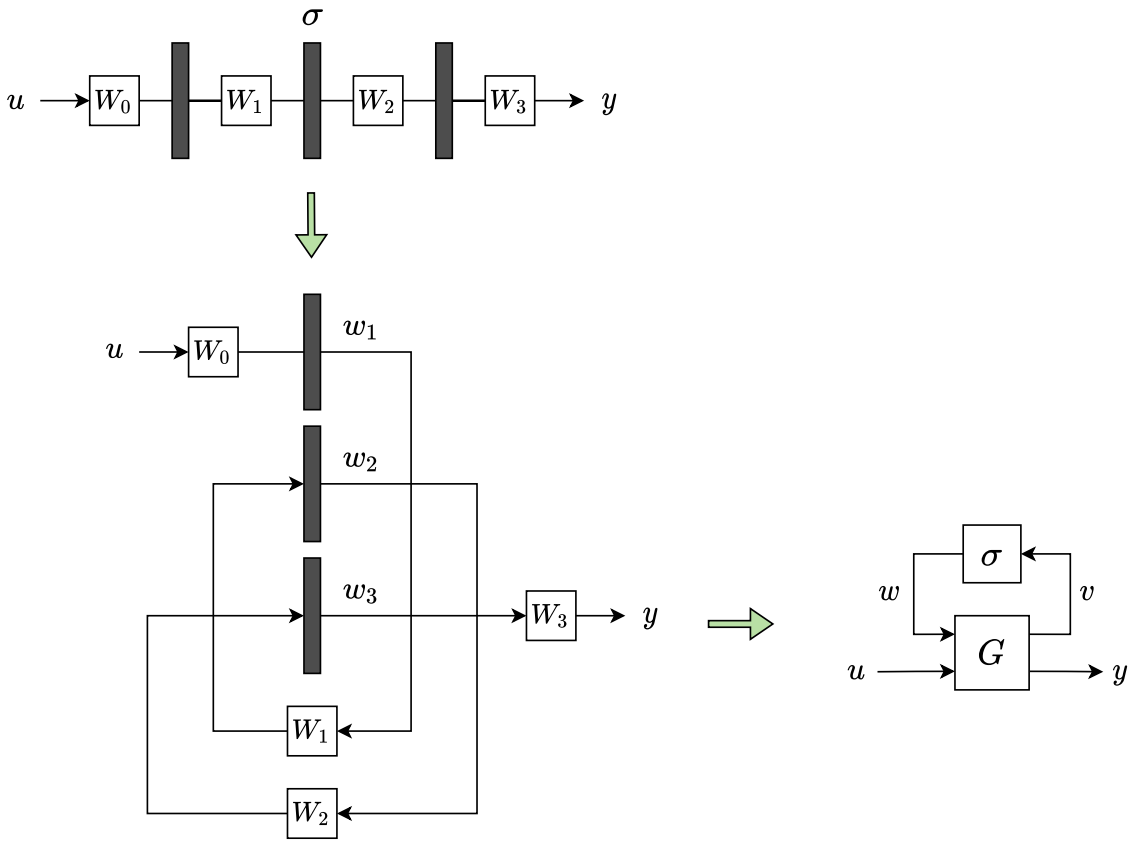



**Figure 2.8:** Illustration of how a DNN can be written in the form of a Lur'e system (and therefore a REN). Simply group the activations and weight matrices to form interconnected nonlinear and linear components $\sigma$ and $\boldsymbol{G}$, respectively. The system $\boldsymbol{G}$ is described by (2.22). Bias terms are omitted for illustrative purposes. Figure adapted from [181].

This is now a REN (2.20) with $A, B_1, B_2, C_1, D_{22}, b_x = 0$. Note that if $D_{11} = 0$, this is simply a single-layer DNN. However, with $D_{11}$ as in (2.23), we can represent $L$-layer feedforward networks. Visually, one can think of the equilibrium layer as a "stack" of nonlinear activations, as depicted in Figure 2.8.

Similar manipulations [136] can be performed to show that a number of other common neural network architectures and model classes are special cases of RENs, including:

1. Linear state-space models, which are essential for linear modelling and control [64] and also for large-scale sequence modelling in machine learning [58, 59, 57, 146]. Choose all weights and biases as zero in (2.20) except $A, B_2, C_2, D_{22}$.

2. RNNs (2.7), which are the classic recurrent neural architecture. Choose all

weights and biases as zero in (2.20) except $C_1, C_2, D_{12}, b_x, b_y$.

3. Static equilibrium networks (2.21), which themselves have been shown to contain a large class of feedforward network architectures including DNNs and residual networks [188, 133, 43]. Choose all weights and biases as zero in (2.20) except $D_{11}, D_{12}, b_v$.

4. Block-structured models such as Wiener-Hammerstein or Hammerstein-Wiener models [140]. Choose $B_1, C_2 = 0$ or $B_2, C_1 = 0$ in (2.20) (respectively).

Despite the remarkable expressive power offered by the equilibrium layer (2.21), having non-zero $D_{11}$ comes at a price. Firstly, one must impose restrictions on $D_{11}$ to ensure that the equilibrium layer is well-posed (i.e., there is a unique solution to (2.21) for a given $D_{11}, b_{w_t}$). Conditions to ensure this are provided in [136]. Secondly, we have to actually solve the equilibrium layer (2.21) to compute $w_t$ for every time-step at which the model is evaluated. Since $w_t$ is the solution to a fixed-point equation, (2.21) must be solved iteratively. There are two cases considered in [136]:

- When $D_{11}$ is dense, one can solve (2.21) with operator-splitting methods [133]. However, this is computationally expensive and can be prohibitively slow for training and inference in moderate- to large-scale models.

- In the special case that $D_{11}$ is lower-triangular, one can explicitly solve (2.21) row-by-row, which still requires iteration but can be implemented with a simple `for` loop, making training and inference of moderately-sized models feasible on standard Central Processing Unit (CPU) and GPU architectures.

RENs with lower-triangular $D_{11}$ are referred to as *acyclic* RENs in [136]. Moreover, it was shown in [136] that acyclic RENs achieve similar performance to RENs with dense $D_{11}$ on common machine-learning tasks. Therefore, for the remainder of this thesis, we only consider acyclic RENs with lower-triangular $D_{11}$.

### Contracting and $(Q, S, R)$-Dissipative RENs

The main contribution of [136] was deriving a direct parametrisation of all RENs (2.20) that are contracting and $(Q, S, R)$-dissipative in the sense of Definitions 2.1 and 2.2, respectively. The authors follow a similar strategy to the parametrisation of Sandwich networks outlined in Section 2.3.2. The basic idea is to construct Linear Matrix Inequality (LMI) conditions for the weights $W$ of a REN (2.20) to be contracting and Lipschitz, which take the form

$$H(W) \succ 0$$

for some matrix $H \in \mathbb{R}^{n_H \times n_H}$ which depends on the components of the weight matrix $W$ from (2.20a). The direct parametrisation is then constructed by introducing a free variable $X^{n_H \times n_H}$ and a small positive scalar $\epsilon \in \mathbb{R}^+$ to parametrise $H$ via

$$H = X^\top X + \epsilon I.$$

The weights $W$ satisfying the LMI are then directly constructed from the elements of $H$. We leave specific details on the direct parametrisation of RENs to Chapter 6 and refer the reader to [136, Sec. V] for a complete overview.

In recent years, several other methods have been proposed which also impose constraints on the internal stability [106, 134, 117] and input-output robustness [170, 6, 125] of neural networks. The advantage of RENs is that they are the most general of these model classes, and include many as special cases. Furthermore, contracting RENs were shown to be universal approximators of *all* contracting and Lipschitz nonlinear systems in [178, Prop. 2].

It should be noted that despite their many advantages, RENs are by no means the perfect solution to learning stable and robust nonlinear models. Despite the expressivity of the unconstrained REN model class (2.20), it is challenging to obtain direct parametrisations of contracting and $(Q, S, R)$-dissipative RENs whose weight matrices

require particular structures – for example, convolutional RENs. Moreover, the size of the equilibrium layer (2.21) scales quadratically with the number of neurons $n_v$ in the REN, making RENs prohibitively slow for training and inference when very large models are required, such as in deep RL applications with image or video data. We will address these limitations further in Chapter 6.

## 2.4 Numerical Methods in Deep RL

The direct parametrisations of Sandwich networks and RENs allow us to train robust neural models with standard, unconstrained optimisation tools. We will use both network architectures to parametrise feedback control policies in numerical examples throughout Chapters 3 to 6. Since it is the motivating problem setting for this thesis, the majority of these examples involve learning controllers via deep RL. We therefore conclude this chapter with an overview of the two RL algorithms applied in this thesis: Analytic Policy Gradients (APG) and Proximal Policy Optimisation (PPO) [143].

In Section 2.1.1, we formulated the problem of learning neural feedback policies $\mathcal{K}_\theta$ as choosing model parameters $\theta$ to minimise an estimate $\hat{J}(\theta)$ of an expected cost $J(\theta)$ over a distribution of initial system states and disturbances. Broadly-speaking, RL algorithms for solving this problem can be divided into two categories: value-based methods, where the control policy is decided by learning a neural value function and using it to select optimal control actions; and policy optimisation methods, where a neural feedback policy is learned directly. We focus on the latter approach in this section, as policy optimisation is most commonly used in scenarios with continuous measurement and control spaces such as robotics [97, 138, 83].

### 2.4.1 Analytic Policy Gradients

Given a dynamical system (2.1), the simplest approach to policy optimisation is to run SGD (2.5) – or variants such as Adam [88] – on the approximate cost (2.4) until

convergence. We repeat these equations below for convenience:

$$\theta^{k+1} \leftarrow \theta^k - \alpha \nabla_\theta \hat{J}(\theta) \tag{2.5}$$

where
$$\hat{J}(\theta) = \frac{1}{M} \sum_{m=0}^{M-1} \left[ \sum_{t=0}^{T-1} g(x_t^m, u_t^m) + g_f(x_T^m) \right]. \tag{2.4}$$

Solving for the policy parameters $\theta$ in this way is referred to as APG, or first-order policy gradients, in the RL literature [100, 184]. The term "analytic" comes from the underlying assumption that we have accurate gradient information for the whole closed-loop system – that is, the cost function, the dynamics, and the policy itself. To see this, consider even the simple case where the measurement is $y_t = x_t$ such that $u_t = \mathcal{K}_\theta(x_t)$. Then at each time-step, the cost gradient is

$$\frac{\partial g}{\partial \theta}(x_t, u_t) = \frac{\partial g}{\partial x}\frac{\partial x_t}{\partial \theta} + \frac{\partial g}{\partial u}\left( \frac{\partial \mathcal{K}_\theta}{\partial \theta} + \frac{\partial \mathcal{K}_\theta}{\partial x}\frac{\partial x_t}{\partial \theta} \right)$$

where
$$\frac{\partial x_t}{\partial \theta} = \frac{\partial f}{\partial x}(x_{t-1}, u_{t-1})\frac{\partial x_{t-1}}{\partial \theta} + \frac{\partial f}{\partial u}(x_{t-1}, u_{t-1})\frac{\partial u_{t-1}}{\partial \theta},$$

which depends on gradients of $g$, $f$, and $\mathcal{K}_\theta$. This is increasingly a reasonable assumption thanks to the rise of differentiable programming libraries [22] and languages [19], and mature implementations of differentiable physics simulators [167, 46, 111] which are used to train RL policies for real-world robotic systems [138, 100, 25].

In practice, APG is typically implemented by splitting up the training horizon of $T$ steps in (2.4) into multiple smaller time windows of length $\bar{T}$ [100]. One then performs SGD on each window in a process akin to multiple shooting in trajectory optimisation [93]. That is, instead of running a single step of SGD on (2.4) over a horizon $T$, run it $T/\bar{T}$ times on

$$\bar{J}_i(\theta) = \frac{1}{M} \sum_{m=0}^{M-1} \left[ \sum_{t=i\bar{T}}^{(i+1)\bar{T}-1} g(x_t^m, u_t^m) + g_f(x_{\bar{T}}^m) \right] \tag{2.24}$$

where $i = 0, \ldots, (T/\bar{T} - 1)$. We will employ this approach to APG in numerical examples throughout Chapters 5 and 6.

### 2.4.2 Policy Gradient Algorithms and PPO

Despite the rise of differentiable physics simulators, there remain many situations in which we cannot, or may not want to, compute gradients of the dynamics. A classic example is the suite of Atari video games [113], where the dynamics of each game environment is defined by a set of discrete rules. Even in scenarios where the gradients can be computed, the quality of numerical gradient estimates in highly-discontinuous tasks (e.g., robotic locomotion and manipulation) is often questionable [158], and differentiating through an entire closed-loop physics simulator can be computationally expensive. Instead, most common policy optimisation algorithms in RL approximate the cost gradient using the so-called policy gradients approach [159]. In this subsection, we introduce the policy gradients method and one of the more popular algorithms – PPO [143].

We briefly consider RL from a probabilistic perspective to introduce policy gradient algorithms. To simplify notation, assume that the full state of the system (2.1) is measurable ($y_t = x_t$), there are no disturbances ($w, v = 0$), and control policies $u_t = \mathcal{K}_\theta(x_t)$ are static functions. We first introduce the notion of a *policy distribution* $\pi_\theta(u_t \mid x_t)$ from which we sample control actions $u_t$ given a state $x_t$. If we already have a deterministic controller, we can easily construct a corresponding policy distribution by adding Gaussian noise to the control output

$$u_t = \mathcal{K}_\theta(x_t) + n_t, \qquad n_t \sim \mathcal{N}(0, \nu^2)$$

with variance $\nu^2$ so that the policy distribution is

$$\pi_\theta(u_t \mid x_t) = \frac{1}{\nu\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{u_t - \mathcal{K}_\theta(x_t)}{\nu}\right)^2\right].$$

We then re-write the cost function (2.2) as

$$J(\theta) = \mathbb{E}_{p(\tau|\theta)}\left[R(\tau)\right]$$

where the expectation is over the probability distribution $p(\tau \mid \theta)$ of trajectories $\tau := (u_0, \ldots, u_{T-1}, x_0, \ldots, x_T)$ given $\theta$, and $R(\tau) := \sum_{t=0}^{T-1} g(x_t, u_t) + g_f(x_T)$. The key step in deriving policy gradient algorithms is to compute the gradient of $J(\theta)$ by applying the well-known log-likelihood trick. This approach is well-documented in works such as [185, 131], but it is a neat trick so we include it below for completeness.

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{p(\tau|\theta)} \left[ R(\tau) \right] \\
&= \nabla_\theta \int R(\tau) p(\tau \mid \theta) d\tau \\
&= \int R(\tau) \nabla_\theta p(\tau \mid \theta) d\tau \\
&= \int R(\tau) \left( \frac{\nabla_\theta p(\tau \mid \theta)}{p(\tau \mid \theta)} \right) p(\tau \mid \theta) d\tau \\
&= \int R(\tau) \nabla_\theta \ln p(\tau \mid \theta) p(\tau \mid \theta) d\tau.
\end{aligned}$$

The third line is due to the fact that $\nabla_\theta R(\tau) = 0$ since $R(\tau)$ is independent of $\theta$ by definition (i.e., it is simply the cost of a given trajectory). The log-likelihood trick allows us to transition from the fourth to the fifth line. The cost gradient can therefore be written as

$$\nabla_\theta J(\theta) = \mathbb{E}_{p(\tau|\theta)} \left[ \nabla_\theta \ln p(\tau \mid \theta) R(\tau) \right].$$

The probability of a trajectory $\tau$ given a particular vector of learnable parameters $\theta$ is

$$p(\tau \mid \theta) = \prod_{t=0}^{T-1} p(x_{t+1} \mid x_t, u_t) \pi_\theta(u_t \mid x_t),$$

where $p(x_{t+1} \mid x_t, u_t)$ describes the probability of the state transition dynamics (i.e., a probabilistic version of the function $f(x_t, u_t)$ in (2.1)). Since $p(x_{t+1} \mid x_t, u_t)$ is independent of $\theta$, then the gradient of $\ln p(\tau \mid \theta)$ is simply

$$\nabla_\theta \ln p(\tau \mid \theta) = \sum_{t=0}^{T-1} \nabla_\theta \ln \pi_\theta(u_t \mid x_t).$$

Hence, with no approximation, we arrive at

$$\nabla_\theta J(\theta) = \mathbb{E}_{p(\tau|\theta)} \left[ R(\tau) \sum_{t=0}^{T-1} \nabla_\theta \ln \pi_\theta(u_t \mid x_t) \right]. \tag{2.25}$$

Equation (2.25) shows that to compute the cost gradient $\nabla_\theta J(\theta)$, all we have to do is sample trajectories $\tau$, compute their corresponding costs $R(\tau)$, and take a gradient of the log of the policy distribution (i.e, the policy gradient). Nowhere here are we required to differentiate through the cost function itself or the dynamics of the system.

The approach used in policy gradient RL algorithms is to define a loss function $\mathcal{L}(\theta)$ whose gradient is either identical to or an unbiased estimate of (2.25). One then applies gradient descent using automatic differentiation tools. The simplest such loss function is

$$\mathcal{L}(\theta) = \mathbb{E}_{p(\tau|\theta)} \left[ R(\tau) \sum_{t=0}^{T-1} \ln \pi_\theta(u_t \mid x_t) \right], \tag{2.26}$$

which is the loss function in the classic REINFORCE algorithm [187]. However, applying gradient descent to (2.26) is known to be quite sample-inefficient in comparison to current state-of-the-art RL algorithms [5].

The perspective taken in PPO is to use a surrogate loss function which is clipped to ensure that updates to the policy parameters $\theta$ are never too aggressive, which helps with sample efficiency and reduces numerical instability during training [141, 143]. Let $\mathfrak{r}_t(\theta)$ denote the probability ratio

$$\mathfrak{r}_t(\theta) = \frac{\pi_\theta(u_t \mid x_t)}{\pi_{\theta_{\text{old}}}(u_t \mid x_t)}$$

between the policy distribution parametrised by the current $\theta$ to be updated, and the parameters $\theta_{\text{old}}$ at the previous iteration of gradient descent. The classic PPO loss function is then

$$\mathcal{L}_{\text{PPO}}(\theta) = \mathbb{E}_{p(\tau|\theta)} \left[ \sum_{t=0}^{T-1} \min(\mathfrak{r}_t(\theta)\hat{A}_t^{\pi_\theta}, \text{clip}(\mathfrak{r}_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t^{\pi_\theta}) \right] \tag{2.27}$$

where $\epsilon$ is a hyperparameter to be chosen (typically $\epsilon \approx 0.25$ [5]) and $\hat{A}_t^{\pi_\theta}$ is an estimate of the advantage function[1].

In practice, the PPO loss function combines multiple loss terms of which (2.27) is just one component (see [143] for a complete overview). Moreover, it is typically assumed that the expectation in (2.27) is approximated by sampling over batches of trajectories (as in (2.4)), and the time order of states, controls, and costs is shuffled since $\mathcal{K}_\theta$ is assumed to be a static function. These implementation details allow the PPO loss to be computed in many parallel minibatches, allowing for extremely efficient implementations of the algorithm over clusters of parallel threads. The parallelism of policy gradient algorithms like PPO is their main selling point over simpler methods like APG in scenarios where gradient information is actually available.

There are numerous mature implementations of the PPO algorithm in open-source RL libraries such as [138, 72, 46, 129]. Furthermore, there is strong empirical evidence demonstrating that PPO performs well on a wide range of deep RL tasks (see [5] for a detailed study). This, together with the efficient parallelism of the algorithm, has lead to PPO being widely adopted by the RL community, particularly for learning policy networks for feedback control of real-world robotic systems [148, 138, 83, 25].

Note that policy gradient methods like PPO can be extended to learning dynamic policies $u_t = \mathcal{K}_\theta(y_t)$ with partial state information. In this case, samples in (2.27) cannot be shuffled in time, and special care must be taken to track the state of the policy network. See [185, 71] for details on extending policy gradient algorithms to recurrent policy networks.

[1] Qualitatively, the advantage $A_t^{\pi_\theta}$ of a policy distribution $\pi_\theta(u_t \mid x_t)$ is a measure of the improvement in cost between a randomly-sampled $u_t \sim \pi_\theta$ and the mean control action of the distribution, $\mathcal{K}_\theta(x_t)$. The estimate $\hat{A}_t^{\pi_\theta}$ is typically computed via Generalised Advantage Estimation (GAE) [142]. See [142, Sec. 3] for a derivation and [5, App. A] for implementation details.

# Chapter 3

# Lipschitz-Bounded Policy Networks

The goal of this thesis is to parametrise robustly-stabilising policies for learning-based control. In this chapter, we begin our journey with an empirical study of robust policy networks in deep RL. Specifically, we investigate the benefits of replacing the black-box policy network in a deep RL controller with a Lipschitz-bounded policy network – i.e., a policy network that is directly parametrised to satisfy a user-defined bound on its Lipschitz constant. We seek to address the following questions:

1. Can Lipschitz-bounded deep networks improve the empirically-observed robustness of control policies in deep RL?

2. If so, does the policy network architecture matter? That is, do more sophisticated policy parametrisations with less-conservative Lipschitz bounds give finer control over the performance-robustness trade-off?

We provide an empirical study of Lipschitz-bounded policy networks on two representative problems: pendulum swing-up, a classical benchmark problem in control and RL; and Atari Pong, a simple proxy for vision-based autonomous decision-making tasks. To the best of our knowledge, our comparison of Lipschitz-bounded policy architectures in deep RL is the first of its kind.

### Publications

The content in this chapter previously appeared as part of the following publication:

[13] Nicholas H. Barbara, Ruigang Wang, and Ian R. Manchester, "On Robust Reinforcement Learning with Lipschitz-Bounded Policy Networks," *Symposium on Systems Theory in Data and Optimization,* 2024.

## 3.1 Introduction

We saw in Section 2.1.1 that the applicability of deep RL to performance- and safety-critical systems is currently limited by questions of its robustness [70]. This is due to the sensitivity of neural networks to small input perturbations [160], which makes policy networks learned via deep RL potentially unrobust to disturbances, noise, and targeted adversarial attacks. Despite sharing similar sensitivity issues to neural classifiers, for which many robust neural networks have recently been developed [126, 170, 181, 125], the use of robust policy architectures in deep RL has not been widely studied.

Most common approaches to improving policy robustness in deep RL are based on adversarial training [122], where adversarial attacks on a policy's inputs are optimised during training to encourage the network to perform well under perturbations. While this works well in applications where the structure of test-time perturbations is always similar to those seen during training, adversarial training only certifies a lower bound on a policy's robustness. It is therefore possible to find new, out-of-sample attacks that cause the policy to fail, such as the examples from [139, 144] in Figure 2.2. An alternative strategy is to learn control policies with a certified *upper* bound on their sensitivity using methods like randomised smoothing [91, 189] and loss-function regularisation [116, 114]. However, these methods bound the sensitivity of a learned policy during the training process, and are therefore fundamentally linked to how the it is trained. In this chapter, we instead study how, via careful choice of a policy's architecture and parameterisation, we can directly bound the sensitivity of a policy independently of how it is trained.

One promising approach is to use a directly-parametrised, Lipschitz-bounded neural network. We introduced the concepts of direct parametrisations and Lipschitz-bounded systems in Definitions 2.3 and 2.4, respectively. For a static neural network $\phi : \mathbb{R}^n \to \mathbb{R}^m$, the Lipschitz condition (2.14) is simply

$$|\phi(x_1) - \phi(x_2)| \leq \gamma |x_1 - x_2|, \quad \forall x_1, x_2 \in \mathbb{R}^n \tag{3.1}$$

for some $\gamma \in \mathbb{R}^+$. The true $\ell_2$ Lipschitz constant, denoted $\text{Lip}(\phi)$, is the smallest $\gamma$ satisfying (3.1). Despite the wealth of recent work on constructing Lipschitz-bounded deep networks [170, 181, 125, 6], to the best of our knowledge, the only method commonly used in deep RL is spectral normalisation [20, 161] which is known to give conservative bounds on the true Lipschitz constant. We therefore study the robustness benefits of different Lipschitz-bounded network parametrisations as substitutes for black-box deep RL policies.

## 3.2 Preliminaries

### 3.2.1 Problem Formulation

We introduced the problem of learning policy networks for nonlinear dynamical systems in Section 2.1. We consider the particular problem formulation depicted in Figure 3.1, where the system has a discrete-time, nonlinear state-space representation

$$x_{t+1} = f(x_t, u_t, w_t) \tag{3.2}$$

with state vector $x_t \in \mathbb{R}^{n_x}$, controlled inputs $u_t \in \mathbb{R}^{n_u}$, and disturbances $w_t \in \mathbb{R}^{n_w}$. For training, we assume that the state is completely measurable (i.e., $y_t = x_t$). We will consider perturbations to the state measurements at test time in the next subsection.

As outlined in Section 2.1, the task in deep RL is to learn feedback control policies $u_t = \mathcal{K}_\theta(x_t)$ parametrised by $\theta \in \mathbb{R}^N$ which (locally and approximately) solve Problem (2.3).

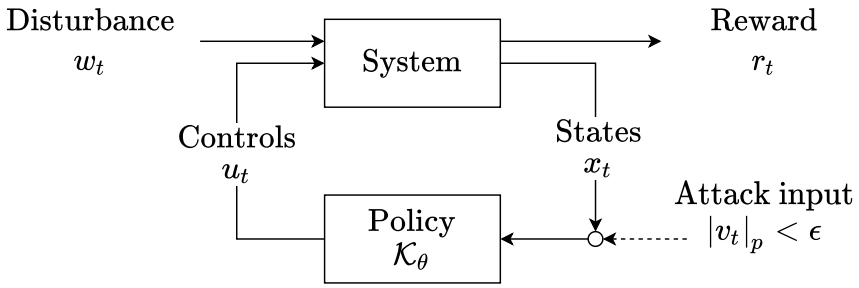



**Figure 3.1:** Reinforcement learning and adversarial attacks. Adversarial attacks are not included during training in this work. They are only used to evaluate the robustness of trained policy networks $\mathcal{K}_\theta$ at test time.

In keeping with traditional RL literature [159], we will refer to maximising a reward

$$\mathfrak{R} := \mathop{\mathbb{E}}_{x_0,w}\left[\sum_{t=0}^{T-1} r_t(x_t,u_t) + r_T(x_T)\right]$$

in this chapter rather than minimising a cost (2.2) (these two perspectives often only differ by a negative sign on the cost or reward). We also restrict ourselves to static nonlinear policies $\mathcal{K}_\theta$, and train them using PPO [143] (Section 2.4.2).

### 3.2.2 Adversarial Attacks for Deep RL

Given a learned policy $\mathcal{K}_\theta$, an adversarial attack is an input sequence $v = (v_0, v_1, \ldots)$ with some restricted "attack size" $\epsilon > 0$ that is designed to reduce the expected cumulative reward of the policy as much as possible. While attacks can be designed to change the closed-loop behaviour in many ways, the most common structure is an additive perturbation to the policy input (i.e., perturbations to the state measurements, Figure 3.1) [70, 122]. The attack problem can be formulated as

$$\min_v \mathfrak{R} \quad \text{s.t.} \quad x_{t+1} = f(x_t, \mathcal{K}_\theta(x_t + v_t), w_t), \quad |v_t|_p \leq \epsilon, \ \forall t, \tag{3.3}$$

where $|\cdot|_p$ can be any $p$-norm with $p \in [1, \infty)$. If the reward function and dynamic model are both differentiable and known to the attacker, (3.3) can be solved directly by gradient descent. If part of the dynamical system is unknown or not differentiable,

solving (3.3) is difficult or impossible and it is common to instead solve the simplified problem

$$\max_{v_t} \quad |\mathcal{K}_\theta(x_t + v_t) - \mathcal{K}_\theta(x_t)|_p \quad \text{s.t.} \quad |v_t|_p \leq \epsilon. \tag{3.4}$$

That is, find an admissible attack $v_t$ which leads to a large perturbation in the policy output at each time step, regardless of its effect on the final reward [70]. Problem (3.4) can be solved by attack methods like Projected Gradient Descent (PGD) [101].

### 3.2.3 Lipschitz-Bounded Deep Networks

The adversarial attack problem in (3.4) is exactly the calculation of the (local) Lipschitz constant of the policy network $\mathcal{K}_\theta$. We can therefore control the effect of adversarial attacks everywhere in state space by bounding the global Lipschitz constant $\text{Lip}(\mathcal{K}_\theta) \leq \gamma$. Policy networks with smaller $\gamma$ are then smoother and are likely to be more robust to attacks than those with a large $\gamma$.

As discussed in Section 2.1, deep RL policy networks are often parametrised by $L$-layer MLPs or CNNs of the form

$$\mathcal{K} = \phi_L \circ \sigma \circ \phi_{L-1} \circ \cdots \circ \sigma \circ \phi_1 \tag{3.5}$$

where $\sigma$ is a fixed monotone and 1-Lipschitz scalar activation (Figure 2.5) and $\phi_k$ is a linear layer $\phi_k(x) = W_k x + b_k$ with weights $W_k$ and biases $b_k$, respectively. The Lipschitz constraint for (3.5) is then

$$\text{Lip}(\mathcal{K}_\theta) = \sup_{J_2,\ldots,J_L} \|W_L J_L W_{L-1} \cdots J_2 W_1\|_2 \leq \gamma \tag{3.6}$$

where $J_k$ is a diagonal matrix with $0 \leq J_{k,ii} \leq 1$ and $\|\cdot\|_2$ denotes the matrix 2-norm (the spectral norm). Since it is NP-hard to compute the exact Lipschitz constant of a DNN [177], a practical approach is to find an upper bound $\overline{\gamma}$ for $\text{Lip}(\mathcal{K}_\theta)$ and then impose the constraint $\overline{\gamma} = \gamma$. There are many bound estimation methods resulting in different constructions of Lipschitz networks. A metric to measure the expressive

power of Lipschitz networks is *tightness*, which is defined in [181] as $\underline{\gamma}/\gamma$ with $\underline{\gamma}$ as the empirical lower Lipschitz bound

$$\underline{\gamma} := \max_{x \in \mathbb{X}, |v| \leq \epsilon} \frac{|\mathcal{K}_\theta(x+v) - \mathcal{K}_\theta(x)|}{|v|} \tag{3.7}$$

where $v \in \mathbb{R}^{n_x}$ and $\mathbb{X} \subset \mathbb{R}^{n_x}$ is some compact region, and $\text{Lip}(\mathcal{K}_\theta) \in [\underline{\gamma}, \gamma]$ by definition.

A simple Lipschitz bound estimation is the layer-wise spectral norm bound $\gamma = \prod_{k=1}^{L} \|W_k\|_2$. This approach uses the fact that common activation functions $\sigma$ are 1-Lipschitz. Thus, we can construct Lipschitz policy networks via scaling factors and 1-Lipschitz linear layers $\phi(x) = Wx + b$, as depicted in Figure 2.6. We give three representative examples of Lipschitz policy networks constructed via this method below, and introduce them in order of increasing tightness $\underline{\gamma}/\gamma$. Note that each of the following layers are directly parametrised to satisfy the 1-Lipschitz constraint, and therefore inherit all the computational benefits of directly-parametrised robust neural networks discussed in Section 2.3.1.

- **(SN)** The Spectral Normalisation (SN) layer [112] is a linear layer with weight matrix

  $$W = (1/\rho)A$$

  directly parametrised by a learnable matrix $A$ whose maximum singular value is $\rho = \|A\|_2$ such that $\|W\|_2 = 1$ by construction. Lipschitz networks composed of SN layers often have quite loose Lipschitz bounds, so $\underline{\gamma}/\gamma$ is very small.

- **(AOL)** The Almost Orthogonal Lipschitz (AOL) layer [126] is a linear layer with weight matrix

  $$W = AD$$

  directly parametrised by a (dense) learnable matrix $A$ and a diagonal matrix $D$ with $D_{ii} = \sqrt{\sum_j |A^\top A|_{ij}}$. The experimental results in [126] show that the learned weight $W$ tends to have singular values close to 1, which helps to improve the model tightness compared to SN.

- **(Cayley)** To obtain an even tighter Lipschitz bound, an orthogonal layer was proposed in [170] which leverages the Cayley transform (2.19). Specifically, given a free learnable matrix $P \in \mathbb{R}^{n\times n}$, one first obtains a skew symmetric matrix $A = P - P^\top$ and then constructs the weight matrix by

$$W = (I - A)(I + A)^{-1}.$$

  It is easy to verify that $W^\top W = I$ and hence all singular values of $W$ are automatically one. Note that the Cayley layer has greater computational overhead than the SN or AOL layers, since the Cayley transform involves taking a matrix inverse. This is the trade-off for a tighter upper bound on the Lipschitz constant.

Each of these layers parametrises 1-Lipschitz feedforward networks by constraining their spectral norm to be less than or equal to one. An alternative strategy is to directly use the tighter and more expressive bound estimation proposed in [44] which exploits both the monotonicity and Lipschitz properties of the activation $\sigma$ by leveraging the IQC framework [108]. This is exactly the approach taken by the Sandwich layer [181], which we introduced in detail in Section 2.3.2.

- **(Sandwich)** The Sandwich layer [181] is a 1-Lipschitz nonlinear layer of the form (2.16b). It is an expressive layer architecture in that it contains all 1-Lipschitz linear layers as a special case, and it allows the spectral norm bound of each layer (and their product) to be greater than one [181, Fig. 4].

Experimental results in [181] on image classification tasks show that the Sandwich layer can achieve better performance than 1-Lipschitz linear layers such as AOL and Cayley. It is natural to ask whether similar results hold true for deep RL.

## 3.3 Experimental Setup

We study two classic RL problems – pendulum swing-up and Atari Pong – to investigate the research questions outlined at the beginning of this chapter. Code for

all experiments is available on GitHub[1,2]. Additional training details are provided in Section 3.6.

## Pendulum Swing-up

The pendulum swing-up task is depicted in Figure 3.2a. The aim is to swing a physical pendulum to its upright equilibrium based on a quadratic reward (negative stage cost)

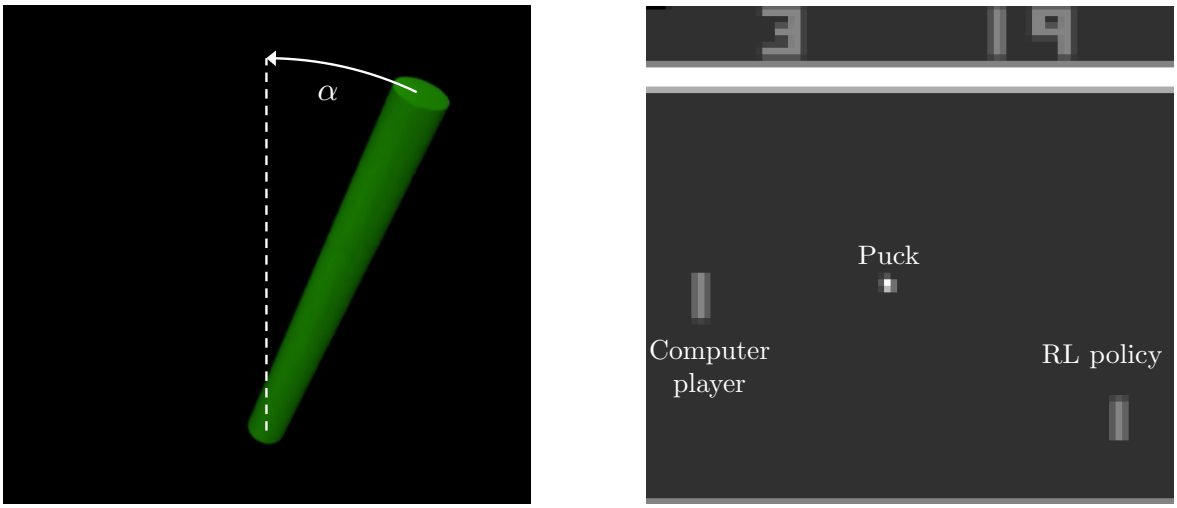
$$r_t = -(\alpha_t^2 + 0.1\dot{\alpha}_t^2 + 0.001u_t^2), \tag{3.8}$$

where $\alpha_t$ is the pendulum angle (wrapped to $[-\pi, \pi]$), $\dot{\alpha}_t$ is its angular velocity, $u_t$ is the pendulum torque, and the reward weightings are from [168]. The optimal policy is well-known to have sharp decision boundaries which make it susceptible to chattering and instability under small measurement perturbations, delays, or uncertainty. We trained unconstrained MLP policies without any bounds on their Lipschitz constant, and Lipschitz-bounded policies using the Sandwich layer. We investigated the robustness of each policy to two sources of perturbations: (a) sample delays; and (b) adversarial attacks with constrained $\ell_2$ norm. Adversarial attacks were computed by solving (3.3) over a sequence of four windows of 50 time-samples each with gradient descent.

## Atari Pong

Atari Pong (Figure 3.2b) is a video game in which two players each control a paddle that can move up and down, and try to deflect a puck into their opponent's goal. The reward is the net game score, with a maximum score of 21 goals to nil. An automated "computer" player controls the left paddle and the RL policy controls the right paddle. It is a commonly-studied benchmark in deep RL [113, 139]. The game can be written as an RL problem where the states are grayscale gameplay images and the control actions are discrete paddle movements. We trained classic, unconstrained CNN models

[1] `https://github.com/nic-barbara/Lipschitz-RL-MJX`
[2] `https://github.com/nic-barbara/Lipschitz-RL-Atari`




**(a)** Pendulum swing-up. **(b)** Atari Pong.

**Figure 3.2:** Feedback control problems to be solved via RL. In pendulum swing-up, the goal is to swing the pendulum upright and maximise a cumulative reward based on (3.8). In Atari Pong, the goal is to be the first player to 21 points.

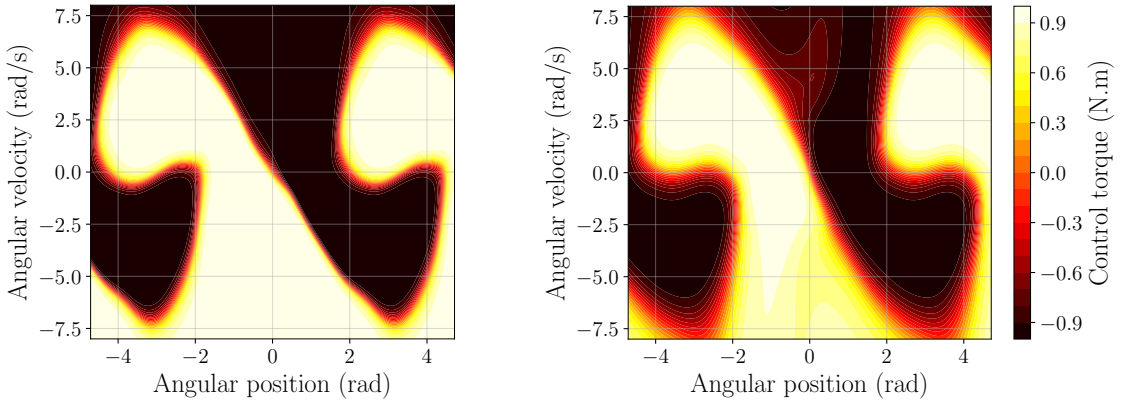
and four different Lipschitz-bounded policy architectures (SN, AOL, Cayley, Sandwich) with various Lipschitz bounds. The architecture for each network is provided in Table 3.2. We compared the robustness of each policy network to: (a) uniform random noise; (b) PGD attacks (3.4) with constrained $\ell_2$ norm; and (c) PGD attacks (3.4) with constrained $\ell_\infty$ norm.

## 3.4 Results and Discussion

We first study the advantages of Lipschitz-bounded policy networks in terms of robustness to perturbations and adversarial attacks in Section 3.4.1, using pendulum swing-up as an illustrative example. We then extend this study to vision-based feedback control in Atari Pong in Section 3.4.2 and compare the benefits of different Lipschitz-bounded policy architectures. The Lipschitz lower bounds $\underline{\gamma}$ in Figures 3.6, 3.7, and Table 3.1 were computed for each policy by performing gradient ascent on (3.7).

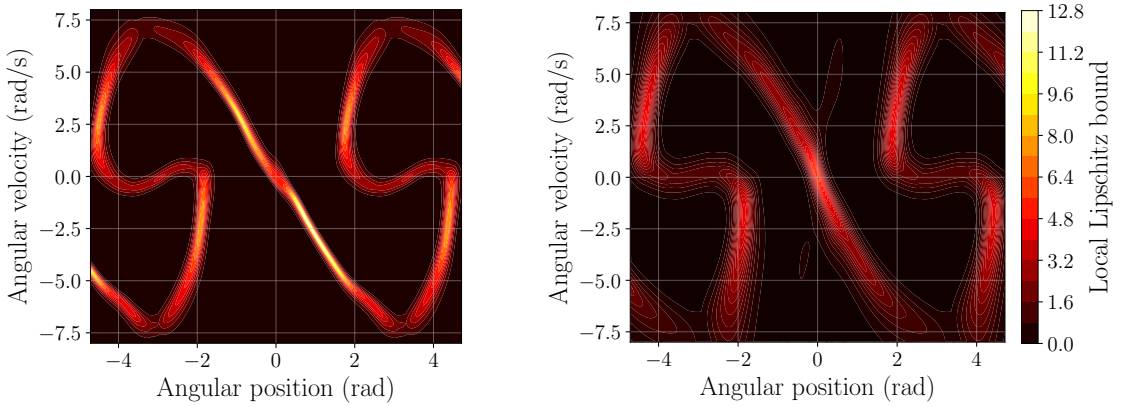
### 3.4.1 Illustrative Example – Pendulum Swing-up

Let us first consider the effect of small Lipschitz bounds on unperturbed policy networks. Figure 3.3 compares the policy landscape and (empirically-estimated) local Lipschitz constant in phase space for an unconstrained MLP policy and a Lipschitz-bounded policy composed of Sandwich layers with $\gamma = 4$. The unconstrained policy has sharp decision boundaries, either side of which it flips the sign of the control torque (limited to $\pm 1$ N.m). While these sharp changes are optimal in the unperturbed case, it is clear that any small uncertainty in the pendulum's position or velocity will cause the




**(a)** Unconstrained policy landscape.

**(b)** Lipschitz-bounded policy landscape.




**(c)** Local Lipschitz (unconstrained).

**(d)** Local Lipschitz (Lipschitz-bounded).

**Figure 3.3:** Contours of control actions (a,b) and local Lipschitz bounds (c,d) for an unconstrained (MLP) and a Lipschitz-bounded (Sandwich, $\gamma = 4$) policy show how Lipschitz bounds regulate a policy's smoothness.

network to apply a drastically different control action, potentially driving the system to instability. In contrast, imposing a bound on the Lipschitz constant with Sandwich layers visibly smooths the decision boundaries with negligible penalty to the final test reward ($-153$ for unconstrained and $-157$ for Lipschitz-bounded, see Figure 3.4).

Figure 3.5 illustrates the effect of this smoothing on each policy's robustness to sample delays and $\ell_2$-optimal adversarial attacks. The simulated trajectories in Figures 3.5a and 3.5b indicate that both policy networks perform similarly in nominal operation. However, when introducing a small sample delay (2 time samples or 0.1 s, Figures 3.5c and 3.5d) or a small adversarial attack ($\epsilon = 0.11$, Figures 3.5e and 3.5f), the unconstrained policy is unable to hold the pendulum stable and upright, whereas the Lipschitz-bounded policy is successful and only exhibits minor oscillations about the equilibrium under adversarial attacks.

We find that this improvement in robustness is highly correlated with the policy's Lipschitz bound. Comparing unconstrained policies to Lipschitz-bounded policies with

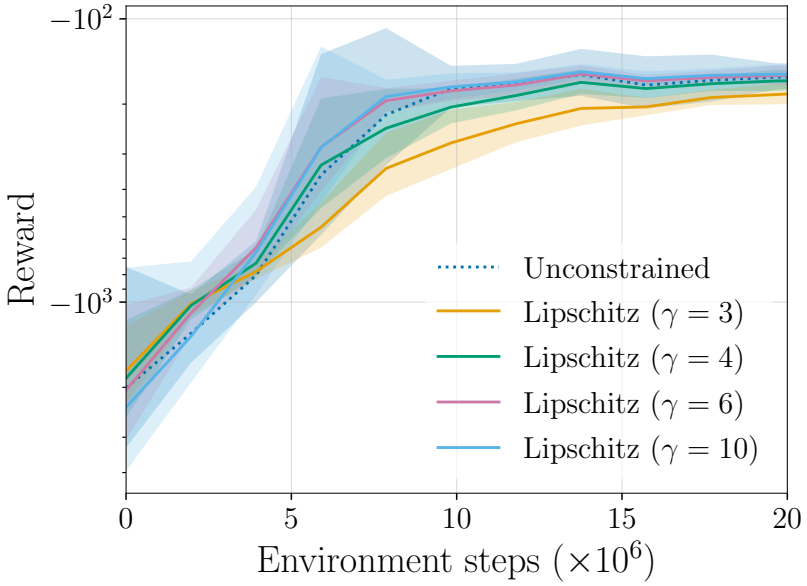
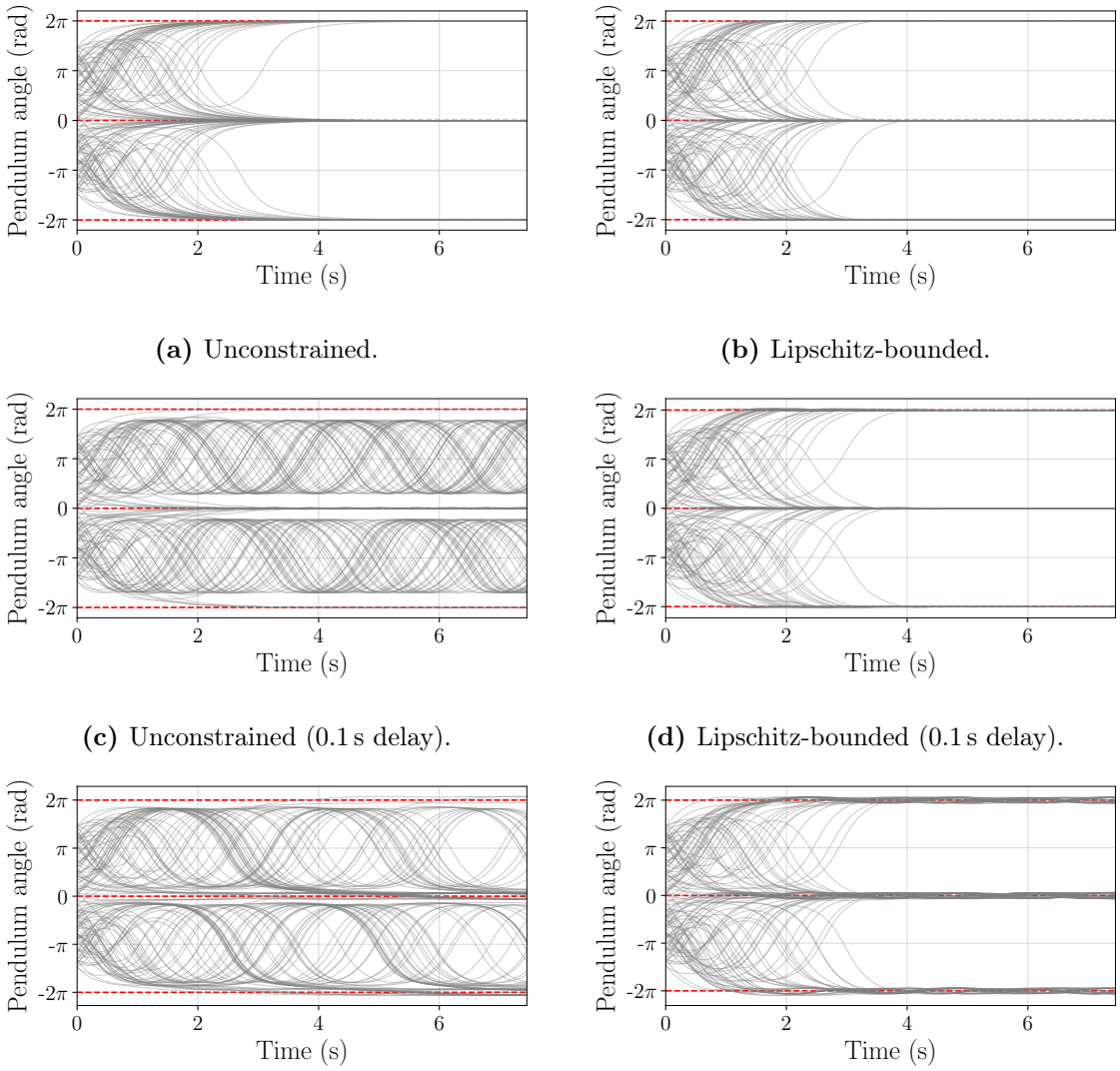



**Figure 3.4:** Test cost vs. training epochs for the pendulum swing-up problem with unconstrained (MLP) and Lipschitz-bounded (Sandwich) policies. Bands show one standard deviation over 10 random model initialisations.




**(a)** Unconstrained.

**(b)** Lipschitz-bounded.

**(c)** Unconstrained (0.1 s delay).

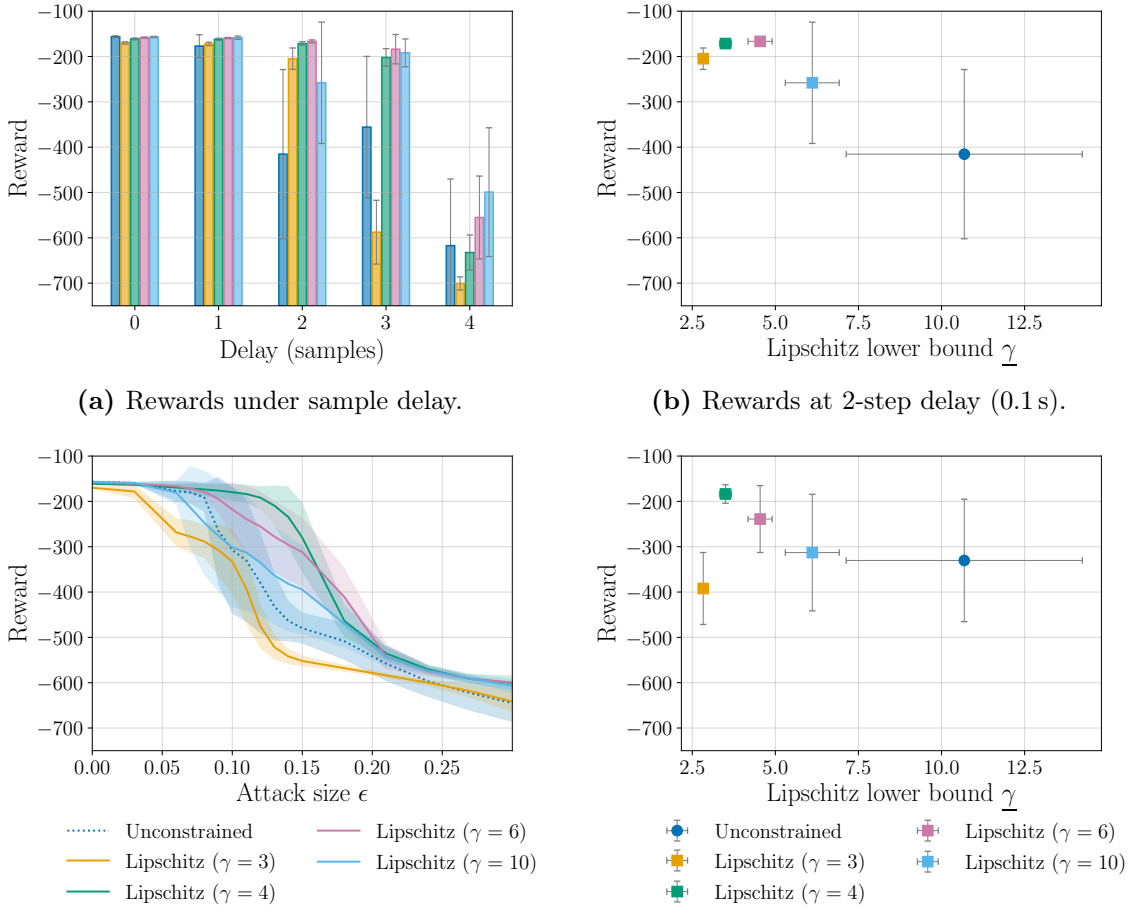
**(d)** Lipschitz-bounded (0.1 s delay).

**(e)** Unconstrained ($\ell_2$ attack, $\epsilon = 0.11$).

**(f)** Lipschitz-bounded ($\ell_2$ attack, $\epsilon = 0.11$).

**Figure 3.5:** Pendulum trajectories generated by unconstrained (MLP) and Lipschitz-bounded (Sandwich, $\gamma = 4$) policies in nominal operation (a,b), with sample delays (c,d), and with $\ell_2$ adversarial attacks (e,f). Red lines indicate the target.

various $\gamma$ in Figure 3.6, there is a smooth transition from high to low robustness to sample delays and $\ell_2$-optimal adversarial attacks as the policy's Lipschitz bound increases. Interestingly, it appears that there is a "best choice" for $\gamma$, and that restricting it to very small values ($\gamma = 3$) harms the closed-loop performance. This is to be expected, since the optimal policy for pendulum swing-up is known to be non-smooth, hence with excessive regularisation it is likely that the network's parameter space does




**(a)** Rewards under sample delay.

**(b)** Rewards at 2-step delay (0.1 s).

**(c)** Reward under $\ell_2$-optimal attacks.

**(d)** Rewards at $\epsilon = 0.11$.

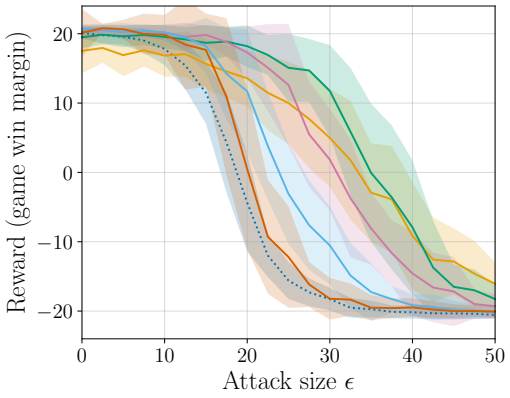
**Figure 3.6:** Robust performance of unconstrained (MLP) and Lipschitz-bounded (Sandwich) policies on pendulum swing-up under sample delays and $\ell_2$-optimal adversarial attacks. Panels (b,d) show cross-sections of (a,c) as a function of each model's empirically-estimated Lipschitz lower bound. Bands and error bars show one standard deviation over 10 random model initialisations.

not contain any high-performing policies. Aside from the policies with $\gamma = 3$, all models were successfully trained to approximately the same final reward (Figure 3.4). It is therefore clear from Figures 3.3 to 3.6 that, at least in the context of pendulum swing-up, Lipschitz-bounded policy networks significantly improve robustness to disturbances and adversarial attacks over standard, unconstrained networks without sacrificing clean performance.

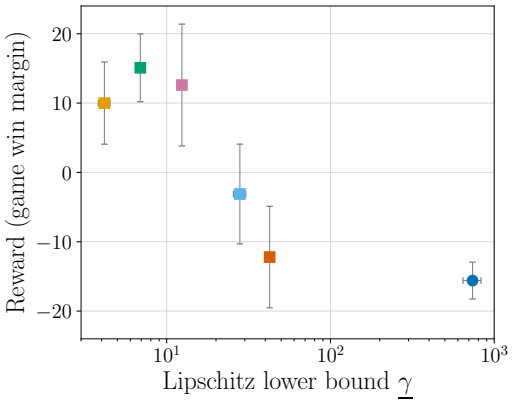
### 3.4.2 Comparing Architectures – Atari Pong

In the game of Pong, we expect small amounts of image noise to make very little difference to the state of the game and the optimal action. We would therefore hope that smooth, Lipschitz-bounded policies can improve robustness to perturbations like random noise and adversarial attacks. We see this immediately in Figure 3.7, where we compare the robustness of unconstrained CNN policies and Lipschitz-bounded Sandwich policies to uniform random noise, $\ell_2$ PGD attacks, and $\ell_\infty$ PGD attacks. The same qualitative results observed for the pendulum in Figure 3.6 can be seen in Figure 3.7: there is a smooth transition from high to low robustness as $\gamma$ decreases; robustness is improved not just for $\ell_2$-constrained attacks (which we expect for policies with a small $\ell_2$ Lipschitz bound), but also for random noise and $\ell_\infty$-constrained attacks; and if $\gamma$ is too small (e.g., $\gamma = 5$ here), the policy's nominal performance and robustness to perturbations is degraded. In this case, it is possible that the $\gamma = 5$ models have not finished training and could perform better if trained over more epochs (see Figure 3.9d).

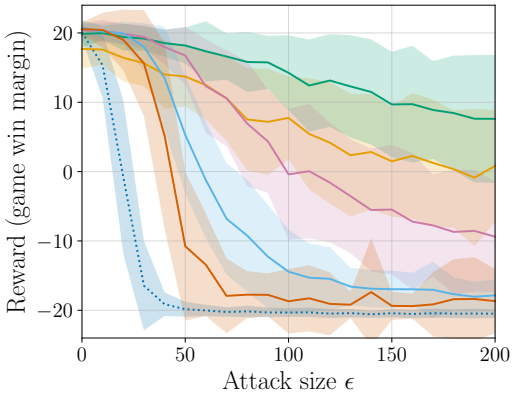
It is interesting to look deeper into the effect of adversarial attacks on these models. Figure 3.8 shows just how much of an improvement Lipschitz-bounded policy networks provide in Pong over a standard, unconstrained CNN. The CNN loses the game when subject to very small amounts of random noise and almost imperceptible adversarial attacks. In contrast, the Lipschitz-bounded policy is only beaten with a level of random noise that could even make the game difficult for a human. Moreover, highly-structured $\ell_2$-constrained attacks are required to beat the Lipschitz-bounded policy. Looking closely at Figure 3.8, successful $\ell_2$ PGD attacks try to trick the policy into thinking the opponent's paddle is shifted from where it actually is while also trying to hide the exact location of the puck. The $\ell_2$ attacks also seem to focus on the white boundary walls of the game. It is less clear why the policies should be sensitive to these features, but we hypothesise that the straight lines of the walls may appear similar to a paddle in feature space after passing through convolutional layers. There is no clear structure to the $\ell_\infty$ PGD attacks and they remain rather small, since the Lipschitz-bounded



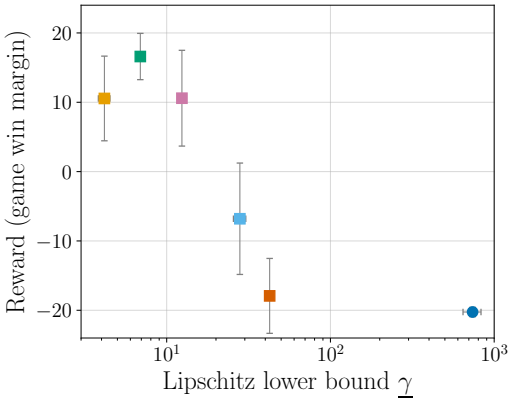

**(a)** Uniform random noise.



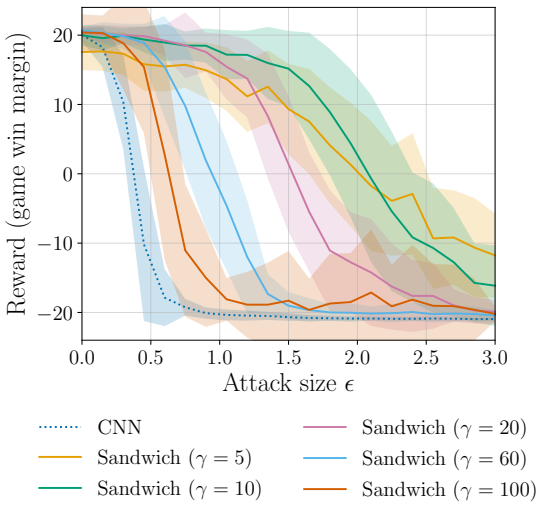

**(b)** Cross-section at $\epsilon = 25$.



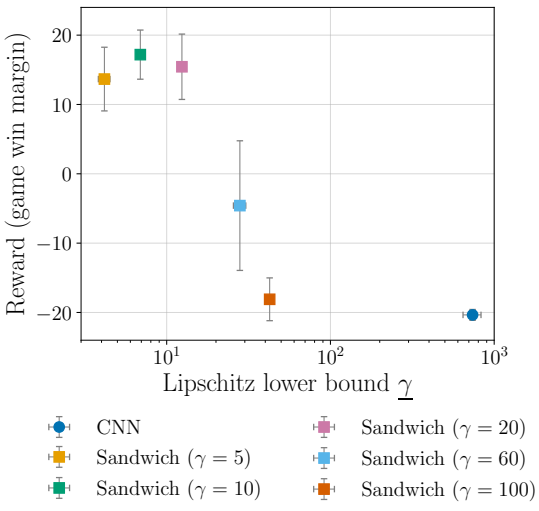

**(c)** $\ell_2$ PGD attacks.




**(d)** Cross-section at $\epsilon = 70$.




**(e)** $\ell_\infty$ PGD attacks.




**(f)** Cross-section at $\epsilon = 1.05$.

**Figure 3.7:** Robust performance of unconstrained (CNN) and Lipschitz-bounded (Sandwich) policies for Atari Pong. Bands and error bars show one standard deviation over 4 random model initialisations.

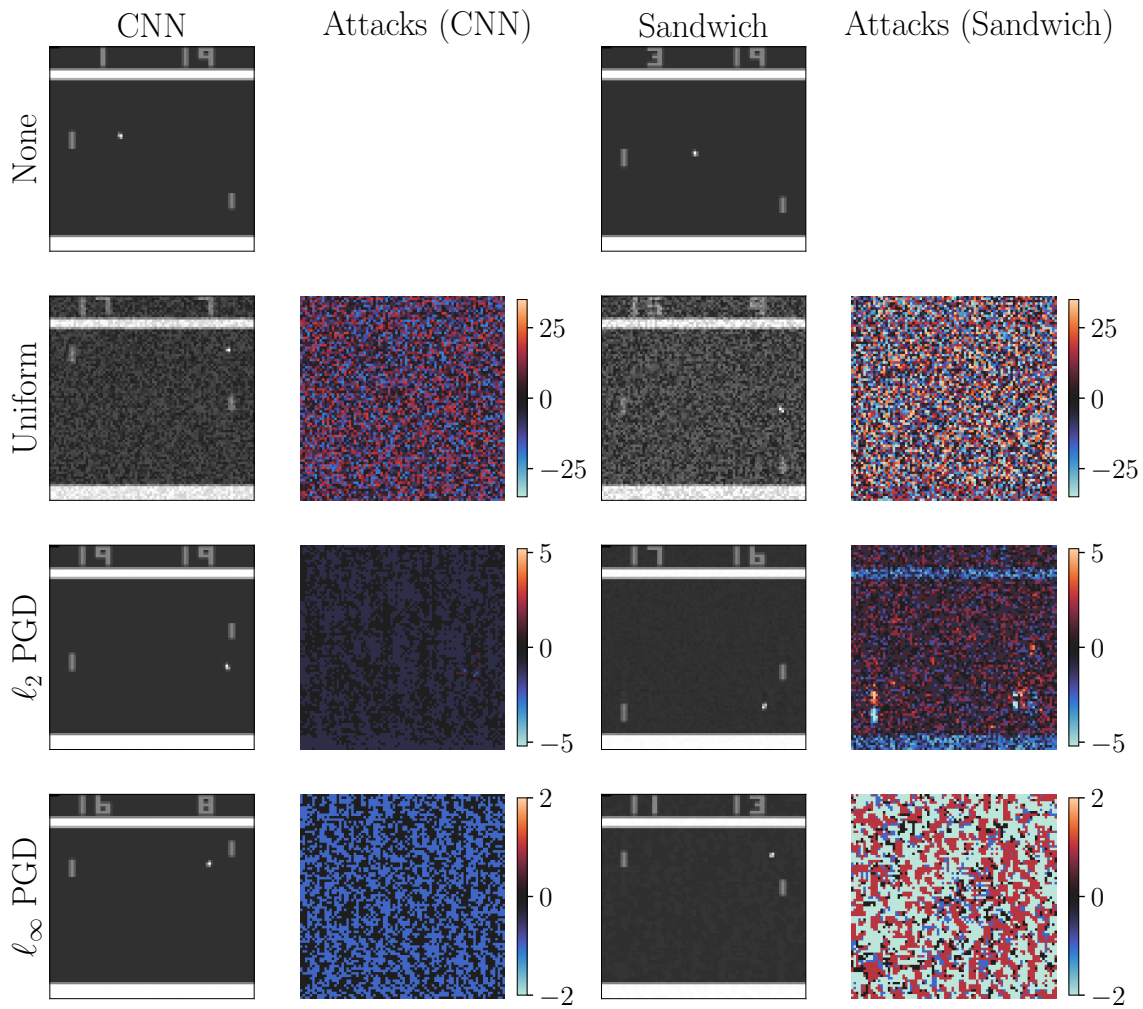
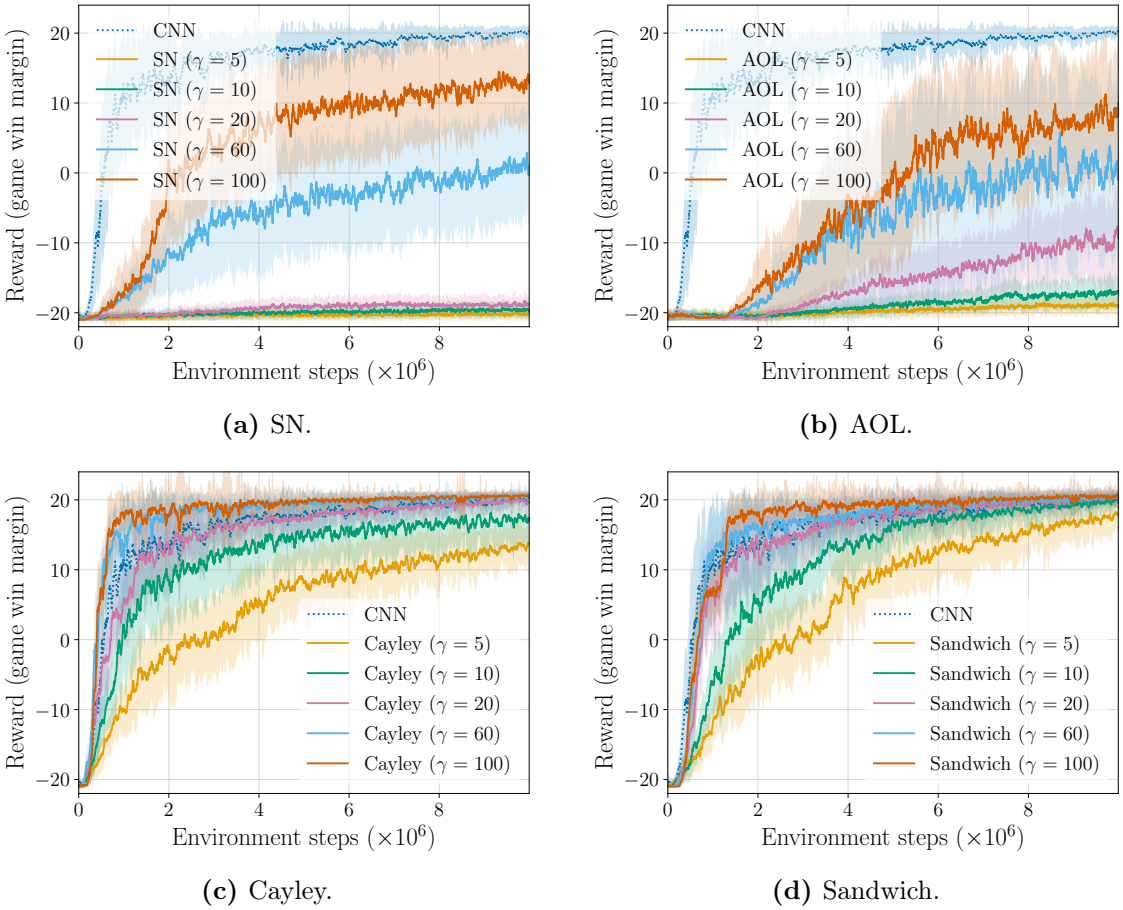



**Figure 3.8:** Adversarial examples showing the smallest attacks required to make an unconstrained (CNN) and a Lipschitz-bounded (Sandwich, $\gamma = 10$) policy lose the game. In each image, the "computer" controls the paddle on the left, while the policy controls the paddle on the right. All perturbed frames show scenarios where the policy is about to concede a goal (puck moving to the right).

policies have a constrained $\ell_2$ Lipschitz bound, and the $\ell_2$ norm is only a loose upper bound of the $\ell_\infty$ norm in high-dimensional spaces. The additional robustness to $\ell_\infty$ PGD attacks over CNN policies is a nice bonus.

So far, we have only investigated Lipschitz-bounded policy networks constructed from Sandwich layers to illustrate that smoother policies can improve robustness in deep RL. It turns out that in addition to choosing its upper bound $\gamma$, the layer architecture we use to bound the Lipschitz constant $\text{Lip}(\mathcal{K}_\theta)$ of a policy $\mathcal{K}_\theta$ is extremely important. Fig-




**(a)** SN. **(b)** AOL. **(c)** Cayley. **(d)** Sandwich.

**Figure 3.9:** Test reward during training on Atari Pong for all layer architectures. Bands show one standard deviation over 4 random model initialisations.

ure 3.9 compares the training curves of all four Lipschitz-bounded policy architectures from Section 3.2.3 with unconstrained CNN policies, while Table 3.1 summarises their nominal and robust performance. It is immediately clear from Figures 3.9a and 3.9b that the two layer architectures known to have conservative bounds, SN and AOL, perform poorly when $\gamma$ is small – e.g., when $\gamma \leq 20$, neither architecture ever produces a winning policy. For larger $\gamma$, the SN and AOL policies learn to win the game, but their training dynamics are extremely slow in comparison to the CNN (this is a known problem for AOL, see [126, Sec.7]). Moreover, Table 3.1 shows that the estimated lower bound $\underline{\gamma}$ on $\text{Lip}(\mathcal{K}_\theta)$ for these policies is very small when $\gamma$ is small. This suggests that the conservative parametrisation of SN and AOL layers restricts their parameter spaces to small sets of very smooth models which do not include high-performing policies. In

| **Policy** | **Lipschitz** | | **Reward** | **Smallest winning attack size $\epsilon$ (↑)** | | |
|---|---|---|---|---|---|---|
| | $\gamma$ | $\underline{\gamma}$ | Unperturbed | Uniform | $\ell_2$ PGD | $\ell_\infty$ PGD |
| CNN | - | 738 | 20.1 | 18.7 | 19.5 | 0.38 |
| SN | 5 | 1.14 | -20.2 | - | - | - |
| | 10 | 2.78 | -19.4 | - | - | - |
| | 20 | 6.05 | -18.7 | - | - | - |
| | 60 | 24.7 | 1.21 | 15.0 | 9.80 | 0.21 |
| | 100 | 42.4 | 14.8 | 21.5 | 21.3 | 0.49 |
| AOL | 5 | 1.40 | -18.9 | - | - | - |
| | 10 | 2.95 | -16.9 | - | - | - |
| | 20 | 7.62 | -9.28 | - | - | - |
| | 60 | 15.3 | 1.31 | 12.4 | 19.2 | 0.52 |
| | 100 | 17.4 | 8.85 | 23.6* | 42.3 | 1.20* |
| Cayley | 5 | 4.56 | 13.6 | 23.7 | 106 | 1.52 |
| | 10 | 8.99 | 17.8 | 28.7 | 154 | 1.59 |
| | 20 | 16.6 | 20.0 | 23.1 | 129* | 1.36 |
| | 60 | 34.8 | **20.6*** | 20.9 | 67.4* | 0.98* |
| | 100 | 47.0 | 20.4* | 17.1 | 54.0* | 0.76 |
| Sandwich | 5 | 4.16 | 17.5* | 33.4* | 183* | 2.01* |
| | 10 | 6.90 | 19.5* | **35.0*** | **> 200*** | **2.08*** |
| | 20 | 12.4 | 20.2* | 30.8* | 99.2 | 1.53* |
| | 60 | 28.0 | **20.6*** | 23.9* | 58.1 | 0.94 |
| | 100 | 42.5 | 20.2 | 20.1 | 43.3 | 0.63 |

**Table 3.1:** Averaged results (4 random model initialisations and 20 random game and attack seeds) on performance and robustness of policy networks trained on Atari Pong. $\gamma$ is the certified Lipschitz upper-bound for a network and $\underline{\gamma}$ is its empirically-estimated lower-bound. The unperturbed reward is the final mean test reward achieved during training. Results for each attack strategy are the smallest average attack size $\epsilon$ required to beat the policy (i.e., reward $< 0$). Attack results are not provided for policies that did not learn a positive reward. **Bold** values indicate the overall best performing models in each column. Values with an * indicate the best performing models for each value of $\gamma$.

contrast, Figures 3.9c and 3.9d show that the two layer architectures with much tighter bounds on $\text{Lip}(\mathcal{K}_\theta)$, Cayley and Sandwich, perform quite well even for small choices of $\gamma$. The choice of $\gamma$ still seems to have a strong impact on the training dynamics for these layers – as $\gamma$ increases, so too does the speed at which the policies converge on a winning strategy. This raises interesting questions about the coupling between a policy's Lipschitz constant and the exploration of its parameter space, which we leave

for future work.

Looking closer at Table 3.1 reveals that simply having a tight Lipschitz bound (i.e., close lower and upper bounds $\underline{\gamma}$ and $\gamma$) is not the only factor contributing to a policy's performance and robustness. Instead, there appears to be an advantage to using expressive policy networks, particularly when high robustness (small $\gamma$) is required. Table 3.1 indicates that both the Sandwich and Cayley policies exhibit strong performance and robustness for $\gamma \geq 20$. Either layer architecture is therefore a suitable choice for reasonable improvement over existing methods in robust RL like SN. For smaller $\gamma < 20$, however, Sandwich policies are far superior, and with $\gamma = 10$ they are the most robust of any policy across all three input perturbations while still achieving a strong unperturbed reward of 19.6. This is despite the fact that Cayley policies often have a tighter Lipschitz bound than the Sandwich policies (take $\gamma = 10$ as an example). We suggest that this is due to the less conservative parametrisation of the Sandwich layers. Each of the SN, AOL, and Cayley policies are composed of linear layers with a spectral norm of approximately 1 (exactly 1 for Cayley). In contrast, Sandwich layers have no direct restriction on their spectral norm [181, Fig. 4], and instead constrain $\text{Lip}(\mathcal{K}_\theta)$ via a composition of nonlinear layers which are a complete parametrisation of *all* networks satisfying the tightest known bounds on the Lipschitz constant of DNNs [44], of which 1-Lipschitz linear layers are a special case. This expressivity allows the Sandwich models to converge on policy networks that are both performant and robust even when their parameter space is restricted by a small value of $\gamma$, allowing finer control over the performance-robustness trade-off in deep RL.

## 3.5 Conclusions

This chapter has studied the robustness benefits of Lipschitz-bounded policy networks in deep RL. We have found that policy networks with small Lipschitz bounds are significantly more robust to perturbations such as disturbances, random noise, and targeted adversarial attacks. Moreover, we have observed that choosing a policy with

non-conservative Lipschitz bounds and an expressive layer architecture like the Sandwich layer gives the user finer control over the performance-robustness trade-off than existing methods based on spectral normalisation. This raises interesting questions for future study, such as whether Lipschitz-bounded policy networks can augment or alleviate the need for adversarial training, and whether the observed benefits can be transferred to real-world robotic systems.

## 3.6 Additional Training Details

### Pendulum Swing-up

The pendulum was modelled in MuJoCo XLA (MJX), a fully-differentiable implementation of the MuJoCo physics simulator [167] written in JAX [22] that allows users to take gradients through the entire closed-loop system. We trained policy networks using the PPO implementation in [46], which leverages the scalability of JAX to massively multi-thread parallel physics simulations on a GPU. Hyperparameters were tuned by varying each parameter one at a time and choosing the best-performing parameters for an unconstrained MLP. The same hyperparameters were used to train Lipschitz-bounded policies without further tuning. Our chosen hyperparameters can be found on our GitHub repository[1].

We trained 10 policies for each model architecture and choice of $\gamma$, each with a different random seed for model initialisation. For each policy architecture, we averaged our results over the 10 random model initialisations and 1024 pendulum environments starting from random initial states. MLP networks were composed of 4 linear layers of 32 hidden nodes each. Lipschitz-bounded policies were composed of 4 Sandwich layers of 21 hidden nodes each to ensure the two model architectures had a similar number of trainable parameters. We chose tanh activations for all policies in accordance with [5].

### Atari Pong

We trained unconstrained CNN models and Lipschitz-bounded policy architectures (SN, AOL, Cayley, Sandwich) across 4 random model initialisations using the PPO implementation in [72] and the ALE/Pong-v5 environment from [183]. We used the default hyperparameters chosen for CNN policies in [72] for all policy architectures. We could not use the reward gradient to directly optimise (3.3) as the Pong environment is not differentiable, hence we implemented attacks with the PGD method [39]. Robust performance to noise and adversarial attacks in Figure 3.7 and Table 3.1 were averaged over the 4 policies and 20 games of Pong, each with a different random seed.

We maintained as consistent a network architecture as possible between all five layer types (CNN, SN, AOL, Cayley, Sandwich). An overview of each network architecture is provided in Table 3.2. The state measurements in the the ALE/Pong-v5 environment [183] are stacks of 4 sequential $(84 \times 84)$-pixel gameplay images, and the number of possible control actions is restricted to 6 options: do nothing; fire; move right; move left; move right and fire; and move left and fire. We added padding to the input images to maintain a square image after each convolutional layer. Note that we added a second pooling layer to the AOL networks because the AOL layer architecture does not support strided convolutions [126]. The number of inputs in the first linear layer of each network (Dense 1 in Table 3.2) differs because the Cayley and Sandwich layers only support circular convolution kernels [170, 181].

| Layer | CNN | SN | AOL | Cayley | Sandwich |
|---|---|---|---|---|---|
| Input Padding | `Pad(2,2,2,2)` | `Pad(2,2,2,2)` | `Pad(2,2,2,2)` | `Pad(2,2,2,2)` | `Pad(2,2,2,2)` |
| Scaling 1 | – | $\sqrt{\gamma}$ | $\sqrt{\gamma}$ | $\sqrt{\gamma}$ | $\sqrt{\gamma}$ |
| Conv 1 | `Conv(4,32,8,2)` | `SN_Conv(4,32,8,2)` | `AOL_Conv(4,32,8)` | `Ca_Conv(4,32,8,2)` | `Sw_Conv(4,32,8,2)` |
| Pooling 1 | `MaxPool(2)` | `MaxPool(2)` | `MaxPool(4)` | `MaxPool(2)` | `MaxPool(2)` |
| Conv 2 | `Conv(32,64,4,2)` | `SN_Conv(32,64,4,2)` | `AOL_Conv(32,64,4)` | `Ca_Conv(32,64,4,2)` | `Sw_Conv(32,64,4,2)` |
| Pooling 2 | – | – | `MaxPool(2)` | – | – |
| Conv 3 | `Conv(64,64,3,1)` | `SN_Conv(64,64,3,1)` | `AOL_Conv(64,64,3)` | `Ca_Conv(64,64,3,1)` | `Sw_Conv(64,64,3,1)` |
| Resize | `Flatten()` | `Flatten()` | `Flatten()` | `Flatten()` | `Flatten()` |
| Dense 1 | `FC(3136,512)` | `SN_FC(3136,512)` | `AOL_FC(6400,512)` | `Ca_FC(7040,512)` | `Sw_FC(7040,512)` |
| Scaling 2 | – | $\sqrt{\gamma}$ | $\sqrt{\gamma}$ | $\sqrt{\gamma}$ | $\sqrt{\gamma}$ |
| Output | `Lin(512,6)` | `SN_Lin(512,6)` | `AOL_Lin(512,6)` | `Ca_Lin(512,6)` | `Sw_Lin(512,6)` |

**Table 3.2:** Policy network architectures used for the Atari Pong experiments. We use similar notation to the standard PyTorch syntax. For example, `Conv(4,32,8,2)` indicates a CNN layer with 4 input channels, 32 output channels, a kernel size of 8, and a stride of 2. All networks used ReLU activations following the three convolutional layers and the Dense 1 layer. We use the following key: `Conv` indicates a convolutional layer (with ReLU activation on the output); `FC` indicates a fully-connected or dense layer (with ReLU activation on the output); and `Lin` indicates a fully-connected linear layer (no activation function). The prefix for each layer specifies which layer parametrisation is used. For example, `Sw_Conv` indicates a convolutional Sandwich layer.

# Chapter 4

# React to Surprises with Youla-REN – Part I

In Chapter 3, we studied parametrisations of neural policies with guarantees on their input-output robustness. We now close the loop and consider the core problem in this thesis of constructing stable-by-design policy parametrisations – i.e., policy parametrisations that automatically guarantee *closed-loop* stability and robustness. Learning controllers from a class of stable-by-design policies decouples closed-loop stability from the optimisation algorithms, reward functions, and data used to train the policy, enabling the learning of high-performance controllers without risking closed-loop instability.

This chapter, together with Chapter 5, contains the main contribution of this thesis. We construct a stable-by-design policy parametrisation based on the classical Youla-Kučera parametrisation from linear systems theory (*a.k.a* the Youla parametrisation) [193, 92]. We study nonlinear versions of the Youla parametrisation and present novel results extending it to strong, incremental notions of stability (contraction) and robustness (Lipschitzness) which are suitable for machine learning applications. We then provide a constructive realisation of the parametrisation called the Youla-REN, which

fuses the Youla framework with RENs [136]. Combined with the built-in properties of RENs, all Youla-REN policies are naturally closed-loop stabilising and robust to disturbances for a broad class of nonlinear systems, making them suitable for plug-and-play learning with data-driven control frameworks such as deep RL.

Our study of the Youla parametrisation and the Youla-REN is split across Chapters 4 and 5 for readability. This chapter introduces the topic, formulates the problem of constructing robustly-stabilising policy parametrisations, and introduces our nonlinear version of the Youla parametrisation. We conclude by presenting theoretical results for simplified settings in which at most two of the following three features are present: (a) nonlinear systems; (b) partially-observed systems; (c) incremental closed-loop stability requirements in the sense of contraction and Lipschitzness. We defer our main theoretical results on systems with all three of (a), (b), and (c), and our numerical experiments with the Youla-REN, to Chapter 5.

## Publications

The content in this chapter previously appeared as part of the following publications. Note that [15], which makes up the majority of Chapters 4 and 5 and builds on [12, 178], is currently under review.

[15] Nicholas H. Barbara, Ruigang Wang, Alexandre Megretski, and Ian R. Manchester, "React to Surprises: Stable-by-Design Neural Feedback Control and the Youla-REN," *Submitted to the IEEE Transactions on Automatic Control,* 2025.

[12] Nicholas H. Barbara, Ruigang Wang, and Ian R. Manchester, "Learning Over Contracting and Lipschitz Closed-Loops for Partially-Observed Nonlinear Systems," *Proceedings of the IEEE Conference on Decision and Control,* 2023.

[178] Ruigang Wang, Nicholas H. Barbara, Max Revay, and Ian R. Manchester, "Learning Over All Stabilizing Nonlinear Controllers for a Partially-Observed Linear System," *IEEE Control Systems Letters,* 2022.

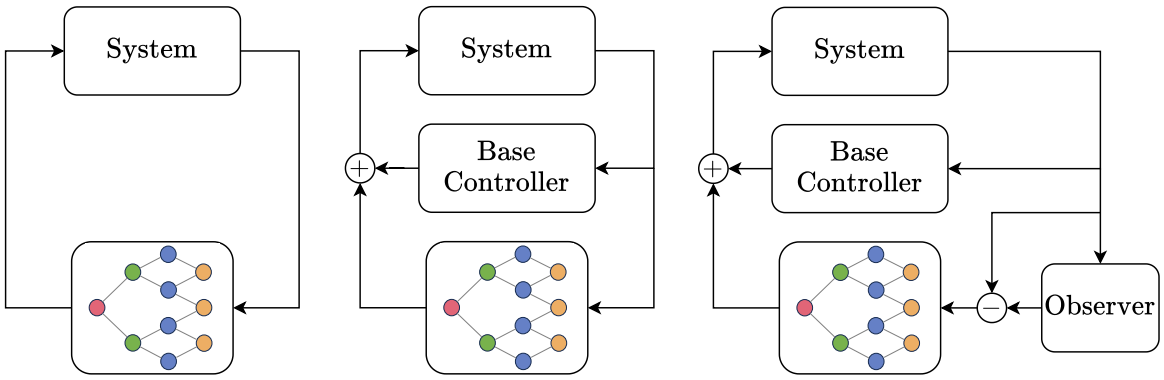
## 4.1 Introduction

Sections 1.1 and 2.1 introduced deep RL as a powerful technology for general-purpose nonlinear control design via simulation, where control policies are parametrised by black-box deep neural networks. However, we have seen that black-box approaches such as deep RL are fundamentally limited by a lack of guarantees on the stability, response to disturbances, and sensitivity to model error of the closed-loop system. While the risk of instability can be reduced with carefully-curated training pipelines [122, 166], or by replacing a black-box policy network with a robust neural network (as in Chapter 3), closed-loop guarantees are not explicitly part of the standard RL problem formulation. This can lead to unexpected failure modes [70, 144]. We seek to remedy this limitation while retaining the computational benefits of simple gradient-based RL tools – i.e., without additional computationally-expensive constraints or stability analysis procedures during training.

### 4.1.1 Feedback Policy Parametrisations

Figure 4.1a shows the typical parametrisation of an RL policy as a neural network in feedback with a (possibly nonlinear, unstable) system. With black-box policies, this architecture offers no closed-loop stability or robustness guarantees, and does not incorporate prior knowledge of the system.

One way to take advantage of prior knowledge is to design a "base controller" – for example, using classical model-based techniques [196, 130, 64] – and augment it with a neural network as in Figure 4.1b. This is called residual RL [151, 79, 100]. The base controller is used to bootstrap performance and allows us to naturally fuse model-based control with data-driven RL. However, closed-loop stability and robustness constraints are typically not imposed in residual RL. It would be possible to do so via (for example) the small-gain theorem if a gain bound from the neural network output to the system output is known and gain-bounded neural networks are employed, such as those studied in Chapter 3. However, this will typically be very conservative.




**(a)** RL. **(b)** Residual RL. **(c)** Youla-REN (ours).

**Figure 4.1:** Feedback controller architectures for deep RL.

We can reduce conservatism by adding a third block to the parametrisation – an observer – as shown in Figure 4.1c, which requires further prior knowledge in the form of a system model. Observers can be thought of as "model-pass filters." They let through information that is consistent with their internal model and filter out unexpected information from the measurements (*a.k.a* the innovations or "surprises"). We say that a neural network *reacts to surprises* if we feed it differences between the measurements and observer predictions. This is a version of the Youla parametrisation, and the neural network in Figure 4.1c is called the Youla parameter. Its main attraction is that, for certain classes of systems and observers: (1) it automatically guarantees closed-loop stability without restricting the network gain, provided that the Youla parameter is itself stable; and (2) it is not a restrictive parametrisation since it covers *all* stabilising controllers for the system. For systems and observers where this is not the case, we can still apply the small-gain theorem to ensure stability. In the Youla parametrisation, the gain from the network output to the surprises is typically much smaller than to the measurements themselves, significantly reducing conservatism.

### 4.1.2 The Youla Parametrisation for Nonlinear Systems

The Youla parametrisation [193, 92] was first established to parametrise all stabilising linear controllers for a given linear system, and has long played a central role in linear robust and adaptive control theory [196, 3], although to our knowledge the use of the

Youla parametrisation in RL was first proposed in [137]. A common construction is to augment an observer-based stabilising base controller with a stable linear system $\mathcal{Q}$ called the Youla parameter, which reacts to surprises much like in Figure 4.1c [41]. The Youla parameter $\mathcal{Q}$ is then optimised to boost the original closed-loop system performance while always preserving stability with hard guarantees [102, 16]. We note that parametrisations essentially equivalent to the Youla parametrisation have been proposed under many names over the years, including: model reference transformations and $\mathcal{Q}$-parametrisations [194]; Internal Model Control (IMC) [53]; disturbance feedback in Model Predictive Control (MPC) [56]; and "nature's y" in online learning approaches to control [152].

Extensions of the Youla parametrisation to nonlinear systems have been studied for many years. Early work addressed the problem by expressing the parametrisation in terms of generic input-output operators [38, 37] and coprime factors [163, 119], which are difficult to realise in practice for general nonlinear systems. [120, 121, 47, 48] proposed using kernel representations to enable intuitive realisations of the nonlinear Youla parametrisation, since they can be translated to observer-based stabilising controllers in state-space. State-space formulations of the problem with observer-based controllers were also studied directly in [99, 74].

Each of the above works focused on parametrising asymptotically or finite-gain stabilising controllers about a particular equilibrium. However, there are many different notions of stability for nonlinear systems. In Section 2.2, we argued that the stronger notion of *incremental* stability is more relevant for RL and learning-based control, since it ensures stable behaviour when generalising to unseen data. In this paper, we therefore seek a formulation of the Youla parametrisation that parametrises incrementally stabilising controllers in the sense of contraction [98] and incremental IQCs [108].

### 4.1.3 Direct Parametrisation of Stable Nonlinear Systems

A bottleneck in applying nonlinear versions of the Youla parametrisation in practice has been the construction of stable nonlinear Youla parameters $\mathcal{Q}$. Prior work in system identification introduced convex parametrisations of contracting and Lipschitz nonlinear dynamic models based on polynomials [165, 164] and neural networks [134]. However, these parametrisations were expressed in the form of linear matrix inequalities via sum-of-squares and IQC constructions, which limited their scalability and required custom optimisation methods [174, 134].

A key recent advance was the direct parametrisation of contracting and Lipschitz models in the form of RENs [136]. We saw in Section 2.3.1 that in a direct parametrisation, the parameters are unconstrained and standard first-order optimisation tools such as stochastic gradient descent can be applied, enabling straightforward integration with existing RL frameworks. Moreover, RENs are expressive in that they universally approximate all contracting and Lipschitz nonlinear systems [178]. Note that alternative parametrisations of RENs are introduced in Chapter 6 to significantly improve their scalability to large neural models.

### 4.1.4 Contributions and Concurrent Work

The main contributions of this chapter, together with Chapter 5, are as follows.

- We introduce a novel version of the Youla parametrisation which guarantees contracting and Lipschitz closed loops in a disturbance-free setting, and prove that the effects of disturbances are constrained to bounded tubes around the disturbance-free trajectories.

- We show that a sufficient condition for contracting and Lipschitz closed loops is that the innovations are decoupled from the Youla augmentation. If the two are coupled, we can still restrict the Youla parameter's input-output properties via IQCs to achieve closed-loop stability.

- We provide partial converse results proving that the parametrisation contains *all* contracting and Lipschitz closed-loops for certain classes of nonlinear systems.
- Combined with RENs, we show via numerical experiments that our resulting Youla-REN policies can achieve strong performance on deep RL tasks while preserving closed-loop stability under disturbances and uncertainty.

Prior work in [180, 136] introduced preliminary versions of the Youla-REN for linear systems. This chapter, together with Chapter 5, studies the general case of partially-observed and nonlinear systems with disturbances, examines the potential loss of contraction and Lipschitzness in this case, introduces the new notion of d-tube contraction and Lipschitzness, proves novel converse results, and tests the Youla-REN in new numerical examples for both linear and nonlinear systems.

Concurrent work in [84, 85] also extends the Youla parametrisation to the contraction framework, and includes similar statements to Theorem 5.2-1 and 5.6. However, [84, 85] do not consider the effect of exogenous disturbances on the closed-loop system. Further concurrent work [50, 51, 52] studies the Youla parametrisation from the perspective of input-output operators, and also utilises RENs as the stable parameter. However, [50] only considers the state-feedback setting, and [51, 52] are limited to finite-gain stability for stable plants and make strong assumptions about measurable disturbances for the results on closed-loop contraction and Lipschitzness.

## 4.2 Preliminaries

### 4.2.1 Problem Formulation

We consider nonlinear systems $\mathcal{G}$ of the form

$$\mathcal{G}: \quad \begin{cases} x_{t+1} = f(x_t, \eta_t, u_t) + w_t \\ y_t = h(x_t) + v_t \end{cases} \tag{4.1}$$

with internal states $x_t \in \mathbb{R}^{n_x}$, controlled inputs $u_t \in \mathbb{R}^{n_u}$, and measured outputs $y_t \in \mathbb{R}^{n_y}$. The states and measurements are perturbed by (unknown) additive process disturbances $w_t \in \mathbb{R}^{n_x}$ and measurement noise $v_t \in \mathbb{R}^{n_y}$, respectively. Here $\eta_t \in \mathbb{R}^{n_\eta}$ are known external inputs representing reference signals, feedforward commands, disturbance previews, or other known inputs. We denote the unknown disturbances $d_t = [w_t; v_t]$ and the controlled outputs $z_t = [x_t; u_t]$.

Our task is to parametrise stabilising feedback policies $u = \mathcal{K}(\eta, y)$ for (4.1). Specifically, our goal is to construct a *policy class* parametrising the set of controllers $\mathcal{K}_\theta$, where $\theta \in \mathbb{R}^N$ is a learnable parameter, which satisfies the following requirements from Section 1.1 (re-stated below for convenience).

(1) **Robust stability certificates:** All controllers in the policy class guarantee robust closed-loop stability with (4.1).

(2) **Differentiable parametrisation:** There exists a differentiable (at least piecewise) map from the learnable parameter $\theta \in \mathbb{R}^N$ to feedback policies $\mathcal{K}_\theta$.

(3) **Expressive policy class:** The policy class is expressive and contains a large set of (preferably all) robustly stabilising controllers for a given system (4.1).

Requirements (1) and (2) are hard requirements which are essential for learning-based control with robust stability guarantees in data-driven problems. Requirement (1) ensures the policies are *always* closed-loop stabilising independently of the data or optimisation algorithms with which they have been trained. Requirement (2) ensures that the policy class is easy to use and compatible with standard tools in machine learning and data-driven control which are based on unconstrained optimisation. Requirement (3) is a soft requirement ensuring the policy class is sufficiently general to include high-performing controllers for a wide range of dynamical systems and objective functions.

### 4.2.2 Stability and Robustness

We introduced contraction, incremental IQCs, and their special case of Lipschitz-bounded systems in Definitions 2.1, 2.2, and 2.3, respectively. We choose these as our requirements for the internal stability (contraction) and input-output stability and robustness (incremental IQCs and Lipschitzness) due to the attractive properties of incremental stability notions outlined in Section 2.2.

Many existing approaches to parametrising stabilising controllers are based on weaker notions of stability, such as finite $\ell_2$-gain stability. We define this as follows. Consider a nonlinear system $u \mapsto y$ of the form

$$x_{t+1} = F(x_t, u_t), \quad y_t = H(x_t, u_t), \tag{2.12}$$

with states $x_t \in \mathbb{R}^n$, inputs $u_t \in \mathbb{R}^m$, outputs $y_t \in \mathbb{R}^p$, and $F, H$ locally Lipschitz. We assume $F(0,0) = 0$ and $H(0,0) = 0$ without loss of generality.

**Definition 4.1** (Finite-gain stability)**.** *The system* (2.12) *is said to be* finite-gain stable *if there exists a $\gamma \in \mathbb{R}^+$ such that for any initial state $x_0 \in \mathbb{R}^n$, inputs $u \in \ell^m$, and corresponding outputs $y \in \ell^p$, we have*

$$\|y\|_T \leq \gamma \|u\|_T + \kappa(x_0) \quad \forall T \in \mathbb{N} \tag{4.2}$$

*where $\kappa(x_0) \geq 0$ and $\kappa(0) = 0$.*

## 4.3 Nonlinear Youla Parametrisation

In this section, we construct the specific version of the Youla parametrisation that we use. We assume that a stabilising "base" controller has already been designed for the plant – or as a special case, where the plant itself is stable – and we wish to improve its closed-loop performance without compromising stability. Referring to Figure 4.2, we learn a dynamical system $\mathcal{Q}$ called the *Youla parameter* which adds controls $\tilde{u}$ to

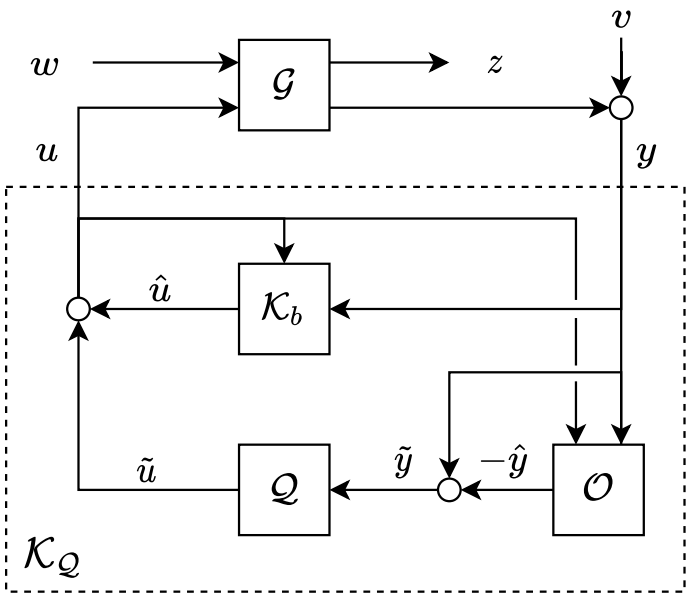



**Figure 4.2:** A version of the Youla-Kučera parametrisation. The Youla parameter $\mathcal{Q}$ augments a feedback controller $\mathcal{K}_b$ to control a plant $\mathcal{G}$. It reacts to difference $\tilde{y}$ between the measurements $y$ and predictions $\hat{y}$ from an observer $\mathcal{O}$. Known exogenous inputs $\eta$ have been omitted to simplify the diagram.

the output $\hat{u}$ of the base controller $\mathcal{K}_b$. The total control signal is then $u = \hat{u} + \tilde{u}$. We assume that $\mathcal{K}_b$ takes the form

$$\mathcal{K}_b : \quad \begin{cases} s_{t+1} = f_b(s_t, \eta_t, u_t, y_t) \\ \hat{u}_t = k(s_t, \eta_t, y_t) \end{cases} \tag{4.3}$$

where $s_t \in \mathbb{R}^{n_s}$ is some internal state. The first key ingredient in our parametrisation is an observer $\hat{y} = \mathcal{O}(\eta, u, y)$ of the form

$$\mathcal{O} : \quad \begin{cases} \hat{x}_{t+1} = f_o(\hat{x}_t, \eta_t, u_t, y_t), \\ \hat{y}_t = h(\hat{x}_t) \end{cases} \tag{4.4}$$

where $\hat{x}_t \in \mathbb{R}^{n_x}$ is the state estimate and $\hat{y}_t \in \mathbb{R}^{n_y}$ is the output estimate. The *innovations* sequence is

$$\tilde{y}_t := y_t - \hat{y}_t. \tag{4.5}$$

Intuitively, the role of the observer is to decompose the measurements $y = \hat{y} + \tilde{y}$ into predictable measurements $\hat{y}$ and "surprises" $\tilde{y}$. The second key ingredient is the Youla

parameter $\tilde{u} = \mathcal{Q}(\eta, \tilde{y})$, which acts on the innovations, "reacting to surprises" as well as known exogenous inputs $\eta$. $\mathcal{Q}$ is itself a nonlinear dynamical system of the form

$$\mathcal{Q}: \quad \begin{cases} q_{t+1} = f_q(q_t, \eta_t, \tilde{y}_t), \\ \tilde{u}_t = h_q(q_t, \eta_t, \tilde{y}_t) \end{cases} \tag{4.6}$$

with internal state $q_t \in \mathbb{R}^{n_q}$ and augmenting control $\tilde{u}_t \in \mathbb{R}^{n_u}$. The resulting augmented controller $u = \mathcal{K}_{\mathcal{Q}}(\eta, y)$ is

$$\mathcal{K}_{\mathcal{Q}}: \quad \begin{cases} s_{t+1} = f_b(s_t, \eta_t, u_t, y_t) \\ \hat{x}_{t+1} = f_o(\hat{x}_t, \eta_t, u_t, y_t) \\ q_{t+1} = f_q(q_t, \eta_t, y_t - h(\hat{x}_t)) \\ u_t = k(s_t, \eta_t, y_t) + h_q(q_t, \eta_t, y_t - h(\hat{x}_t)). \end{cases} \tag{4.7}$$

**Remark 4.1.** *If external inputs $\eta$ are not provided to the Youla parameter such that $\tilde{u} = \mathcal{Q}(\tilde{y})$, then the Youla parameter* only *reacts to surprises – i.e., disturbances, model error, or initial condition error. In this case, we can think of the controller* (4.7) *as consisting of a nominal controller $\mathcal{K}_b$ with good performance in response to known inputs, and a robustifying parameter $\mathcal{Q}$ whose role is to respond to error and uncertainty. This perspective is explored in the linear setting in [197].*

## 4.4 Results for Simplified Settings

Our aim is to study the Youla controller $\mathcal{K}_{\mathcal{Q}}$ as a parametrisation of policies for (a) nonlinear (b) partially-observed systems achieving (c) contracting and Lipschitz closed loops. In this section, we first consider simplified settings in which at most two of these three features are present at a time, namely: (b), (c) but not (a) (Sec. 4.4.1); (a), (c) but not (b) (Sec. 4.4.2); and (a), (b) but not (c) (Sec. 4.4.3), where the stability requirement is relaxed to finite-gain stability. In these settings, we find that a controller stabilises the closed-loop system if and only if it can be written in the Youla form.

### 4.4.1 Restricting to LTI Systems

Consider the simplified setting in which (4.1) is an LTI system

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t + w_t \\ y_t &= Cx_t + v_t \end{aligned} \tag{4.8}$$

with $(A, B)$ stabilisable and $(A, C)$ detectable. Then, we can obtain a linear base controller $\hat{u} = \mathcal{K}_b(y)$ as

$$\begin{aligned} \hat{x}_{t+1} &= A\hat{x}_t + Bu_t + L(y_t - C\hat{x}_t) \\ \hat{u}_t &= -K\hat{x}_t \end{aligned} \tag{4.9}$$

where the gain matrices $K, L$ are chosen such that $(A - BK)$ and $(A - LC)$ are stable using standard methods from linear control theory (such as Linear Quadratic Gaussian (LQG) control [196]). By taking the innovations $\tilde{y} = y - C\hat{x}$, the Youla controller (4.7) becomes

$$u = -K\hat{x} + \mathcal{Q}(\eta, \tilde{y}) \tag{4.10}$$

where $\mathcal{Q}$ is the nonlinear system (4.6).

If $\mathcal{Q}$ is an LTI system, it is well-known that (4.10) parametrises all linear stabilising controllers for the plant (4.8) if and only if $\mathcal{Q}$ itself is a stable system [41, 3]. However, there are many situations in which parametrising a nonlinear controller can be advantageous, even if the system to be controlled is linear. For example, one may consider LTI systems with constrained states or inputs, or LTI systems with non-quadratic cost functions. In both cases, the optimal control can be nonlinear.

The following theorem extends the classic Youla-Kučera result to nonlinear Youla parameters $\mathcal{Q}$ of the form (4.6) and nonlinear controllers $u = \mathcal{K}(\eta, y)$ described by

$$\mathcal{K}: \quad \begin{cases} \phi_{t+1} = f_{\mathcal{K}}(\phi_t, \eta_t, y_t) \\ u_t = h_{\mathcal{K}}(\phi_t, \eta_t, y_t) \end{cases} \tag{4.11}$$

with states $\phi \in \mathbb{R}^{n_k}$, and $f_\mathcal{K}, h_\mathcal{K}$ locally Lipschitz. Put simply, the theorem states that the Youla parametrisation covers all and only the controllers achieving a contracting and Lipschitz closed loop with a given LTI system.

**Theorem 4.1.** *Consider the LTI system* (4.8) *with controller* (4.9), (4.10) *parametrised by* $\mathcal{Q}$ (4.6).

1. *For any contracting and Lipschitz* $\mathcal{Q}$*, the closed-loop system is contracting and the map* $(\eta, d) \mapsto z$ *is Lipschitz.*

2. *Any controller* $u = \mathcal{K}(\eta, y)$ *of the form* (4.11) *achieving a contracting and Lipschitz closed loop can be written in the form* (4.9), (4.10) *with contracting and Lipschitz* $\mathcal{Q}$*.*

*Proof of Theorem 4.1-1.* The result follows directly from Theorem 5.4. However, we include a detailed proof below as it provides useful insight into the advantages of the Youla parametrisation. We ignore $\eta$ for notational convenience.

We will show below that, under the base controller (4.9) with a controller augmentation $\tilde{u}$, the system response is described by

$$\begin{bmatrix} z \\ \tilde{y} \end{bmatrix} = \begin{bmatrix} \mathcal{T}_0 & \mathcal{T}_1 \\ \mathcal{T}_2 & 0 \end{bmatrix} \begin{bmatrix} d \\ \tilde{u} \end{bmatrix} \tag{4.12}$$

where $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2$ are each stable linear systems. Importantly, $\tilde{y}$ is completely decoupled from $\tilde{u}$ in (4.12). This allows us to close the loop by connecting $\tilde{u} = \mathcal{Q}(\tilde{y})$ in series with $\mathcal{T}_1, \mathcal{T}_2$ to achieve

$$z = \mathcal{T}_0(d) + \mathcal{T}_1(\mathcal{Q}(\mathcal{T}_2(d))). \tag{4.13}$$

The closed-loop response is therefore a series/parallel interconnection of the stable linear systems and $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2$ and a contracting and Lipschitz system $\mathcal{Q}$. Hence, it is contracting and Lipschitz by the series/parallel composition properties of contracting

systems and Lipschitz mappings.

It remains to construct the stable systems $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2$ in (4.12), for which we follow the approach in [196]. First, let us define the observer error $\tilde{x} := x - \hat{x}$. The closed-loop dynamics of (4.8) in feedback with the base controller (4.9) are

$$x_{t+1} = (A - BK)x_t + BK\tilde{x}_t + w_t \tag{4.14a}$$

$$\tilde{x}_{t+1} = (A - LC)\tilde{x}_t - Lv_t + w_t \tag{4.14b}$$

$$\tilde{y}_t = C\tilde{x}_t + v_t \tag{4.14c}$$

$$z_t = [x_t; K(\tilde{x}_t - x_t)]. \tag{4.14d}$$

Equations (4.14a), (4.14b) and (4.14d) define the stable closed-loop response under the base controller, $\mathcal{T}_0 : d \mapsto z$. Similarly, (4.14b) and (4.14c) define the stable system $\mathcal{T}_2 : d \mapsto \tilde{y}$. To construct $\mathcal{T}_1$, we add $\tilde{u} = u - \hat{u} = u + K(x - \tilde{x})$ to the base controller signal $\hat{u} = K(\tilde{x} - x)$. This only modifies (4.14a) and (4.14d) such that

$$\begin{aligned} x_{t+1} &= (A - BK)x_t + BK\tilde{x}_t + B\tilde{u}_t + w_t \\ z_t &= [x_t; \tilde{u}_t + K(\tilde{x}_t + x_t)]. \end{aligned}$$

Hence, by the superposition of linear systems, we have that $z = \mathcal{T}_0(d) + \mathcal{T}_1(\tilde{u})$, where the stable linear system $\mathcal{T}_1 : \tilde{u} \mapsto z$ is given by

$$\begin{aligned} \breve{x}_{t+1} &= (A - BK)\breve{x}_t + B\tilde{u}_t \\ \breve{z}_t &= [\breve{x}_t; \tilde{u}_t - K\breve{x}_t]. \end{aligned}$$

Inserting these stable linear systems $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2$ into (4.12) completes the proof. □

*Proof of Theorem 4.1-2.* The result follows directly from Corollary 5.8. □

Before moving on, we note two key features of the Youla parametrisation for LTI systems which are both highlighted in the proof of Theorem 4.1-1. Firstly, (4.12) implies

that the innovations $\tilde{y}$ are completely decoupled from the output $\tilde{u}$ of the Youla parameter $\mathcal{Q}$, which is fundamentally why closed-loop stability is guaranteed with contracting and Lipschitz $\mathcal{Q}$. We will explore this property further in Section 5.1.4. Secondly, the closed-loop response (4.13) is linear in $\mathcal{Q}$. If $\mathcal{Q} = \mathcal{Q}_\theta$ is linearly parametrised by a vector $\theta \in \mathbb{R}^N$, then one can pose the problem of nonlinear control design as a convex optimisation problem. This perspective is taken in, for example, [136, Sec. IX] using RENs as echo-state networks [192].

### 4.4.2 Restricting to Perfect State Feedback

In our second simplified setting, we consider cases in which the plant (4.1) is nonlinear but we have perfect knowledge of the states such that

$$\begin{aligned} x_{t+1} &= f(x_t, \eta_t, u_t) + w_t \\ y_t &= x_t \end{aligned} \tag{4.15}$$

where $f(x_{-1}, \eta_{-1}, u_{-1}) := 0$ and $w_{-1} := x_0$. This scenario is relevant if, for example, we have a perfect disturbance observer. Suppose the base controller (4.3) takes the form $\hat{u}_t = k(\eta_t, y_t)$. The Youla parametrisation (4.7) can then be reduced to

$$u = k(\eta, x) + \mathcal{Q}(\eta, \tilde{y}) \tag{4.16}$$

where the innovations at time $t$ are simply

$$\tilde{y}_t = w_{t-1} = x_t - f(x_{t-1}, \eta_{t-1}, u_{t-1}) \tag{4.17}$$

since we have a perfect model of the plant and no measurement noise. A similar statement to Theorem 4.1 is then as follows.

**Theorem 4.2.** *Suppose that $k$ is Lipschitz, the closed-loop system under the base controller $\hat{u} = k(\eta, x)$ is contracting, and its map $(\eta, w, \tilde{u}) \mapsto z$ is Lipschitz. Consider the system* (4.15) *with controller* (4.16) *parametrised by* $\mathcal{Q}$ (4.6).

1. *For any contracting and Lipschitz $\mathcal{Q}$, the closed-loop system is contracting and its map $(\eta, d) \mapsto z$ is Lipschitz.*

2. *Any controller $u = \mathcal{K}(\eta, y)$ of the form* (4.11) *achieving a contracting and Lipschitz closed loop can be written in the form* (4.16) *with contracting and Lipschitz $\mathcal{Q}$.*

*Proof of Theorem 4.2.* The results follow directly from Theorem 5.4 and Corollary 5.7, respectively. □

As with the LTI setting, the Youla parametrisation represents all and only contracting and Lipschitz closed-loops with (4.15). This state-feedback formulation has strong connections to disturbance-feedback parametrisations in MPC [56]. Note also that similar results to Theorem 4.2 in terms of finite-gain stability have been derived for System-Level Synthesis (SLS) in [66, 50].

### 4.4.3 Restricting to Finite-Gain Stability

Moving to the partially-observed nonlinear setting, similar results have been proven for the weaker notion of finite-gain stability (e.g., [74, 48]). We summarise these results for our specific problem setup given the following assumptions.

**A1′** *Finite-gain stabilising base controller:* the closed-loop system $\mathcal{G}_{\mathcal{K}_b} : (\eta, d, \tilde{u}) \mapsto z$ under the base controller, i.e., (4.1) in feedback with (4.3), is finite-gain stable.

**A2** *Contracting and Lipschitz observer:* The observer (4.4) is contracting and the map $(\eta, u, y) \mapsto \hat{x}$ is Lipschitz.

**A3** *Observer correctness:* When $d \equiv 0$ and $x_0 = \hat{x}_0$, the observer exactly replicates the plant dynamics. That is, $f(x_t, \eta_t, u_t) = f_o(x_t, \eta_t, u_t, h(x_t))$ $\forall t \in \mathbb{N}$.

**A4** *Smooth functions:* The systems $\mathcal{G}, \mathcal{K}_b, \mathcal{O}, \mathcal{Q}, \mathcal{K}$ have piecewise-differentiable dynamics functions and Lipschitz continuous output maps.

Comparing to the partially-observed LTI setting, assumptions **A1′**–**A3** are similar to assuming that we can design a stabilising linear controller (4.9). Importantly, **A2**–**A3** imply that the observer error exponentially decays when there are no disturbances, regardless of the control inputs [136, Prop. 2]. Under these assumptions, the Youla parametrisation achieves the following result (similar to e.g., [48, Prop. 21], [74, Thm. 3.1]), which implies that the Youla parametrisation represents all and only finite-gain stabilising controllers for a given partially-observed nonlinear system.

**Theorem 4.3.** *Suppose assumptions **A1′**–**A4** hold. Consider the system* (4.1) *with controller* (4.7) *parametrised by* $\mathcal{Q}$ (4.6)*.*

1. *For any finite-gain stable* $\mathcal{Q}$*, the closed-loop system is also finite-gain stable.*
2. *If the base controller* (4.3) *is finite-gain stable, then any controller* $u = \mathcal{K}(\eta, y)$ *of the form* (4.11) *achieving a finite-gain stable closed loop can be written in the form* (4.7) *with finite-gain stable* $\mathcal{Q}$*.*

**Remark 4.2.** *[48, Prop. 21], [74, Thm. 3.1] use a similar construction to* (4.7)*, but with the base controller and observer coupled in the form*

$$\hat{x}_{t+1} = f_o(\hat{x}_t, u_t, y_t), \quad \hat{u}_t = k(\hat{x}_t, y_t)$$

*rather than separated out as in* (4.3) *and* (4.4)*, respectively. We therefore provide a proof for our specific statement of Theorem 4.3 for completeness.*

*Proof of Theorem 4.3-1.* Without loss of generality, assume that $f(0,0,0) = 0, h(0) = 0$ and similarly for all other functions. We first show that the innovations signal $\tilde{y}$ is bounded by a term that is independent of the Youla augmentation $\tilde{u}$, despite there being a feedback loop between $\tilde{u}$ and $\tilde{y}$ in Figure 4.2. Applying observer correctness

(**A3**) to the nonlinear system $\mathcal{G}$, we have

$$\begin{aligned} x_{t+1} &= f(x_t, \eta_t, u_t) + w_t \\ &= f_o(x_t, \eta_t, u_t, h(x_t)) + w_t \\ &= f_o(x_t, \eta_t, u_t, y_t - v_t) + w_t. \end{aligned} \tag{4.18}$$

We denote the trajectory of (4.18) by $(x, \eta, u, y, w, v)$. The observer dynamics are

$$\hat{x}_{t+1} = f_o(\hat{x}_t, \eta_t, u_t, y_t) \tag{4.4}$$

with $(\hat{x}, \eta, u, y)$ as its trajectory. This shares the same inputs $(\eta, u, y)$ as (4.18). Since the observer is contracting and Lipschitz (**A2**), from Lemma 2.1 we have

$$\|\tilde{x}\|_T = \|x - \hat{x}\|_T \leq \tilde{\gamma}(\|v\|_T + \|w\|_T) + \tilde{\kappa}(x_0, \hat{x}_0), \tag{4.19}$$

where $\tilde{\gamma} \in \mathbb{R}^+$ and $\tilde{\kappa}(x_0, \hat{x}_0) \geq 0$ with $\tilde{\kappa}(x_0, x_0) = 0 \; \forall \, x_0 \in \mathbb{R}^{n_x}$. Since the output map $h$ is Lipschitz (**A4**), we obtain

$$\begin{aligned} \|\tilde{y}\|_T &= \|h(x) - h(\hat{x}) + v\|_T \leq \gamma_h \|\tilde{x}\|_T + \|v\|_T \\ &\leq \gamma_0(\|w\|_T + \|v\|_T) + \kappa_0(x_0, \hat{x}_0) \end{aligned} \tag{4.20}$$

where $\gamma_0 = \tilde{\gamma}\gamma_h + 1$ and $\kappa_0 = \gamma_h \tilde{\kappa}$ with $\gamma_h$ as the Lipschitz constant of $h$.

We now show that the closed-loop system is finite-gain stable. Let $\xi_t = [x_t; s_t] \in \mathbb{R}^{n_x + n_s}$ and $q_t \in \mathbb{R}^{n_q}$ be the states of the base-controlled closed-loop system $(\eta, d, \tilde{u}) \mapsto z$ and the Youla parameter $(\eta, \tilde{y}) \mapsto \tilde{u}$, respectively. For convenience we denote the state of the total closed-loop system by $\bar{x}_t = [\xi_t; \hat{x}_t; q_t]$. From assumption **A1′** and the finite-gain stability of the Youla parameter respectively, we have

$$\begin{aligned} \|z\|_T &\leq \gamma_1(\|\tilde{u}\|_T + \|\eta\|_T + \|w\|_T + \|v\|_T) + \kappa_1(\xi_0), \\ \|\tilde{u}\|_T &\leq \gamma_2(\|\tilde{y}\|_T + \|\eta\|_T) + \kappa_2(q_0), \end{aligned}$$

for some $\gamma_1, \gamma_2 \in \mathbb{R}^+$ and $\kappa_1(\xi_0), \kappa_2(q_0) \geq 0$ with $\kappa_i(0) = 0$ for $i = 1, 2$. Combining these inequalities with (4.20) gives

$$\|z\|_T \leq \gamma(\|\eta\|_T + \|w\|_T + \|v\|_T) + \kappa(\bar{x}_0) \tag{4.21}$$

with $\gamma = \gamma_1 (1 + \gamma_2 \max(\gamma_0, 1))$ and $\kappa = \gamma_1\gamma_2\kappa_0 + \kappa_1 + \gamma_1\kappa_2$. Noting that $x_0 = \hat{x}_0 = 0$ implies $\kappa(0) = 0$ completes the proof. □

*Proof of Theorem 4.3-2.* The closed-loop system $(\eta, d) \mapsto z$ consisting of (4.1) in feedback with $u = \mathcal{K}(\eta, y)$ (4.11) has dynamics

$$\begin{aligned} x_{t+1} &= f(x_t, \eta_t, h_\mathcal{K}(\phi_t, \eta_t, h(x_t) + v_t)) + w_t \\ \phi_{t+1} &= f_\mathcal{K}(\phi_t, \eta_t, h(x_t) + v_t) \\ z_t &= [x_t; h_\mathcal{K}(\phi_t, \eta_t, h(x_t) + v_t)]. \end{aligned} \tag{4.22}$$

This system is finite-gain stable by assumption. We will use it to show that $\mathcal{K}$ can be re-written in the form (4.7) with a finite-gain stable Youla parameter $\mathcal{Q}_\mathcal{K}$.

Augment $\mathcal{K}$ with a base controller $\mathcal{K}_b$ (4.3) and observer $\mathcal{O}$ (4.4). Using $y_t = \tilde{y}_t + h(\hat{x}_t)$ and $\tilde{u}_t = u_t - \hat{u}_t$, the resulting system $\mathcal{Q}_\mathcal{K} : (\eta, \tilde{y}) \mapsto \tilde{u}$ has state-space form

$$\begin{aligned} \hat{x}_{t+1} &= f_o(\hat{x}_t, \eta_t, h_\mathcal{K}(\phi_t, \eta_t, h(\hat{x}_t) + \tilde{y}_t), h(\hat{x}_t)) + \Delta f_o \\ s_{t+1} &= f_b(s_t, \eta_t, h_\mathcal{K}(\phi_t, \eta_t, h(\hat{x}_t) + \tilde{y}_t), h(\hat{x}_t) + \tilde{y}_t) \\ \phi_{t+1} &= f_\mathcal{K}(\phi_t, \eta_t, h(\hat{x}_t) + \tilde{y}_t) \\ \tilde{u}_t &= h_\mathcal{K}(\phi_t, \eta_t, h(\hat{x}_t) + \tilde{y}_t) - k(s_t, \eta_t, h(\hat{x}_t) + \tilde{y}_t), \end{aligned} \tag{4.23}$$

where $\Delta f_o := f_o(\hat{x}_t, \eta_t, u, h(\hat{x}_t) + \tilde{y}_t) - f_o(\hat{x}_t, \eta_t, u, h(\hat{x}_t))$. We show that $\mathcal{Q}_\mathcal{K}$ is finite-gain stable in two steps. First, notice the $\mathcal{Q}_\mathcal{K}$ is composed of a system $\mathcal{K}$ in feedback with $\mathcal{O}$, connected in parallel with the system $\mathcal{K}_b$ (see Section 5.2 and Figure 5.3 for further discussion on this interpretation). Applying observer correctness (**A3**) to the

first system, we can re-write its internal dynamics as

$$\begin{aligned}\hat{x}_{t+1} &= f(\hat{x}_t, \eta_t, h_{\mathcal{K}}(\phi_t, \eta_t, h(\hat{x}_t) + \tilde{y}_t)) + \Delta f_o \\ \phi_{t+1} &= f_{\mathcal{K}}(\phi_t, \eta_t, h(\hat{x}_t) + \tilde{y}_t).\end{aligned} \tag{4.24}$$

This is exactly the internal dynamics of the finite-gain stable system (4.22) if we re-label $x$ as $\hat{x}$ and choose $v_t = \tilde{y}_t$, $w_t = \Delta f_o$. Furthermore, we know $|\Delta f_o| \leq \gamma_{oy}|\tilde{y}_t|$ for some $\gamma_{oy} \in \mathbb{R}^+$ because $f_o$ is Lipschitz *w.r.t* the measurements by assumption. Hence the map $(\eta, \tilde{y}) \mapsto z$ is finite-gain stable under our transformation.

Second, we use the fact that, under our re-labelling, the map $z \mapsto \tilde{u}$ is given by $\tilde{u}_t = [0, I]z_t - \hat{u}_t$ where, with slight abuse of notation,

$$\hat{u} = \mathcal{K}_b(\eta, [0\ I]z, h([I\ 0]z) + \tilde{y})$$

is finite-gain stable by assumption. Since $h$ is also Lipschitz (**A4**), then clearly the map $z \mapsto \tilde{u}$ is also finite-gain stable. Therefore, $\mathcal{Q}_{\mathcal{K}} : (\eta, \tilde{y}) \mapsto \tilde{u}$ is finite-gain stable since it is a parallel connection of finite-gain stable systems. To conclude, notice that if $\mathcal{Q} = \mathcal{Q}_{\mathcal{K}}$ in the Youla architecture (4.7), then the control signal $u_t = \hat{u}_t + h_{\mathcal{K}}(\phi_t, \eta_t, y_t) - \hat{u}_t$ is unchanged from $u = \mathcal{K}(\eta, y)$. Hence $\mathcal{K}$ can be written in the form (4.7) with a finite-gain stable Youla parameter $\mathcal{Q} = \mathcal{Q}_{\mathcal{K}}$. □

## 4.5 Conclusions

This chapter has introduced our study of a nonlinear version of the Youla parametrisation for learning-based control with built-in stability and robustness guarantees. We have seen that in simplified settings with at most two of three complicating factors – (a) nonlinear systems, (b) partially-observed systems, (c) and incremental stability – the Youla parametrisation covers all and only stabilising controllers for a given system, in analogy with the classical results for linear systems and controllers from [193, 92].

# Chapter 5

# React to Surprises with Youla-REN – Part II

We now continue our study of the Youla parametrisation from Chapter 4, where we established it as a parametrisation of all stabilising controllers in simplified settings.

This chapter provides detailed theoretical results for the general setting of partially-observed nonlinear systems with incremental stability. We study the conditions under which: (1) the Youla parametrisation guarantees contracting and Lipschitz closed-loops; and (2) the Youla parametrisation covers all controllers achieving contracting and Lipschitz closed loops. We follow this with numerical experiments combining our version of the Youla parametrisation with RENs to construct our stable-by-design policy class, the Youla-REN. We demonstrate the advantages of learning Youla-REN policies – which have built-in stability certificates – on three illustrative examples in deep RL with: (i) "economic" costs which do not naturally encourage closed-loop stability; (ii) short training horizons; and (iii) model uncertainty.

This chapter is based on the same three publications as listed in Chapter 4. The majority of the content is based on [15], which builds on our previous work in [12, 178].

## 5.1 Partially-Observed Nonlinear Systems with Incremental Stability

In this section, we study the case of partially-observed nonlinear systems (4.1) and the incremental stability properties of contraction and Lipschitzness of the closed loop. We first show by counterexample that incremental stability can be lost in this context. We follow this by showing that a weaker-form of stability can be maintained, and discuss stronger conditions under which contracting and Lipschitz closed loops can still be achieved. All proofs are provided in Section 5.5.

We first introduce a stronger version of assumption **A1′** which will be used for all subsequent results in this section

**A1** *Contracting and Lipschitz stabilising base controller:* the closed-loop system $\mathcal{G}_{\mathcal{K}_b} : (\eta, d, \tilde{u}) \mapsto z$ under the base controller, i.e., (4.1) in feedback with (4.3), is contracting and Lipschitz.

### 5.1.1 Counterexample Exhibiting Loss of Contraction

One might expect a similar result to Theorems 4.1 to 4.3 to hold for partially-observed nonlinear systems in the incremental stability setting – i.e., if the base controller satisfies assumptions **A1**–**A4** and the Youla parameter is contracting and Lipschitz, then the Youla controller (4.7) parametrises all contracting and Lipschitz closed loops $(\eta, d) \mapsto z$ for partially-observed nonlinear systems (4.1). However, Example 5.1 shows a system for which these assumptions are satisfied, but the closed-loop system is neither contracting nor Lipschitz.

**Example 5.1.** *Consider the following scalar nonlinear system*

$$x_{t+1} = 0.5|x_t| + u_t + w_t, \quad y_t = x_t \tag{5.1}$$

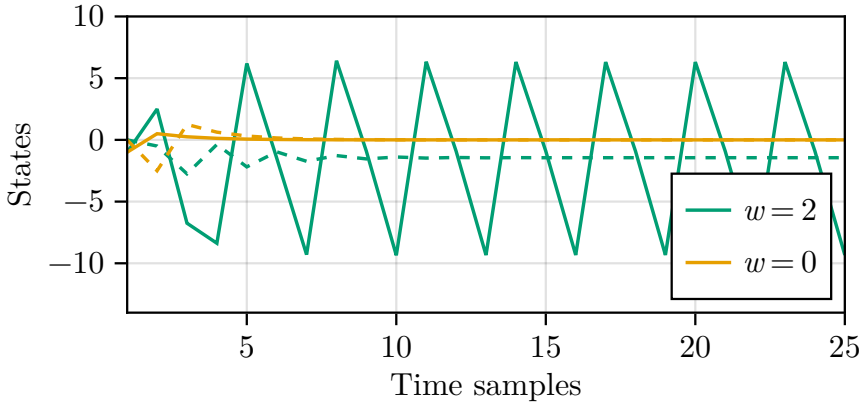



**Figure 5.1:** Simulations of the closed-loop system in Example 5.1 show plant state trajectories from different initial states (solid and dashed). They converge when $w = 0$ but with $w = 2$ there are distinct trajectories that do not converge, so the system is neither contracting nor Lipschitz.

*with $w_t$ as the unknown disturbance. Assumption **A1** holds with the base controller $\mathcal{K}_b = 0$. We then choose an observer*

$$\hat{x}_{t+1} = 0.5|\hat{x}_t| + u_t, \quad \hat{y}_t = \hat{x}_t, \tag{5.2}$$

*which satisfies assumptions **A2**–**A4**. Finally, we pass the innovations $\tilde{y} = y - \hat{y}$ to a memoryless Youla parameter*

$$u = \mathcal{Q}(\tilde{y}) := \min(2.5 - 5\tilde{y}, 0), \tag{5.3}$$

*which is obviously contracting and Lipschitz. As shown in Figure 5.1 (green), the closed-loop state trajectories under the same disturbance $w \equiv 2$ converge to multiple trajectories, and hence the closed-loop system under the nonlinear Youla controller $\mathcal{K}_\mathcal{Q}$ is neither contracting nor Lipschitz.*

To understand the behaviour of the system (5.1)–(5.3), we first derive the dynamics of the observer error $\tilde{x} = x - \hat{x}$ as follows,

$$\tilde{x}_{t+1} = 0.5|\tilde{x}_t + \hat{x}_t| - 0.5|\hat{x}_t| + w_t, \quad \tilde{y}_t = \tilde{x}_t, \tag{5.4}$$

which implies that the innovations $\tilde{y}$ implicitly depend on the control input $u$ (the output of $\mathcal{Q}$) through $\hat{x}$. This introduces a feedback loop between the error dynamics (5.4) and the Youla parameter $\mathcal{Q}$ (5.3), leading to non-contracting closed-loop behaviour when the gain of $\mathcal{Q}$ is sufficiently large.

On the other hand, we can establish that increasing the gain of $\mathcal{Q}$ does not lead to unbounded closed-loop trajectories with bounded disturbances. The reason is as follows. Both the plant (5.1) and its observer (5.2) are copies of the following contracting and Lipschitz system

$$\xi_{t+1} = 0.5|\xi_t| + \mu_t, \quad \nu_t = \xi_t \tag{5.5}$$

with $(\xi^a, \mu^a, \nu^a) = (x, u + w, y)$ and $(\xi^b, \mu^b, \nu^b) = (\hat{x}, u, \hat{y})$ for the plant and observer, respectively. From (2.14) we have

$$\|\tilde{y}\|_T = \|y - \hat{y}\|_T \leq \gamma \|w\|_T + \kappa(x_0, \hat{x}_0) \tag{5.6}$$

with $\gamma \geq 0$ and $\kappa(a, b) \geq 0$. In other words, the innovations $\tilde{y}$ are bounded by the external disturbance $w$, despite there being feedback through $u$, which might be large due to some high-gain $\mathcal{Q}$. Moreover, since the Youla parameter $\mathcal{Q}$ and the system under the base controller are both contracting and Lipschitz, we can conclude that the $\ell_2$-norms of $u, x, y$ are also bounded.

### 5.1.2 d-Tube Contracting and Lipschitz Responses

The counterexample above shows that closed-loop contraction and Lipschitzness cannot be guaranteed for partially-observed nonlinear systems (4.1). However, we observed that signals did not exhibit *unbounded* growth, and they instead satisfied a finite $\ell_2$ gain bound from the unknown disturbance $w$. The main result of this section establishes this as a general property in the partially-observed nonlinear setting. In the following, $\bar{x}_t := [x_t; s_t; \hat{x}_t; q_t]$ is the state of the closed-loop system under the Youla controller (4.7).

**Theorem 5.1.** *Suppose assumptions **A1**–**A4** hold. Then for any contracting and Lipschitz Youla parameter $\mathcal{Q}$ (4.6), the closed-loop system consisting of (4.1) in feedback with (4.7) satisfies the following properties:*

1. *With zero disturbances (i.e., $d \equiv 0$, $\tilde{x}_0 = 0$), the closed-loop system is contracting and $\eta \mapsto z$ is Lipschitz.*

2. *Let $(\bar{x}^\star, \eta^\star, z^\star) \in \ell$ be any zero-disturbance trajectory of the closed-loop system. Then with non-zero disturbances (i.e., $\forall d \in \ell$, $x_0 \neq \hat{x}_0$), the closed-loop system satisfies*

$$\|z - z^\star\|_T \leq \gamma(\|\eta - \eta^\star\|_T + \|d\|_T) + \kappa(\bar{x}_0, \bar{x}_0^\star) \tag{5.7}$$

*for all $T \in \mathbb{N}$, $\bar{x}_0 \in \mathbb{R}^{n_{\bar{x}}}$, $\eta \in \ell$, where $\gamma \in \mathbb{R}^+$, $\kappa(\bar{x}_0, \bar{x}_0^\star) \geq 0$, $\kappa(\bar{x}_0^\star, \bar{x}_0^\star) = 0$, and $n_{\bar{x}} = 2n_x + n_q + n_s$. We call this property* d-tube contracting and Lipschitz.

The interpretation of Theorem 5.1 is that in the disturbance-free case, the closed-loop system exhibits smooth responses to *known* external inputs $\eta$ (e.g., disturbance previews, reference trajectories, feedforward controls) as they can be seen by both the plant and the observer. However, with unknown disturbances present, the system contracts to a tube around the disturbance-free trajectory with a radius proportional to $\|d\|$. This is a stronger property than the finite-gain stability of Theorem 4.3, but weaker than the closed-loop contraction and Lipschitzness of Theorems 4.1 and 4.2.

**Remark 5.1.** *The proof of Theorem 5.1 (Section 5.5) does not actually rely on the base controller achieving contracting and Lipschitz closed loops with respect to $d$. Assumption **A1** can be replaced with the weaker assumption that the base-controlled closed-loop system $\mathcal{G}_{\mathcal{K}_b} : (\eta, d, \tilde{u}) \mapsto z$ is d-tube contracting and Lipschitz.*

### 5.1.3 Response to Observer Initial Condition Error

Theorem 5.1 indicates that the Youla parametrisation guarantees contracting and Lipschitz closed-loops in the disturbance-free setting with zero initial observer error, but

not in the presence of disturbances and initial observer error. An interesting special case is when we have non-zero initial observer error, but no other unknown perturbations.

Consider Example 5.1 again, now with $w \equiv 0$. Since both $(x, u, y)$ and $(\hat{x}, u, \hat{y})$ are solutions of the contracting system (5.5) with the same inputs $u$, from Definition 2.1 we have $|\tilde{x}_t| = |x_t - \hat{x}_t| \leq \beta\alpha^t |x_0 - \hat{x}_0|$ for some $\beta > 0$ and $\alpha \in [0, 1)$. This means that the feedback from $\mathcal{Q}$ to $\tilde{y}$ is constrained within an envelope that exponentially converges to zero. Since the Youla parameter $\mathcal{Q}$ and the system under the base controller $\mathcal{K}_b$ are both contracting, we conclude that both $x, u$ will exponentially converge to some fixed values, which we see empirically in Figure 5.1 (yellow). The following theorem formalises the closed-loop stability result when the unknown disturbances are zero.

**Theorem 5.2.** *Suppose assumptions **A1**–**A4** hold, $x_0 \neq \hat{x}_0$ but $d \equiv 0$, and $f, f_b, f_q$ are Lipschitz continuous w.r.t. their respective system inputs. Consider any two trajectories $z^a, z^b \in \ell$ of the closed-loop system (4.1) in feedback with (4.7) with initial states $\bar{x}_0^a, \bar{x}_0^b \in \mathbb{R}^{n_{\bar{x}}}$ and inputs $\eta^a, \eta^b \in \ell$. Then for any contracting and Lipschitz Youla parameter $\mathcal{Q}$ (4.6):*

1. *When $\eta^a \equiv \eta^b$, the closed-loop states satisfy*

$$|\Delta\bar{x}_t| \leq A\rho^t \left(|\Delta\xi_0| + |\Delta q_0| + |\tilde{x}_0^a| + \left|\tilde{x}_0^b\right|\right) \tag{5.8}$$

   *for some $A \in \mathbb{R}^+$ and $\rho \in [0, 1)$, where $\Delta x := x^a - x^b$ and similar for all other variables, and $\xi_t := [x_t; s_t]$.*

2. *The closed-loop trajectories satisfy*

$$\|\Delta z\|_T \leq \bar{\gamma}\|\Delta\eta\|_T + \kappa(\bar{x}_0^a, \bar{x}_0^b) + \frac{\tilde{\gamma}\beta\left(|\tilde{x}_0^a| + \left|\tilde{x}_0^b\right|\right)}{\sqrt{1-\alpha^2}} \tag{5.9}$$

   *for all $T \in \mathbb{N}$, where $\bar{\gamma}, \tilde{\gamma} \in \mathbb{R}^+$, $\kappa(\bar{x}_0^a, \bar{x}_0^b) \geq 0$ with $\kappa(\bar{x}_0, \bar{x}_0) = 0 \; \forall \, \bar{x}_0 \in \mathbb{R}^{n_{\bar{x}}}$, and $\beta \in \mathbb{R}^+, \alpha \in [0, 1)$ are the overshoot and contraction rate of the observer (4.4).*

**Remark 5.2.** *The effect of non-zero initial observer error is that the closed-loop system is not contracting and Lipschitz in the sense of Definitions 2.1 and 2.3 (respectively). Instead,* (5.8) *and* (5.9) *have additional terms depending on* $|\tilde{x}_0^a|, |\tilde{x}_0^b|$*, which are not necessarily zero even if* $\Delta\tilde{x}_0 = 0$*. These terms are due to the transient feedback interaction between* $\tilde{u}$ *and* $\tilde{y}$ *before the observer error* $\tilde{x}$ *converges to 0. We therefore called this behaviour* contracting and Lipschitz with transients *in [12], and it is consistent with recent results in [85, Thm. 4.3].*

### 5.1.4 Decoupled Innovations and Youla Augmentation

The source of the loss of contraction in the counterexample (Section 5.1.1) was the indirect interaction between the augmentation $\tilde{u}$ and the innovations $\tilde{y}$. If, however, $\tilde{y}$ is completely decoupled from $\tilde{u}$, then adding $\mathcal{Q}$ does not introduce a feedback loop in the closed-loop system, and we retain the closed-loop contracting and Lipschitz properties of the original base-controlled system. We have already seen this property in three special cases – LTI systems (Theorem 4.1-1), fully-observed nonlinear systems (Theorem 4.2-1), and partially-observed nonlinear systems with zero uncertainty (Theorem 5.1-1).

**Proposition 5.3.** *In each of the closed-loop systems from Theorems 4.1, 4.2 and 5.1-1, the innovations* $\tilde{y}$ *are the outputs of a contracting and Lipschitz system* $\mathcal{T} : (\eta, d) \mapsto \tilde{y}$ *given by*

$$\begin{aligned} \psi_{t+1} &= \tilde{F}(\psi_t, \eta_t, d_t) \\ \tilde{y}_t &= \tilde{H}(\psi_t, \eta_t, d_t) \end{aligned} \tag{5.10}$$

*with internal state* $\psi_t \in \mathbb{R}^{n_\psi}$ *and* $\tilde{F}, \tilde{G}$ *locally Lipschitz.*

The decoupling of $\tilde{y}$ and $\tilde{u}$ in this way is a sufficient condition for the Youla parametrisation to guarantee contracting and Lipschitz closed-loop responses. We therefore unify our three special cases with the following result.

**Theorem 5.4.** *Suppose the base-controlled system $\mathcal{G}_{\mathcal{K}_b} : (\eta, d, \tilde{u}) \mapsto z$ satisfies assumption **A1** and the innovations $\tilde{y}$ are generated by a contracting and Lipschitz system $\mathcal{T}$ (5.10). Then for any contracting and Lipschitz Youla parameter $\mathcal{Q}$ (4.6), the closed-loop system $z = \mathcal{G}_{\mathcal{K}_b}(\eta, d, \mathcal{Q}(\eta, \mathcal{T}(\eta, d)))$ mapping $(\eta, d) \mapsto z$ is also contracting and Lipschitz.*

### 5.1.5 Achieving Closed-Loop Contracting and Lipschitz Responses to Disturbances and Uncertainty

When the innovations are coupled with the Youla augmentation (e.g., due to disturbances or model parameter uncertainty), we can still achieve contracting and Lipschitz closed loops by imposing further restrictions on the Youla parameter. Our approach is based on IQC theory and loop shaping. We first group together the nonlinear plant $\mathcal{G}$, the base controller $\mathcal{K}_b$, and the observer $\mathcal{O}$ as the system $\mathcal{G}_\Delta : (\eta, d, \tilde{u}) \mapsto (z, \tilde{y})$ shown in Figure 5.2. We then take $\mathcal{Q} = \mathcal{W}_1 \circ \bar{\mathcal{Q}} \circ \mathcal{W}_2$ where $\mathcal{W}_1$ and $\mathcal{W}_2$ are fixed, stable linear filters, and $\bar{\mathcal{Q}}$ is a learnable contracting and Lipschitz system. $\mathcal{Q}$ is therefore also contracting and Lipschitz. The following proposition gives a sufficient condition for achieving a stable closed loop.

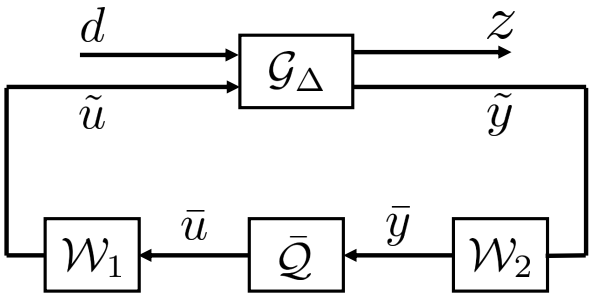



**Figure 5.2:** Youla parametrisation via loop shaping, where known inputs $\eta$ have been omitted for simplicity.

**Proposition 5.5.** *Suppose the system $\mathcal{G}_\Delta$ is contracting and there exist stable linear filters $\mathcal{W}_1 : \bar{u} \mapsto \tilde{u}$ and $\mathcal{W}_2 : \tilde{y} \mapsto \bar{y}$ with $\bar{u} \in \ell^{n_u}$ and $\bar{y} \in \ell^{n_y}$ such that the weighted system $\bar{\mathcal{G}}_\Delta : (\eta, d, \bar{u}) \mapsto (z, \bar{y})$ admits the incremental IQC defined by $(Q, S, R)$ with*

*$Q \prec 0$ and $R \succ 0$ partitioned as*

$$Q = \begin{bmatrix} Q_{zz} & Q_{z\bar{y}} \\ Q_{z\bar{y}}^\top & Q_{\bar{y}\bar{y}} \end{bmatrix}, S = \begin{bmatrix} S_{dz} & S_{d\bar{y}} \\ S_{\bar{u}z} & S_{\bar{u}\bar{y}} \end{bmatrix}, R = \begin{bmatrix} R_{dd} & R_{d\bar{u}} \\ R_{d\bar{u}}^\top & R_{\bar{u}\bar{u}} \end{bmatrix}.$$

*Take $\mathcal{Q} = \mathcal{W}_1 \circ \bar{\mathcal{Q}} \circ \mathcal{W}_2$, and suppose further that the system $\bar{\mathcal{Q}} : \bar{y} \mapsto \bar{u}$ is contracting and satisfies the incremental IQC defined by $(\bar{Q}, \bar{S}, \bar{R})$. Then the closed loop of $\mathcal{G}_\Delta$ and $\mathcal{Q}$ is contracting and Lipschitz if*

$$\begin{bmatrix} Q_{\bar{y}\bar{y}} + \bar{R} & S_{\bar{u}\bar{y}}^\top + \bar{S} & Q_{z\bar{y}} \\ S_{\bar{u}\bar{y}} + \bar{S}^\top & R_{\bar{u}\bar{u}} + \bar{Q} & S_{\bar{u}z} \\ Q_{z\bar{y}}^\top & S_{\bar{u}z}^\top & Q_{zz} \end{bmatrix} \prec 0. \tag{5.11}$$

**Remark 5.3.** *Using a pre-compensator $\mathcal{W}_1$ and/or a post-compensator $\mathcal{W}_2$, the system $\bar{\mathcal{G}}_\Delta$ has a desired open-loop shape, which can be understood as a dynamic version of input normalisation in deep RL [5]. This allows for learning high-performance controllers via simple IQCs for $\bar{\mathcal{G}}_\Delta$ and $\bar{\mathcal{Q}}$, e.g.,*

$$Q_{\tilde{y}\tilde{y}} = -1/\nu I,\ S_{\tilde{u}\tilde{y}} = 0,\ R_{\tilde{u}\tilde{u}} = \nu I,\ Q_{z\bar{y}} = 0,\ S_{\bar{u}z} = 0,$$
$$\bar{Q} = -1/\mu I,\ \bar{S} = 0,\ \bar{R} = \mu I.$$

*In this case, condition* (5.11) *is reduced to the requirement for the incremental small-gain theorem, i.e., $\mu\nu < 1$.*

## 5.2 Converse Results

In Section 4.4 we saw that, in certain simplified settings, the proposed Youla parametrisation includes all possible and only those controllers resulting in contracting and Lipschitz closed-loops. In Section 5.1 we saw by counterexample that this is not true in the partially-observed nonlinear case, but that the weaker property of d-tube contraction and Lipschitzness holds. This raises the question: what class of controllers *are* cov-

ered by the Youla parametrisation? We provide an answer in this section, and include proofs in Section 5.5.

We first introduce the following system which connects the observer (4.4) and base controller dynamics (4.3) in parallel, forming a map $(\eta, u, \tilde{y}) \mapsto (y, \hat{z})$:

$$\begin{aligned} \mathcal{O} : \hat{x}_{t+1} &= f_o(\hat{x}_t, \eta_t, u_t, \hat{y}_t + \tilde{y}_t), \quad \hat{y} = h(\hat{x}_t) \\ \mathcal{K}_b : s_{t+1} &= f_b(s_t, \eta_t, u_t, \hat{y}_t + \tilde{y}_t), \quad \hat{u} = k(s_t, \eta_t, \hat{y}_t + \tilde{y}_t) \\ y_t &= h(\hat{x}_t) + \tilde{y}_t \\ \hat{z}_t &= [\hat{x}_t; s_t; u_t] \end{aligned} \tag{5.12}$$

We then introduce an analogous assumption to **A1** for (5.12), which allows us to state Theorem 5.6.

**A1″** *Observer-stabilising base controller:* the closed-loop system $\tilde{\mathcal{G}}_{\mathcal{K}_b} : (\eta, \tilde{y}, \tilde{u}) \mapsto \hat{z}$ under the base controller, i.e., (5.12) in feedback with (4.3), is contracting and Lipschitz.

**Theorem 5.6.** *Suppose assumption **A4** holds, and consider the system* (5.12) *with controller* (4.7) *parametrised by* $\mathcal{Q}$ (4.6).

1. *Suppose assumption **A1″** also holds. Then for any contracting and Lipschitz* $\mathcal{Q}$*, the closed-loop system with* (5.12) *is contracting and the map* $(\eta, \tilde{y}) \mapsto \hat{z}$ *is Lipschitz.*

2. *Any controller* $u = \mathcal{K}(\eta, y)$ *of the form* (4.11) *achieving a contracting and Lipschitz closed loop with* (5.12) *can be written as* (4.7) *with a contracting and Lipschitz* $\mathcal{Q}$*.*

The interpretation is that for a given base controller and observer, the Youla controller (4.7) with contracting and Lipschitz $\mathcal{Q}$ parametrises all and only controllers $\mathcal{K}$ stabilising the system (5.12). The structure of this interconnection is shown in Figure 5.3. It

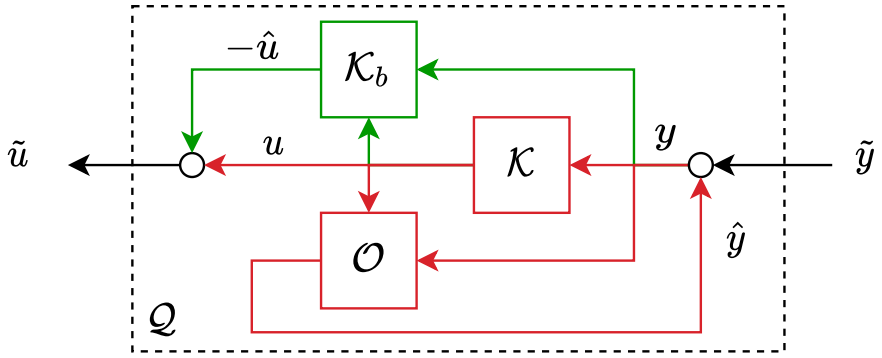



**Figure 5.3:** Block diagram showing how a Youla parameter $\tilde{u} = \mathcal{Q}(\tilde{y})$ can be constructed so that any controller $u = \mathcal{K}(y)$ can be realised in the Youla form (4.7). Known exogenous inputs $\eta$ have been omitted to simplify the diagram.

can be seen that there is a feedback interconnection of the controller $\mathcal{K}$ and the observer $\mathcal{O}$ (red), which is connected in series/parallel with the base controller $\mathcal{K}_b$ (green).

The proof (Section 5.5) is based on the idea that for a given base controller and observer, any controller $\mathcal{K}$ can be realised in the Youla form (4.7) with $\mathcal{Q}$ as the mapping $(\eta, \tilde{y}) \mapsto \tilde{u}$ shown in Figure 5.3. With this construction, the mapping $y \mapsto u$ of the controller $\mathcal{K}$ remains unchanged. Stability of the resulting map $(\eta, \tilde{y}) \mapsto \hat{z}$ is essentially equivalent to stability of each of the red and green components, since $\hat{z}$ includes the states of both, and contraction and Lipschitzness are preserved through series/parallel system connections. We take a deeper look into the meaning of stabilising the red and green systems below.

**(1) Red system:** stabilising the red system corresponds to the controller $\mathcal{K}$ stabilising the observer. This is related to stabilising the plant (4.1), but not the same when disturbances are present. When $\tilde{y} = 0$, observer correctness (**A3**) implies that $f_o(x_t, \eta_t, u_t, h(x_t)) = f(x_t, \eta_t, u_t)$, which is exactly the disturbance-free plant dynamics. The controller $\mathcal{K}$ stabilises the observer against the particular disturbance structure

$$\hat{x}_{t+1} = f_o(\hat{x}_t, \eta_t, u_t, h(x_t) + w_t)$$

entering via the measurement channel, rather than an additive disturbance as in (4.1).

**(2) Green system:** stability of the green system is just stability of the base controller. At first, this may look like a requirement of strong stabilisation, which is known to be restrictive. However, this is not the case since $\mathcal{K}_b$ takes the true control signal $u$ as an input. For example, in the LTI case from Section 4.4.1, this is the requirement that $(A - LC)$ is stable, which will generally be the case, whereas strong stabilisation requires that $(A - LC - BK)$ be stable, which is restrictive. The difference is that we use the total control signal $u$ in place of $\hat{u} = -K\hat{x}$.

**Remark 5.4.** *Stabilising the red system in Figure 5.3 can be interpreted as defining a stable image representation of the controller $\mathcal{K}$ (i.e., a right-coprime factor), since it represents a mapping from $\tilde{y} \mapsto (\mathcal{K}(y), y)$. Similarly, one can think of the original observer* (4.4) *together with the innovations map* (4.5) *as a stable kernel representation (i.e., a left-coprime factor) mapping $(u, y) \mapsto \tilde{y}$ (where we have omitted $\eta$ for illustrative purposes). The perspective of constructing the Youla parametrisation via stable image and kernel representations is explored in detail in [48, 85].*

In certain special cases, the requirement that a controller $\mathcal{K}$ stabilises the system (5.12) follows naturally from $\mathcal{K}$ stabilising the plant (4.1). These special cases arise when we make stronger assumptions on the structure of the plant, base controller, and observer. In the following corollaries, we show that the Youla parametrisation contains all contracting and Lipschitz closed-loops for two particular system structures. Note that Theorem 4.1-2 for LTI systems follows directly from Corollary 5.8.

**Corollary 5.7.** *Theorem 4.2-2 follows directly from Theorem 5.6 with measurements $y_t = x_t$, observer $\hat{x}_{t+1} = f(y_t, \eta_t, u_t)$, and a static base controller $\hat{u}_t = k(\eta_t, y_t)$.*

**Corollary 5.8.** *Suppose assumption **A4** holds and the base controller* (4.3) *and ob-*

*server* (4.4) *are in the certainty-equivalence form*

$$\begin{aligned} \hat{x}_{t+1} &= f(\hat{x}_t, \eta_t, u_t) + L(\eta_t, y_t - h(\hat{x}_t)) \\ \hat{u}_t &= k(\hat{x}_t, \eta_t, y_t), \end{aligned} \tag{5.13}$$

*where* $L : \mathbb{R}^{n_\eta} \times \mathbb{R}^{n_y} \to \mathbb{R}^{n_x}$ *is Lipschitz. Then any controller* $u = \mathcal{K}(\eta, y)$ *of the form* (4.11) *achieving a contracting and Lipschitz closed loop* $(\eta, d) \mapsto z$ *with* (4.1) *can be written in the form* (4.7) *with a contracting and Lipschitz* $\mathcal{Q}$ (4.6).

**Remark 5.5.** *To understand why Corollary 5.8 cannot be extended to a necessary and sufficient condition parametrising all and only stabilising controllers* $\mathcal{K}$*, we examine the difference between stabilising the system* (5.12) *and stabilising the plant* (4.1)*. Consider Example 5.1 again. In this case, the observer* (5.2) *is just an open-loop simulation of the disturbance-free plant* (5.1)*. It does not depend on* $y$ *(or* $\tilde{y}$*) at all, hence adding a disturbance to the "measurement channel" of the observer has no effect (i.e., the Lipschitz constant of* $f_o$ *w.r.t.* $y$ *is zero). Therefore, a controller* $\mathcal{K}$ *stabilising this observer only stabilises the plant in a disturbance-free setting, which implies nothing about its ability to stabilise the plant in the presence of additive disturbances* $w$*.*

Figure 5.4 illustrates the connection between Theorems 5.1, 5.4 and 5.6 for stabilising controllers satisfying the conditions in Theorem 5.6. The Youla parametrisation (4.7) contains all policies (4.11) that achieve contracting and Lipschitz closed loops with the plant (set C). When the innovations and control inputs are coupled due to additive disturbances, the closed-loop system does not necessarily achieve these same strong stability properties. Instead, it guarantees d-tube contraction and Lipschitzness, which is a slightly weaker stability result (set A). In the special case where the innovations are decoupled from the control signal as per (5.10), all three sets coincide (e.g., LTI systems, full state-feedback).

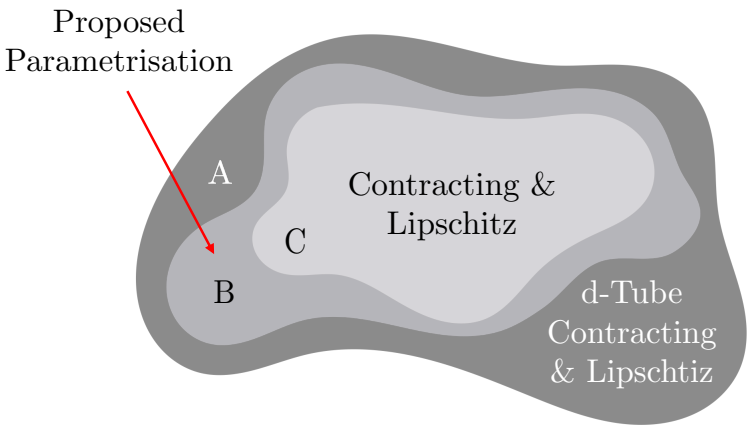



**Figure 5.4:** Illustration of stabilising policy sets for controllers satisfying the conditions in Theorem 5.6: A) controllers achieving d-tube contracting and Lipschitz closed-loops, B) the proposed nonlinear Youla parametrisation, and C) controllers achieving contracting and Lipschitz closed loops. The three sets coincide if the innovations also satisfy (5.10).

## 5.3 Deep RL with Youla-REN

Having established the stability properties of the nonlinear Youla parametrisation (4.7), we now investigate its use in learning-based control. In this section, we first introduce our proposed Youla-REN policy class (Section 5.3.1), where we use a REN [136] as the contracting and Lipschitz Youla parameter $\mathcal{Q}$. We then illustrate the advantages of its built-in stability certificates on three deep RL tasks with: (i) general reward functions without stability consideration; (ii) short training horizons; and (iii) systems with model uncertainty (Sections 5.3.2 to 5.3.4, respectively). All experiments[1] were performed in Julia using the `RobustNeuralNetworks.jl` package (Appendix B) and the APG RL algorithm [100] (see Section 2.4.1).

### 5.3.1 The Youla-REN Policy Class

We parametrise the Youla parameter $\mathcal{Q}$ via RENs [136], which were introduced in detail in Section 2.3.3. Together with the Youla control architecture (4.7), the resulting policy class is called Youla-REN. The motivations for choosing a REN as the Youla parameter are as follows.

[1] `https://github.com/nic-barbara/ReactToSurprises`

1. RENs admit built-in behavioural guarantees such as contraction, Lipschitzness, or other properties described by incremental IQCs. Thus, they are compatible with the analysis framework for our policy parametrisation. In the experiments below, we use $\gamma$REN to denote a REN with a prescribed Lipschitz bound of $\gamma \in \mathbb{R}^+$.

2. RENs are highly flexible and include many existing models as special cases, including MLPs, RNNs, and stable linear dynamical systems (Section 2.3.3). It was proven in [178, Prop. 2] that contracting RENs are also universal approximators for contracting and Lipschitz dynamical systems.

3. RENs are compatible with standard machine-learning tools as they permit a direct (smooth, unconstrained) parametrisation such that for any $\theta \in \mathbb{R}^N$, the resulting REN $\mathcal{Q}_\theta$ (2.20) is contracting and Lipschitz. This enables training of large-scale models via automatic differentiation and variants of SGD.

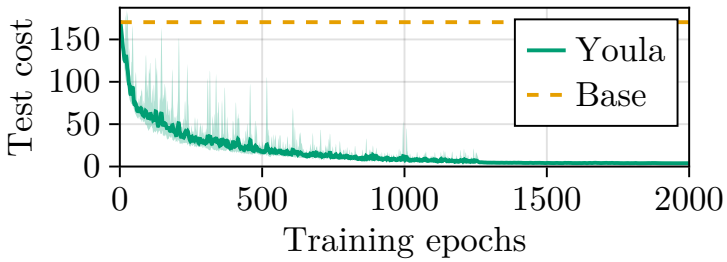
### 5.3.2 Nonlinear System with Economic Cost

In our first example, we learn stabilising policies for partially-observed nonlinear system while optimising an economic cost that does not naturally encourage stability. Consider the following academic example, adapted from [4]:

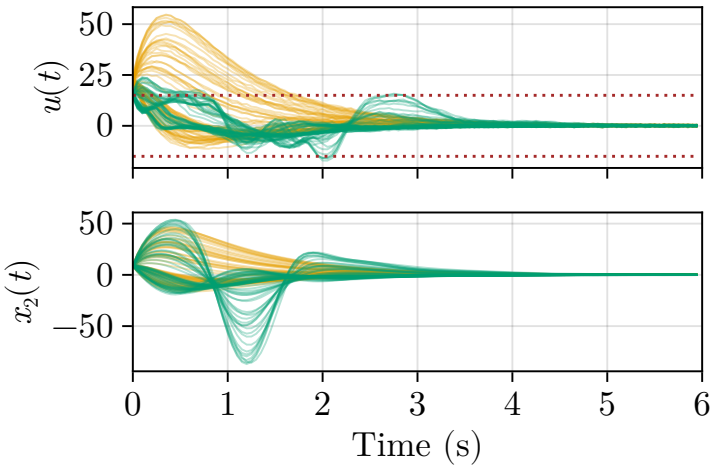
$$\begin{aligned} \dot{x}_1 &= -x_1 + x_3 + w_1 \\ \dot{x}_2 &= x_1^2 - x_2 - 2x_1x_3 + x_3 + w_2 \\ \dot{x}_3 &= -x_2 + u + w_3 \\ y &= [x_2; x_3] + v. \end{aligned} \tag{5.14}$$

We chose a stabilising base controller $u = y_1 - ky_2$ with gain $k = 1.5$, and an observer of the form

$$\begin{aligned} \dot{\hat{x}}_1 &= -\hat{x}_1 + \hat{x}_3 \\ \dot{\hat{x}}_2 &= \hat{x}_1^2 - \hat{x}_2 - 2\hat{x}_1\hat{x}_3 + \hat{x}_3 \\ \dot{\hat{x}}_3 &= -\hat{x}_2 + u + (y_2 - \hat{x}_3) - (y_1 - \hat{x}_2). \end{aligned} \tag{5.15}$$




**(a)** Test cost during training.




**(b)** Trajectory rollouts of $u$ and $x_2$ for a trained Youla-REN.

**Figure 5.5:** Youla-REN learns to improve economic performance while maintaining closed-loop stability. Bands in (a) show the range across 10 random seeds. Dotted lines in (b) indicate the soft constraints $\pm u_{\max}$ imposed on the controls.

It can be shown that assumptions **A1**–**A4** hold for the this base controller and observer via [182]. We set $w, v$ as zero-mean Gaussian noise with variances $\sigma_{w_i} = 10^{-2}, \sigma_v = 10^{-3}$ respectively, and discretised the system using the forward-Euler approximation and a time-step of $\Delta t = 0.05\,\mathrm{s}$.

Referring to the general cost structure in (2.2), we chose an economic stage cost

$$g(x_t, u_t) := |u_t| + 500 \max(|u_t| - u_{\max}, 0), \tag{5.16}$$

which is the $\ell_1$-norm of the control inputs plus a strong penalty for exceeding the soft threshold $u_{\max} = 15$. Note that if stability is ignored, then the optimal policy is simply $u \equiv 0$, which results in an unstable closed loop.

Figure 5.5 shows that with the proposed Youla-REN, we are able to learn a stabilising controller which achieves significant performance improvements compared to the base controller while maintaining closed-loop stability. The learned Youla parameter $\mathcal{Q}$ adds corrections to the base controller signal to keep the total $u$ within the soft bounds $\pm u_{\max}$ most of the time. It only allows small deviations outside the bounds for short intervals, and encourages $u$ to be close to zero within these bounds. Moreover, the trajectories of both $u$ and $x_2$ indicate that the Youla-REN preserves the closed-loop stability of the base controller even though such behaviour was not explicitly encoded in the cost function. We therefore avoid the laborious process of reward-shaping to encourage stability in this task, which is commonly required for deep RL policies to be both stabilising and performant (e.g., [138]).

### 5.3.3 Long-Term Stability with Short-Term Training

Our second example illustrates that our proposed Youla-REN policies can achieve good long-term test performance even when they are trained over short time horizons. We consider the following well-known LQG example from [40],

$$\begin{aligned} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u + \begin{bmatrix} 1 \\ 1 \end{bmatrix} w \\ y &= x_1 + v, \end{aligned} \tag{5.17}$$

where $w, v \in \mathbb{R}$ are Gaussian noises with variances $\sigma_w > 0$ and $\sigma_v = 1$, respectively. The control objective is

$$J = \mathbb{E}_{x_0,w,v} \int_{t=0}^{\infty} \left[ q^2 (x_1 + x_2)^2 + u^2 \right] \, dt \tag{5.18}$$

with $q > 0$. The optimal LQG controller has arbitrarily small-gain and phase margins as $\sigma_w, q$ increase [40]. This makes the problem challenging in a deep RL setting, since optimal and unstable policies may appear very close in parameter space.

We discretised the system with the forward-Euler method ($\Delta t = 0.01\,\mathrm{s}$) and chose a

training horizon of 100 s. For the base controller, we chose a de-tuned LQG controller designed to minimise (5.18) with $q = 10^3$ and $\sigma_w = 10$, while for training and test evaluations we used $q, \sigma_w = 10^3$.

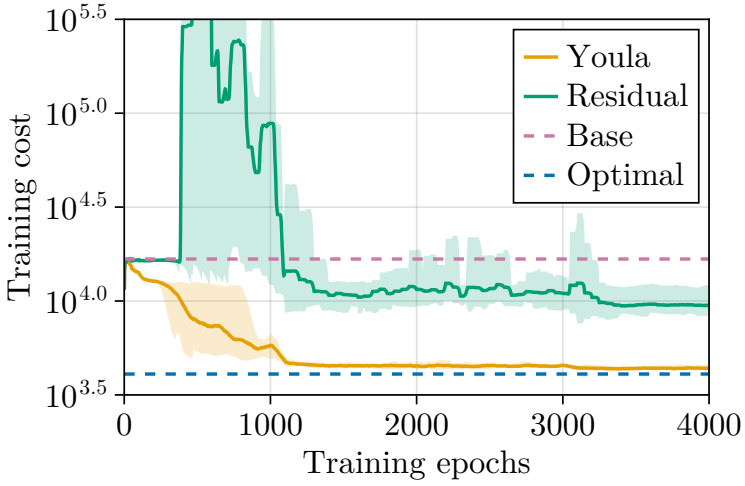
For comparison, we also trained *Residual-REN* policies

$$u = \mathcal{K}_b(y) + \mathcal{Q}(y), \tag{5.19}$$

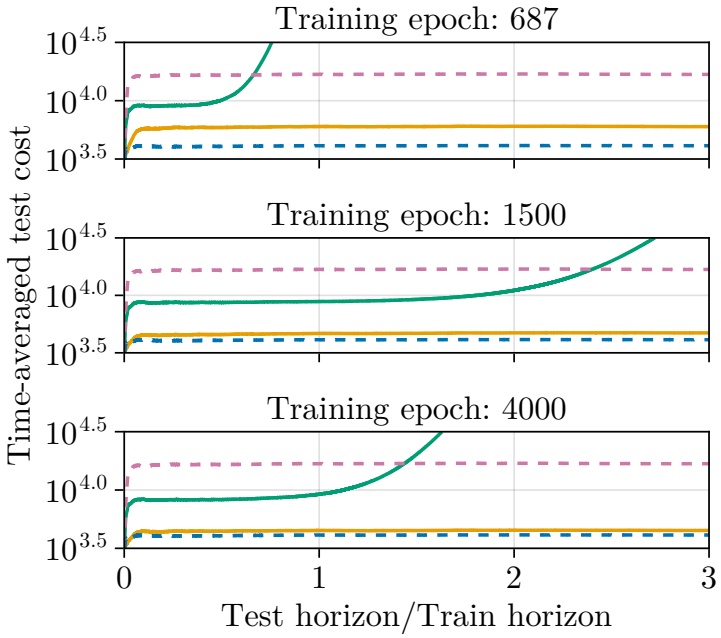
where $\mathcal{K}_b$ is the stabilising base controller and the learnable component $\mathcal{Q}$ is parametrised by a REN. As shown in Figure 4.1, the main difference from Youla-REN is that the system $\mathcal{Q}$ in a Residual-REN acts directly on measurements $y$ rather than the innovations $\tilde{y}$. A direct consequence is that Youla-REN guarantees closed-loop stability (provided that $\mathcal{Q}$ is contracting and Lipschitz) while Residual-REN does not.

The training curves in Figure 5.6a show that the Youla-REN policies consistently outperform the Residual-RENs and smoothly converge to a near-optimal final cost. In contrast, the Residual-RENs never approach the optimal LQG cost and yield large variance during training. We found that the Residual-RENs were highly sensitive to the learning rate, with slightly smaller values resulting in almost no improvement over the base controller, while larger values lead to even greater fluctuations and a worse final cost. Since the true optimal policy is near the boundary of the stabilising controller set, it is common to encounter unstable Residual-REN policies during training (e.g., at epoch 687 in the top panel of Figure 5.6b). This is not possible with the Youla-REN due to its built-in stability guarantees.

We also compare the long-term test performance of each policy architecture in Figure 5.6b. Testing Residual-RENs at epochs 1500 and 4000, the policies appears to stabilise the closed-loop system *over the training horizon* (i.e., the cost is less than the base cost). However, rolling out these policies over longer horizons reveals unstable responses. The Youla-REN demonstrates consistently strong performance over long-term horizons at all stages during and after training.




**(a)** Mean training cost based on short-horizon rollouts.




**(b)** Mean test cost for long-term policy rollouts.

**Figure 5.6:** Youla-REN achieves long-term stability even with short-horizon training on the classic LQG problem from [40]. Bands in (a) show the range across 10 random seeds.

### 5.3.4 Robust Stability with Model Uncertainty

Our final example demonstrates that Youla-REN policies can achieve good performance while maintaining closed-loop stability in the presence of model parameter uncertainty. Consider the linearised cart-pole system

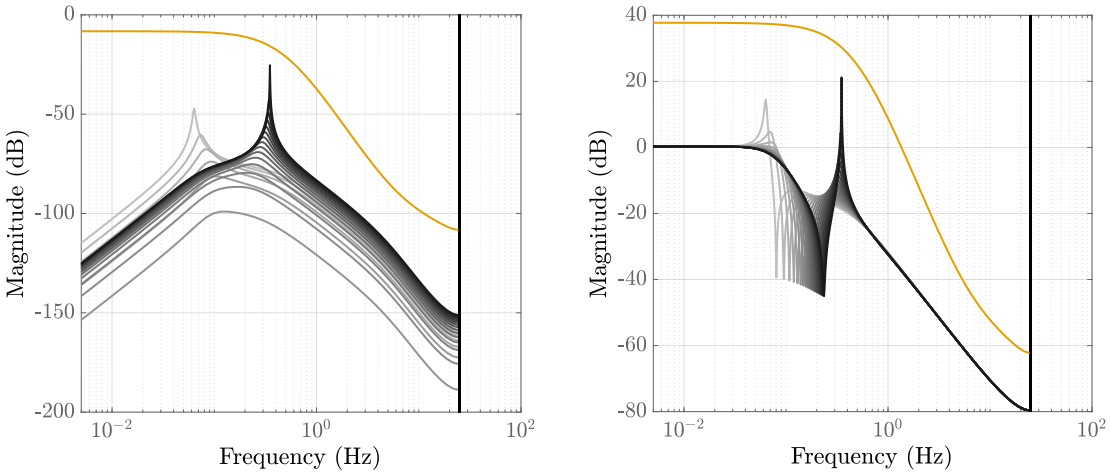
$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{m_p g}{m_c} 0 & \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{(m_c+m_p)g}{m_c l} 0 & \end{bmatrix} x + \begin{bmatrix} 0 \\ \frac{1}{m_c} \\ 0 \\ -\frac{1}{m_c} \end{bmatrix} u + w \tag{5.20}$$
$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} x + v$$

where $w, v$ are zero-mean Gaussian noise with variances $\sigma_w = 10^{-3}$, $\sigma_v = 10^{-4}$, respectively. We discretised (5.20) with the forward-Euler method ($\Delta t = 0.02\,\mathrm{s}$) and chose $g = 9.81$, $l = 10$, $m_c = 1$ and $m_p \in [0.14, 0.35]$. The control objective was to minimise an infinite-horizon quadratic cost with weights $Q = \texttt{diag}(1, 0, 1, 0)$ and $R = I$. We chose a discrete-time, time-invariant LQG controller as the base controller, designed at a nominal mass $m_p^\star = 0.2$. We computed its control gain with the same $Q, R$ as the cost function, but the steady-state Kalman filter was designed with covariances $\Sigma_w = \texttt{diag}(1, 10^3, 1, 10^3) \times \Delta t$ and $\Sigma_v = 10^{-3} I / \Delta t$ for the process and measurement noises, respectively, to ensure closed-loop stability across the whole uncertain $m_p$ range.

We parametrised robustly-stabilising policies for this system with Youla-$\gamma$REN, a Lipschitz-bounded variant of Youla-REN. We chose the Lipschitz bound $\gamma$ using a small-gain approach with loop shaping to enable both closed-loop stability and strong performance (Proposition 5.5). For frequency weightings, we chose the output filter to be $\mathcal{W}_1(s) = 1$ and the input filter

$$\mathcal{W}_2(s) = \frac{(s+3)^4}{\nu(s+50)^4} \tag{5.21}$$

with $\nu$ given in Table 5.1. The inverse of $\mathcal{W}_2(s)$ (Figure 5.7a) serves as an upper bound for the base-controlled system with uncertain pole mass. We then chose $\gamma$ as the inverse of the $\mathcal{H}_\infty$ norm of the base-controlled system $\tilde{u} \mapsto \tilde{y}$ with filtering. Based on an empirical study, we observed only minor differences in the results when swapping $\mathcal{W}_1$ and $\mathcal{W}_2$. Note that the shape of $\mathcal{W}_2(s)$ allows for higher gains at moderately high frequencies, which can help to learn a policy that rapidly corrects for displacements




**(a)** $\tilde{u} \mapsto \tilde{y}$ **(b)** $\tilde{u} \mapsto y$

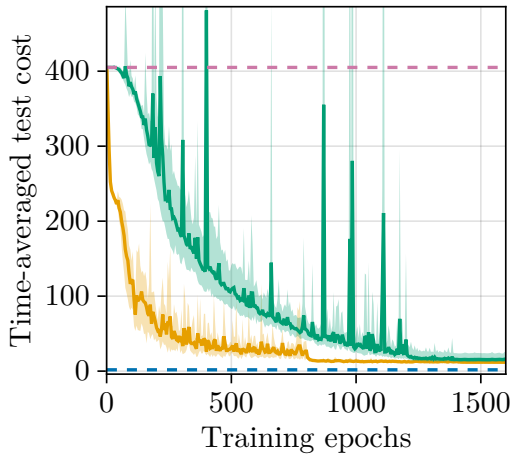
**Figure 5.7:** Bode magnitude plots for system (5.20) under the base LQG controller with different pole masses $m_p$, where the color gradient darkens as $m_p$ increases. The yellow lines show the inverse of the (discretised) weighting filters (5.21).

of the pole angle. We did not shape $\mathcal{W}_2$ at low frequencies, since low-frequency poles require training over very long horizons to capture their behaviour.

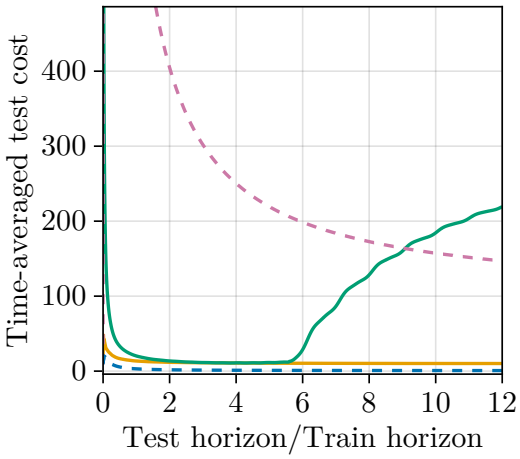
For comparison, we also trained Residual-LSTM policies – the residual RL architecture (5.19) with $\mathcal{Q}$ parametrised by a LSTM [67]. LSTMs are popular recurrent networks known to perform well on RL tasks [10]. We found that Residual-LSTMs were easier to train for this problem than unconstrained Residual-RENs.

### Results and Discussion

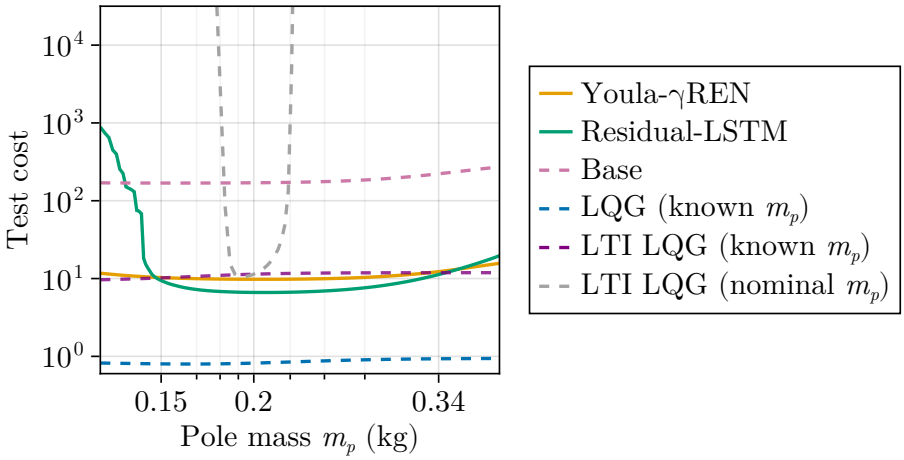
We first compare the Youla-$\gamma$REN and Residual-LSTM in Figures 5.8a and 5.8b to investigate the effect of stability guarantees on policy training and long-term test roll-outs with model uncertainty. We observe similar qualitative results to Figure 5.6. Specifically, the Youla-$\gamma$RENs quickly and smoothly converge to a near-optimal cost during training, and yield stable closed-loop responses over much longer test horizons than the training horizon. In contrast, the Residual-LSTMs often converge to unstable policies during training, causing spikes in their cost curves, and also exhibit closed-loop instability at test time when rolled out over long horizons (e.g., $6\times$ the training horizon in Figure 5.8b).




**(a)** Training curves.




**(b)** Cost rollouts.




**(c)** Sensitivity to model uncertainty.

**Figure 5.8:** Mean test cost for the cart-pole problem with uncertain pole mass: (a) short-horizon cost during training (2× training horizon); (b) cost for long-term rollouts of particular policies; and (c) long-horizon cost as a function of the uncertain pole mass (8× training horizon). LQG refers to the optimal time-varying implementation while LTI LQG refers to the combination of a steady-state Kalman filter and a Linear Quadratic Regulator (LQR).

The closed-loop responses under different pole masses (the nominal $m_p^\star$ and two end points) are depicted in Figure 5.9. Both the Youla-$\gamma$REN and Residual-LSTM policies can stabilise the system with a nominal and large pole mass. However, when the pole mass is small, the Residual-LSTM exhibits marginally unstable behaviour. That is, the policy rollout of Residual-LSTM is close to that of Youla-$\gamma$REN within the training horizon, but quickly diverges as the time increases. This highlights the im-

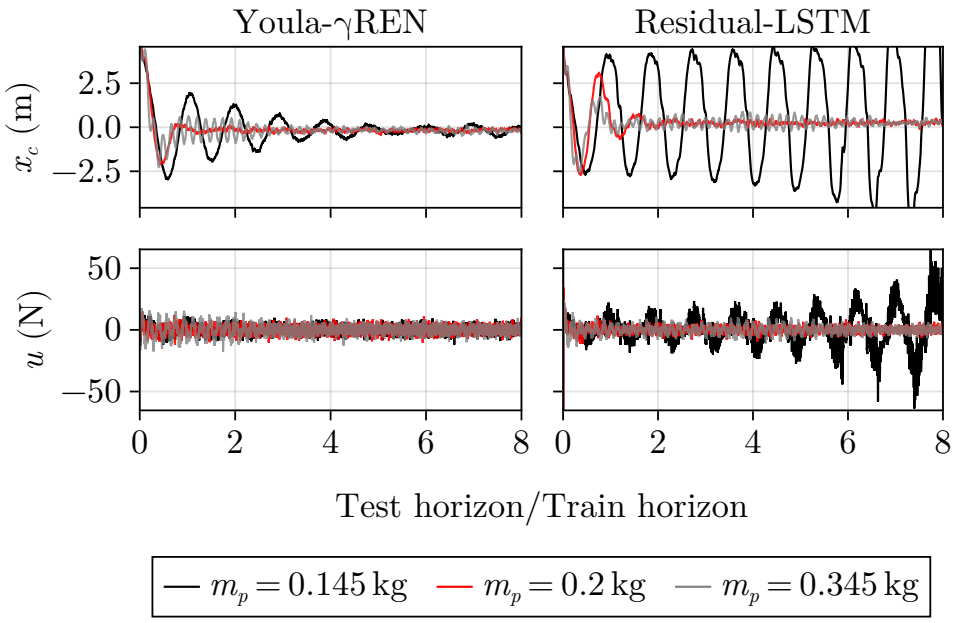



**Figure 5.9:** Policy rollouts on (5.20) with the same initial state and different pole mass $m_p$, where $x_c$ is the cart position.

portance of model-based stability guarantees in deep RL to achieve reliable long-term test performance.

The sensitivity of the learned policies to variations in the uncertain parameter is shown further in Figure 5.8c. The Residual-LSTM performs well over most of the range, but exhibits instability at small pole masses. The proposed Youla-$\gamma$REN ensures stability and achieves near optimal performance across the whole parameter range. Although its cost is greater than the (optimal) time-varying LQG with known mass, it is very close the time-invariant LQG with known mass, which is reasonable as our policy is also time-invariant. This performance cannot simply be achieved by an LQG controller designed at the nominal mass (shown in gray in Figure 5.8c) or a Youla-$\gamma$REN controller with linear $\mathcal{Q}$ (see Figure 5.10). This suggests that the proposed nonlinear Youla-REN controller has the ability to learn to adapt to different model parameters.

### Ablation Study

We now conduct an ablation study on the Youla-$\gamma$REN policy class. We trained policies with the three architectures in Table 5.2, all of which have built-in stability guarantees

**Table 5.1:** Policy hyperparameters.

| **Policy Architecture** | Param. $\nu$ in $\mathcal{W}_2$ | Lip. bound $\gamma$ of $\mathcal{Q}$ |
|---|---|---|
| Youla-$\gamma$REN | $5 \times 10^{-4}$ | 1.7 |
| Youla-$\gamma$REN (linear) | $5 \times 10^{-4}$ | 1.7 |
| Youla-$\gamma$REN (no filter) | - | 120 |
| Residual-$\gamma$REN | $10^{-2}$ | 0.15 |
| Residual-LSTM | - | - |

via proper choice of the hyperparameters in Table 5.1.

Figure 5.10 shows that each of the three main components of the Youla-$\gamma$REN (the Youla parametrisation, nonlinear $\mathcal{Q}$, and frequency-weighted filter) is crucial to learn high-performing policies. The linear Youla-$\gamma$REN plateaus at a much higher cost than its nonlinear counterparts due to the poor expressivity of a linear $\mathcal{Q}$. When the weighting filter is removed, the gain of the Youla-$\gamma$REN is restricted equally across all frequencies, leading to a higher cost ceiling. Perhaps most interestingly, there is an extreme drop in performance when switching from the Youla configuration to residual RL. The main reason is that the gain of $\tilde{u} \mapsto \tilde{y}$ is much smaller than $\tilde{u} \mapsto y$ (Figure 5.7), allowing for a more expressive $\mathcal{Q}$ (i.e., larger Lipschitz bound) when ensuring closed-loop stability via the small-gain theorem. This reinforces the fact that the policy architecture does matter in deep RL [137]. We learn performant policies with built-in robustness to model uncertainty by combining all three key elements of the Youla-$\gamma$REN policy architecture.

**Table 5.2:** Policy architectures for the ablation study (Figure 5.10).

| Policy | Architecture | $\mathcal{Q}$ | Weight filter |
|---|---|---|---|
| Youla-$\gamma$REN | Youla | Nonlinear | Yes |
| Youla-$\gamma$REN (linear) | Youla | Linear | Yes |
| Youla-$\gamma$REN (no filter) | Youla | Nonlinear | No |
| Residual-$\gamma$REN | Residual | Nonlinear | Yes |

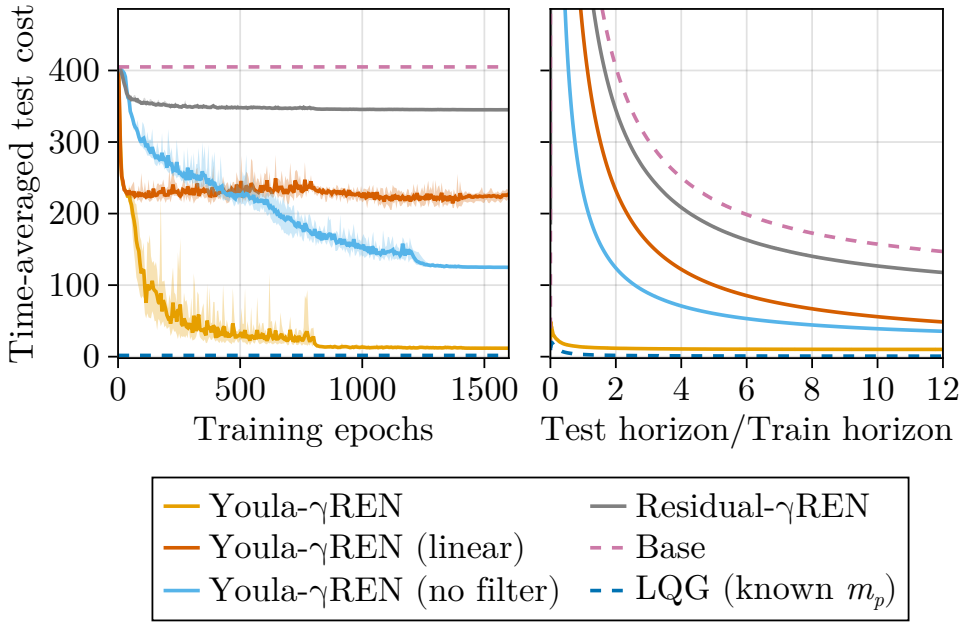



**Figure 5.10:** Ablation study on the Youla-γREN.

## 5.4 Conclusions

Chapters 4 and 5 have studied a nonlinear version of the Youla parametrisation for nonlinear dynamical systems, and presented a constructive neural realisation called the Youla-REN. We have proven that: the parametrisation guarantees contracting and Lipschitz closed loops when the innovations signal is decoupled from the Youla augmentation; if these signals are coupled due to additive disturbances, the disturbed trajectories are constrained within bounded tubes centred on the disturbed-free trajectories; and we cover *all* contracting and Lipschitz closed loops for certain nonlinear systems. Using RENs for the Youla parameter, empirical studies show that we can learn stabilising and performant Youla-REN policies on feedback control tasks, even if they are trained with arbitrary (potentially de-stabilising) cost functions, over short time horizons, and with model uncertainty.

## 5.5 Proofs

### Proof of Theorem 5.1

*Claim 1.* The result follows by combining Proposition 5.3 with Theorem 5.4. □

*Claim 2.* This first part of the proof (up until (4.20)) is identical to that of Theorem 4.3, but we repeat it here to make it self-contained. We first show that the bound of the innovations signal $\tilde{y}$ (4.5) is independent of the Youla parameter output $\tilde{u}$, despite there being a feedback loop between $\tilde{u}$ and $\tilde{y}$. Applying observer correctness (**A3**) to the nonlinear system $\mathcal{G}$, we have

$$\begin{aligned} x_{t+1} &= f(x_t, \eta_t, u_t) + w_t \\ &= f_o(x_t, \eta_t, u_t, h(x_t)) + w_t \\ &= f_o(x_t, \eta_t, u_t, y_t - v_t) + w_t \end{aligned} \tag{4.18}$$

whose trajectory is denoted by $(x, \eta, u, y, w, v)$. The observer dynamics are

$$\hat{x}_{t+1} = f_o(\hat{x}_t, \eta_t, u_t, y_t) \tag{4.4}$$

with $(\hat{x}, \eta, u, y)$ as its trajectory. This shares the same inputs $(\eta, u, y)$ as (4.18). Since the observer is contracting and Lipschitz (**A2**), from Lemma 2.1 we have

$$\|\tilde{x}\|_T = \|x - \hat{x}\|_T \leq \tilde{\gamma}(\|v\|_T + \|w\|_T) + \tilde{\kappa}(x_0, \hat{x}_0), \tag{5.22}$$

where $\tilde{\gamma} \in \mathbb{R}^+$ and $\tilde{\kappa}(x_0, \hat{x}_0) \geq 0$ with $\tilde{\kappa}(x_0, x_0) = 0 \; \forall \, x_0 \in \mathbb{R}^{n_x}$. Since the output map $h$ is Lipschitz (**A4**), we obtain

$$\begin{aligned} \|\tilde{y}\|_T &= \|h(x) - h(\hat{x}) + v\|_T \leq \gamma_h \|\tilde{x}\|_T + \|v\|_T \\ &\leq \gamma_0(\|w\|_T + \|v\|_T) + \kappa_0(x_0, \hat{x}_0) \end{aligned} \tag{4.20}$$

where $\gamma_0 = \tilde{\gamma}\gamma_h + 1$ and $\kappa_0 = \gamma_h \tilde{\kappa}$ with $\gamma_h$ as the Lipschitz constant of $h$.

We now show that the effect of disturbances in constrained within a tube of bounded $\ell_2$ norm about the disturbance-free trajectories. Let $\xi_t = [x_t; s_t] \in \mathbb{R}^{n_x+n_s}$ and $q_t \in \mathbb{R}^{n_q}$ be the states of the base-controlled closed-loop system $(\eta, d, \tilde{u}) \mapsto z$ and the Youla parameter $(\eta, \tilde{y}) \mapsto \tilde{u}$, respectively. For convenience we denote the state of the total closed-loop system by $\bar{x}_t = [\xi_t; \hat{x}_t; q_t]$. Consider two copies of the system starting at initial states $\bar{x}_0, \bar{x}_0^\star$ with inputs $(\eta, w, v) \in \ell$ and $(\eta^\star, 0, 0) \in \ell$, respectively, where $x_0^\star = \hat{x}_0^\star$. From assumption (**A1**) and Lipschitzness of the Youla parameter respectively, we have

$$\begin{aligned}
\|\Delta z\|_T &\leq \gamma_1(\|\Delta\tilde{u}\|_T + \|\Delta\eta\|_T + \|w\|_T + \|v\|_T) + \kappa_1(\xi_0, \xi_0^\star), \\
\|\Delta\tilde{u}\|_T &\leq \gamma_2(\|\Delta\tilde{y}\|_T + \|\Delta\eta\|_T) + \kappa_2(q_0, q_0^\star),
\end{aligned}$$

for some $\gamma_1, \gamma_2 \in \mathbb{R}^+$ and $\kappa_1(\xi_0, \xi_0^\star), \kappa_2(q_0, q_0^\star) \geq 0$ with $\kappa_i(a, a) = 0$ for $i = 1, 2$, where $\Delta z = z - z^\star$ and similar for all other variables. Combining these inequalities with (4.20) gives

$$\|\Delta z\|_T \leq \gamma(\|\Delta\eta\|_T + \|w\|_T + \|v\|_T) + \kappa(\bar{x}_0, \bar{x}_0^\star) \tag{5.23}$$

with $\gamma = \gamma_1\,(1 + \gamma_2 \max(\gamma_0, 1))$ and $\kappa = \gamma_1\gamma_2\kappa_0 + \kappa_1 + \gamma_1\kappa_2$. Noting $x_0^\star = \hat{x}_0^\star$ implies $\kappa(\bar{x}_0^\star, \bar{x}_0^\star) = 0$ completes the proof. □

## Proof of Theorem 5.2

*Claim 1.* We are interested in the case where $d \equiv 0$ but $x_0 \neq \hat{x}_0$. We first show that the observer error exponentially decays when $d \equiv 0$ regardless of $\tilde{u}$. By setting $w_t = 0$ and $v_t = 0$ in (4.18), the plant dynamics become a copy of the system observer (4.4) with identical inputs but a different state:

$$x_{t+1} = f_o(x_t, \eta_t, u_t, y_t). \tag{5.24}$$

The observer is contracting by assumption (**A2**), hence there exist $\beta_o \in \mathbb{R}^+$ and $\alpha_o \in [0, 1)$ such that $|\tilde{x}_t| \leq \beta_o\alpha^t|\tilde{x}_0|$, and therefore $\|\tilde{x}\|_T \leq \beta_o/\sqrt{1-\alpha_o^2}|\tilde{x}_0|$.

We consider two closed-loop trajectories $(\bar{x}^a, \eta)$ and $(\bar{x}^b, \eta)$ with the same input $\eta$ but different initial states $\bar{x}_0^a, \bar{x}_0^b$. The innovations difference $\Delta\tilde{y} = \tilde{y}^a - \tilde{y}^b$ decays exponentially as

$$|\Delta\tilde{y}_t| \leq |\tilde{y}_t^a| + |\tilde{y}_t^b| \leq \gamma_h(|\tilde{x}_t^a| + |\tilde{x}_t^b|) \leq \gamma_h\beta_o\alpha_o^t(|\tilde{x}_0^a| + |\tilde{x}_0^b|),$$

where $\gamma_h$ is the Lipschitz constant of the output map $h$. We now estimate the bound of $\Delta\tilde{u}$ via the Youla parameter $\mathcal{Q}$ (4.6). Note that the trajectories $(q^a, \eta, \tilde{y}^a)$ and $(q^b, \eta, \tilde{y}^b)$ satisfy

$$\begin{aligned} q_{t+1}^a &= f_q(q_t^a, \eta_t, \tilde{y}_t^b) + \Delta f_{q_t} \\ q_{t+1}^b &= f_q(q_t^b, \eta_t, \tilde{y}_t^b) \end{aligned}$$

where $\Delta f_{q_t} := f_q(q_t^a, \eta_t, \tilde{y}_t^a) - f_q(q_t^a, \eta_t, \tilde{y}_t^b)$, and $|\Delta f_{q_t}| \leq \gamma_{f_q}|\Delta\tilde{y}_t|$, since $f_q$ has a finite Lipschitz constant $\gamma_{f_q}$ *w.r.t.* $\tilde{y}$ by assumption. Therefore, $\Delta f_{q_t}$ decays exponentially, and so by Lemma A.1 and the assumption that $\mathcal{Q}$ is contracting with rate $\alpha_q \in [0, 1)$ and overshoot $\beta_q \in \mathbb{R}^+$, we have

$$\begin{aligned} |\Delta q_t| &\leq \beta_q\rho_1^t\left(|\Delta q_0| + C\gamma_{f_q}\gamma_h\beta_o(|\tilde{x}_0^a| + \left|\tilde{x}_0^b\right|)\right) \\ &\leq A_1\rho_1^t\left(|\Delta q_0| + |\tilde{x}_0^a| + \left|\tilde{x}_0^b\right|\right) \end{aligned} \tag{5.25}$$

where $\max(\alpha_o, \alpha_q) \leq \rho_1 < 1$, $C \in \mathbb{R}^+$ depending on $\alpha_o, \alpha_q$, and $A_1 = \beta_q \max(1, C\gamma_{f_q}\gamma_h\beta_o)$. Since the output mapping $h_q$ of $\mathcal{Q}$ is also Lipschitz, we can further conclude that

$$|\Delta\tilde{u}_t| \leq \gamma_{h_q}(|\Delta q_t| + |\tilde{y}_t|) \leq \tilde{A}_1\rho_1^t\left(|\Delta q_0| + |\tilde{x}_0^a| + \left|\tilde{x}_0^b\right|\right)$$

where $\tilde{A}_1 = \gamma_{h_q}(A_1 + \gamma_h\beta_o)$ and $\gamma_{h_q} \in \mathbb{R}^+$.

We now consider the system mapping $(\eta, \tilde{u}) \mapsto z$ under the base controller, which is contracting with some rate $\alpha_b \in [0, 1)$ and overshoot $\beta_b \in \mathbb{R}^+$ (**A1**). By applying Lemma A.1 and following a similar procedure as above, we have

$$|\Delta\xi_t| \leq A_2\rho_2^t\left(|\Delta\xi_0| + |\Delta q_0| + \left|\tilde{x}_0^1\right| + \left|\tilde{x}_0^2\right|\right) \tag{5.26}$$

with $\max(\rho_1, \alpha_b) \leq \rho_2 < 1$ and $A_2 \in \mathbb{R}^+$ depending on $\tilde{A}_1, \beta_b$ and Lipschitz constants of $f$, $f_o$, $f_b$, and $k$. Here $\xi_t = [x_t; s_t]$ is the state of the base-controlled system $(\eta, \tilde{u}) \mapsto z$. By summing (5.25) with (5.26) we finally obtain that

$$|\Delta \bar{x}_t| \leq A\rho_2^t \left( |\Delta \xi_0| + |\Delta q_0| + |\tilde{x}_0^a| + |\tilde{x}_0^b| \right), \tag{5.27}$$

where $A = A_1 + A_2$. □

*Claim 2.* Now consider the trajectories $(\bar{x}^a, \eta^a, z^a)$ and $(\bar{x}^b, \eta^b, z^b)$. The closed-loop system $(\eta, \tilde{u}) \mapsto z$ under the base controller is Lipschitz (**A1**), as is the Youla parameter $\tilde{u} = \mathcal{Q}(\eta, \tilde{y})$. Therefore, there exist $\gamma_1, \gamma_2 \in \mathbb{R}^+$ and $\kappa_1(\xi_0^a, \xi^2 b_0), \kappa_2(q_0^a, q_0^b) \geq 0$ with $\kappa_i(a, a) = 0$ for $i = 1, 2$ s.t.

$$\begin{aligned} \|\Delta z\|_T &\leq \gamma_1 \|\Delta \eta\|_T + \gamma_1 \|\Delta \tilde{u}\|_T + \kappa_1(\xi_0^a, \xi_0^b), \\ \|\Delta \tilde{u}\|_T &\leq \gamma_2 \|\Delta \eta\|_T + \gamma_2 \|\Delta \tilde{y}\|_T + \kappa_2(q_0^a, q_0^b). \end{aligned}$$

Since the output mapping $h$ is Lipschitz, we have $\|\Delta \tilde{y}\|_T \leq \gamma_h \|\Delta \tilde{x}\|_T \leq \gamma_h(\|\tilde{x}^a\|_T + \|\tilde{x}^b\|_T)$ and so

$$\|\Delta \tilde{y}\|_T \leq \frac{\gamma_h \beta_o(|\tilde{x}_0^a| + |\tilde{x}_0^b)}{\sqrt{1 - \alpha_o^2}}.$$

Combining the above inequalities gives (5.9) with $\bar{\gamma} = \gamma_1(1 + \gamma_2)$, $\tilde{\gamma} = \gamma_1 \gamma_2 \gamma_h$ and $\kappa = \kappa_1 + \gamma_1 \kappa_2$. □

## Proof of Proposition 5.3

*Proof.* First consider the LTI system (4.8) with the linear observer from (4.9). The innovations depend only on the observer error $\tilde{x}$ and external disturbances, since

$$\begin{aligned} \tilde{x}_{t+1} &= (A - LC)\tilde{x}_t - Lv_t + w_t \\ \tilde{y}_t &= C\tilde{x}_t + v_t. \end{aligned}$$

This system is contracting and Lipschitz and of the form (5.10).

Next, consider the fully-observed nonlinear system (4.15) with innovations (4.17). We can re-write (4.17) as

$$\psi_{t+1} = w_t, \quad \tilde{y}_t = \psi_t$$

which is also contracting and Lipschitz and of the form (5.10).

Finally, consider the partially-observed nonlinear system (4.1) with the observer (4.4). Under zero disturbances ($d \equiv 0$, $\tilde{x}_0 \equiv 0$), by observer correctness (**A3**) the observer is an exact replica of the plant, hence $\hat{x}_t = x_t \; \forall \, t \in \mathbb{N}$. Therefore $\tilde{y}_t = 0 \; \forall \, t \in \mathbb{N}$, which is a trivial case of (5.10). □

## Proof of Theorem 5.4

*Proof.* Since all of $\mathcal{T}$, $\mathcal{Q}$, and $\mathcal{G}_\Delta$ are contracting and Lipschitz by assumption, then the result follows by the series composition properties of contracting systems and Lipschitz mappings. □

## Proof of Proposition 5.5

*Proof.* To simplify notation, we introduce $\bar{d}_t := [\eta_t, d_t]$ and denote the state of $\bar{\mathcal{G}}_\Delta$ by $\breve{x}_t := [x_t; s_t, \hat{x}_t] \in \mathbb{R}^{2n_x+n_s}$. The systems $\bar{\mathcal{G}}_\Delta$ and $\bar{\mathcal{Q}}$ are assumed to be both contracting and dissipative in the sense of incremental IQCs (Definition 2.2). From the well-established theory of dissipative systems [186, 175], and from [169, Thms. 11 & 15] for contracting systems, there exist quadratically-bounded, incremental storage functions $V_t^g = V_t^g(\breve{x}_t^a, \breve{x}_t^b)$, $V_t^q = V_t^q(q_t^a, q_t^b)$ such that

$$V_{t+1}^g - V_t^g \leq \begin{bmatrix} \Delta z_t \\ \Delta \bar{y}_t \\ \Delta \bar{d}_t \\ \Delta \bar{u}_t \end{bmatrix}^\top \begin{bmatrix} Q & S^\top \\ S & R \end{bmatrix} \begin{bmatrix} \Delta z_t \\ \Delta \bar{y}_t \\ \Delta \bar{d}_t \\ \Delta \bar{u}_t \end{bmatrix}, \quad V_{t+1}^q - V_t^q \leq \begin{bmatrix} \Delta \bar{u}_t \\ \Delta \bar{y}_t \end{bmatrix}^\top \begin{bmatrix} \bar{Q} & \bar{S}^\top \\ \bar{S} & \bar{R} \end{bmatrix} \begin{bmatrix} \Delta \bar{u}_t \\ \Delta \bar{y}_t \end{bmatrix}$$

and $V_t^g(\breve{x}_t^a, \breve{x}_t^a) = V_t^q(q_t^a, q_t^a) = 0$ for any $\breve{x}_t^a, \breve{x}_t^b \in \mathbb{R}^{2n_x+n_s}$ and $q_t^a, q_t^b \in \mathbb{R}^{n_q}$, where $\Delta z_t = z_t^a - z_t^b$ and similar for all other quantities. By summing the above inequalities and choosing $V_t := V_t^g + V_t^q$ as the incremental storage function, we have that the closed-loop system is incrementally dissipative with respect to the supply rate

$$V_{t+1} - V_t \leq \left[\begin{array}{c} \Delta\tilde{y}_t \\ \Delta\tilde{u}_t \\ \Delta z_t \\ \hline \Delta\bar{d}_t \end{array}\right]^\top \left[\begin{array}{ccc|c} Q_{\bar{y}\bar{y}} + \bar{R} & S_{\bar{u}\bar{y}}^\top + \bar{S} & Q_{z\bar{y}} & S_{d\bar{y}}^\top \\ S_{\bar{u}\bar{y}} + \bar{S}^\top & R_{\bar{u}\bar{u}} + \bar{Q} & S_{\bar{u}z} & R_{d\bar{u}} \\ Q_{z\bar{y}}^\top & S_{\bar{u}z}^\top & Q_{zz} & S_{dz}^\top \\ \hline S_{d\bar{y}} & R_{d\bar{u}}^\top & S_{dz} & R_{dd} \end{array}\right] \left[\begin{array}{c} \Delta\tilde{y}_t \\ \Delta\tilde{u}_t \\ \Delta z_t \\ \hline \Delta\bar{d}_t \end{array}\right]. \tag{5.28}$$

The upper-left block of the supply rate matrix is negative definite by assumption. Furthermore, each of $|\Delta\tilde{y}_t|, |\Delta\tilde{u}_t|, |\Delta\tilde{z}_t|$ are bounded from above by $|\Delta\bar{x}_t|$, where $\bar{x}_t = [x_t; s_t, \hat{x}_t; q_t]$, due to the assumption of Lipschitz continuous output maps (**A4**). Hence, when $\Delta\bar{d}_t = 0$, there exists a $c \in \mathbb{R}^+$ such that

$$V_{t+1} - V_t \leq -c|\Delta\bar{x}_t|^2$$

where $V_t$ is quadratically-bounded by $|\Delta x_t^2|$. The closed-loop system is therefore contracting by [169, Thms. 11 & 15].

For the closed-loop system $\bar{d} \mapsto z$ to also be Lipschitz, we need a $\gamma \in \mathbb{R}^+$ such that

$$V_{t+1} - V_t \leq \begin{bmatrix} \Delta z_t \\ \Delta\bar{d}_t \end{bmatrix}^\top \begin{bmatrix} -\frac{1}{\gamma}I & 0 \\ 0 & \gamma I \end{bmatrix} \begin{bmatrix} \Delta z_t \\ \Delta\bar{d}_t \end{bmatrix}.$$

Using this as an upper bound on the inequality (5.28) for the closed-loop system, the LMI to be satisfied is

$$\left[\begin{array}{ccc|c} Q_{\bar{y}\bar{y}} + \bar{R} & S_{\bar{u}\bar{y}}^\top + \bar{S} & Q_{z\bar{y}} & S_{d\bar{y}}^\top \\ S_{\bar{u}\bar{y}} + \bar{S}^\top & R_{\bar{u}\bar{u}} + \bar{Q} & S_{\bar{u}z} & R_{d\bar{u}} \\ Q_{z\bar{y}}^\top & S_{\bar{u}z}^\top & Q_{zz} + \frac{1}{\gamma}I & S_{dz}^\top \\ \hline S_{d\bar{y}} & R_{d\bar{u}}^\top & S_{dz} & R_{dd} - \gamma I \end{array}\right] \prec 0. \tag{5.29}$$

Taking the Schur complement of the left-hand side of (5.29) about the top-left block, we can always choose a sufficiently large $\gamma$ such that the resulting expression is negative definite. Hence, any Youla parameter admitting the incremental IQC defined by $(\bar{Q}, \bar{S}, \bar{R})$ which satisfies (5.11) will also satisfy (5.29) for some $\gamma \in \mathbb{R}^+$, so the closed-loop system is Lipschitz. □

## Proof of Theorem 5.6 and Associated Corollaries

*Theorem 5.6 Claim 1.* Since both of $\tilde{\mathcal{G}}_{\mathcal{K}_b}$, $\mathcal{Q}$ are contracting and Lipschitz by assumption, then so is the closed-loop system $\hat{z} = \tilde{\mathcal{G}}_{\mathcal{K}_b}(\eta, \tilde{y}, \mathcal{Q}(\eta, \tilde{y}))$ by the series composition properties of contracting systems and Lipschitz mappings. □

*Theorem 5.6 Claim 2.* The closed-loop system $(\eta, \tilde{y}) \mapsto \hat{z}$ consisting of (5.12) in feedback with $u = \mathcal{K}(\eta, y)$ (4.11) has dynamics

$$\begin{aligned}
\hat{x}_{t+1} &= f_o(\hat{x}_t, \eta_t, h_{\mathcal{K}}(\phi_t, \eta_t, h(\hat{x}_t) + \tilde{y}_t), h(\hat{x}_t) + \tilde{y}_t) \\
s_{t+1} &= f_b(s_t, \eta_t, h_{\mathcal{K}}(\phi_t, \eta_t, h(\hat{x}_t) + \tilde{y}_t), h(\hat{x}_t) + \tilde{y}_t) \\
\phi_{t+1} &= f_{\mathcal{K}}(\phi_t, \eta_t, h(\hat{x}_t) + \tilde{y}_t) \\
\hat{z}_t &= [\hat{x}_t; s_t; h_{\mathcal{K}}(\phi_t, \eta_t, h(\hat{x}_t) + \tilde{y}_t)].
\end{aligned} \tag{5.30}$$

This system is contracting and Lipschitz by assumption. We will use it to show that $\mathcal{K}$ can be re-written in the form (4.7) with a contracting and Lipschitz Youla parameter $\mathcal{Q}_{\mathcal{K}}$.

Augment $\mathcal{K}$ with a base controller (4.3) and observer (4.4). Using $y_t = \tilde{y}_t + h(\hat{x}_t)$ and $\tilde{u}_t = u_t - \hat{u}_t$, the resulting system $\mathcal{Q}_{\mathcal{K}} : (\eta, \tilde{y}) \mapsto \tilde{u}$ has the same state dynamics as (5.30) and is therefore contracting by assumption. Its output map is

$$\tilde{u}_t = h_{\mathcal{K}}(\phi_t, \eta_t, h(\hat{x}_t) + \tilde{y}_t) - k(s_t, \eta_t, h(\hat{x}_t) + \tilde{y}_t),$$

hence $\mathcal{Q}_{\mathcal{K}}$ is also Lipschitz since $(\eta, \tilde{y}) \mapsto \hat{z}$ is Lipschitz by assumption, $k, h$ are Lipschitz

functions (**A4**), and

$$\tilde{u}_t = [0, 0, I]\hat{z}_t - k([0, I, 0]\hat{z}_t, \eta_t, h([I, 0, 0]\hat{z}_t) + \tilde{y}_t).$$

Moreover, if $\mathcal{Q} = \mathcal{Q}_\mathcal{K}$ in the Youla architecture (4.7), then the control signal $u_t = \hat{u}_t + h_\mathcal{K}(\phi_t, \eta_t, y_t) - \hat{u}_t$ is unchanged from $u = \mathcal{K}(\eta, y)$. Hence $\mathcal{K}$ can be written in the form (4.7) with $\mathcal{Q} = \mathcal{Q}_\mathcal{K}$ contracting and Lipschitz. □

*Corollary 5.7.* The closed-loop system $(\eta, w) \mapsto z$ consisting of (4.15) in feedback with $u = \mathcal{K}(\eta, y)$ (4.11) has dynamics

$$\begin{aligned} x_{t+1} &= f(x_t, \eta_t, h_\mathcal{K}(\phi_t, \eta_t, x_t)) + w_t \\ \phi_{t+1} &= f_\mathcal{K}(\phi_t, \eta_t, x_t) \\ z_t &= [x_t; h_\mathcal{K}(\phi_t, \eta_t, x_t)]. \end{aligned} \tag{5.31}$$

This system is contracting and Lipschitz by assumption. Make the substitutions $x_t = \hat{x}_t + w_{t-1}$ and $\tilde{y}_t = w_{t-1}$ which follow by (4.5), (4.17) since $y_t = x_t$. Then (5.31) becomes

$$\begin{aligned} \hat{x}_{t+1} &= f(\hat{x}_t + \tilde{y}_t, \eta_t, h_\mathcal{K}(\phi_t, \eta_t, \hat{x}_t + \tilde{y}_t)) \\ \phi_{t+1} &= f_\mathcal{K}(\phi_t, \eta_t, \hat{x}_t + \tilde{y}_t) \\ z_t &= [\hat{x}_t + \tilde{y}_t; h_\mathcal{K}(\phi_t, \eta_t, \hat{x}_t + \tilde{y}_t)]. \end{aligned} \tag{5.32}$$

This system is exactly (5.30) with the observer $\hat{x}_t = f(y_t, \eta_t, u_t)$ but for the addition of $\tilde{y}_t$ in the output $z_t$, which does not change the fact that it is contracting or Lipschitz. The result then follows by Theorem 5.6. □

*Corollary 5.8.* The closed-loop system $(\eta, d) \mapsto z$ consisting of (4.1) in feedback with $u = \mathcal{K}(\eta, y)$ (4.11) has dynamics

$$\begin{aligned} x_{t+1} &= f(x_t, \eta_t, h_\mathcal{K}(\phi_t, \eta_t, h(x_t) + v_t)) + w_t \\ \phi_{t+1} &= f_\mathcal{K}(\phi_t, \eta_t, h(x_t) + v_t) \\ z_t &= [x_t; h_\mathcal{K}(\phi_t, \eta_t, h(x_t) + v_t)]. \end{aligned} \tag{4.22}$$

This system is contracting and Lipschitz by assumption. Take (4.22), re-label $x$ as $\hat{x}$, and choose $v_t = \tilde{y}_t$, $w_t = L(\eta_t, \tilde{y}_t)$. Then the dynamics are exactly (5.30) when the base controller and observer have the form (5.13). The above transformation is Lipschitz (because $L$ is Lipschitz), hence if (4.22) is contracting and Lipschitz so too is (5.30). The result follows by Theorem 5.6. □

# Chapter 6

# Robust Recurrent Deep Networks

In Chapter 5, we saw how RENs [136] could be used to parametrise stable and robust nonlinear systems for leaning-based control. The key advantages of RENs were the expressivity of the model class and its direct parametrisation, which allowed us to train stable and robust models with unconstrained optimisation tools such as deep RL. However, the equilibrium layer in a REN fundamentally restricts the scalability of the model class to large network sizes with many neurons. Large models are typically necessary for processing high-dimensional information such as images, video, or language data, and are increasingly common in modern applications of machine learning [57].

This chapter presents the Robust Recurrent Deep Network (R2DN) as a scalable, computationally-efficient alternative to RENs. We construct R2DNs as a feedback interconnection of an LTI system and a 1-Lipschitz deep feedforward network – such as the Sandwich network [181] – and directly parametrise the network weights so that the models are automatically contracting and Lipschitz. Our parametrisation uses a similar structure to RENs but without relying on an equilibrium layer, making it computationally feasible to scale up the network size to solve problems requiring large models. We conclude the chapter by studying the scalability of our R2DN parametrisation in comparison to RENs, and compare the them via numerical experiments.

### Publications

The content in this chapter previously appeared as part of the following publication, which is available on the arXiv with an updated version currently in preparation.

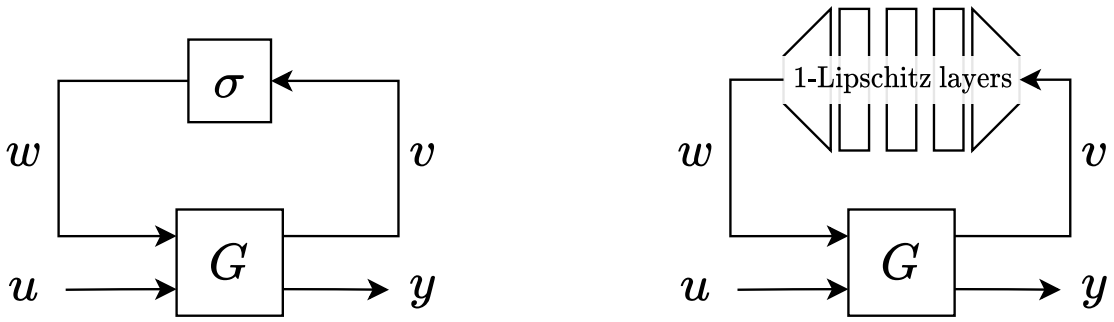
[14] Nicholas H. Barbara, Ruigang Wang, and Ian R. Manchester, "R2DN: Scalable Parametrization of Contracting and Lipschitz Recurrent Deep Networks," *In Preparation for the IEEE Control Systems Letters,* 2025.

## 6.1 Introduction

We established a need for machine learning with robust neural networks in Chapter 2, and reviewed the REN [136] as a powerful tool for robust machine learning.

Despite its many attractive features, a key limitation of RENs is the requirement to iteratively solve an equilibrium layer every time the model is called. Typical equilibrium solvers (see [133, 9]) require sequential computation and cannot efficiently leverage the parallelism that makes it feasible to train very large models on GPUs. Moreover, the direct parametrisation of the equilibrium layer in [136] does not allow the user to impose sparsity constraints or specify particular structures, such as those corresponding to a CNN. Instead, the equilibrium layer is either dense or lower-triangular (Section 2.3.3) and the number of learnable parameters scales quadratically with the number of neurons. This is acceptable for tasks requiring small models, such as learning-based control with moderately-sized dynamical systems, but is prohibitively slow for large models.

We propose the R2DN to address this limitation while retaining the flexibility, internal stability, and robustness properties of RENs. The R2DN architecture is compared to a REN in Figure 6.1. Our key observation is that a similar construction to RENs with two small tweaks to the original architecture leads to dramatic improvements in scalability and computational efficiency. The tweaks are: (a) we eliminate the equilibrium layer by removing direct feedthrough in the neural component of the LTI system; and (b) we replace scalar activation functions $\sigma$ with a (scalable) 1-Lipschitz DNN.




**(a)** REN architecture [136]. **(b)** R2DN architecture (ours).

**Figure 6.1:** Block diagrams for the REN and the proposed R2DN architectures. We replace the scalar activation function $\sigma$ with a 1-Lipschitz feedforward network, and modify the LTI system $G$ to remove direct feedthrough from $w \to v$.

## 6.2 Preliminaries

### 6.2.1 Problem Formulation

Given a dataset $\mathcal{D}$, we consider the problem of learning a nonlinear state-space models

$$x_{t+1} = f_\theta(x_t, u_t), \quad y_t = h_\theta(x_t, u_t), \tag{6.1}$$

where $x_t \in \mathbb{R}^n, u_t \in \mathbb{R}^m, y_t \in \mathbb{R}^p$ are the states, inputs, and outputs of the system at time $t \in \mathbb{N}$, respectively. Here, the dynamics and measurement functions $f_\theta : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ and $h_\theta : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^p$ are both parametrised by some learnable parameter $\theta \in \Theta \subseteq \mathbb{R}^N$ (e.g., the weights and biases of a DNN). The learning problem can be formulated as an optimisation problem

$$\min_{\theta \in \Theta} \quad \mathcal{L}(f_\theta, h_\theta; \mathcal{D}) \tag{6.2}$$

for some loss function $\mathcal{L}$. Note that problem (6.2) describes the general problem of learning nonlinear state-space models. We do not limit ourselves to the specific example of learning feedback control policies from problem (2.3) as in Chapters 3 to 5, but this is of course a special case where the dataset $\mathcal{D}$ is generated by collecting data from closed-loop trajectories of a controlled dynamical system.

The focus of this chapter is to directly parametrise stable and robust nonlinear models (6.1) in the sense of Definition 2.4. We have seen that direct parametrisations allow the training problem (6.2) to be solved using standard optimisation tools without having to project the model parameters $\theta$ into a constrained set $\Theta \subset \mathbb{R}^N$. In keeping with the theme of this thesis, we choose contraction (Definition 2.1) as our notion of internal stability, and incremental IQCs (Definition 2.2) and Lipschitz bounds (Definition 2.3) as our measures of the input-output robustness of nonlinear models (6.1).

### 6.2.2 Review of Recurrent Equilibrium Networks

We briefly review the structure of a REN since we will compare to it directly throughout this chapter. RENs (Figure 6.1a) are described by state-space models

$$\begin{bmatrix} x_{t+1} \\ v_t \\ y_t \end{bmatrix} = \overbrace{\left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right]}^{W} \begin{bmatrix} x_t \\ w_t \\ u_t \end{bmatrix} + \overbrace{\begin{bmatrix} b_x \\ b_v \\ b_y \end{bmatrix}}^{b} \tag{2.20a}$$

$$w_t = \sigma(v_t). \tag{2.20b}$$

Here, $W, b$ are the weights and biases of $\boldsymbol{G}$ in Figure 6.1a (respectively), $v_t, w_t \in \mathbb{R}^q$, and $\sigma$ is a scalar activation function satisfying assumption 2.1. If the feedthrough term $D_{11}$ is non-zero in (2.20a), this structure leads to the formation of an equilibrium layer

$$w_t = \sigma(D_{11} w_t + b_{w_t}) \tag{2.21}$$

where $b_{w_t} = C_1 x_t + D_{12} u_t + b_v$. If (2.21) is well-posed (i.e., for any $b_{w_t}$ there exists a unique $w_t$), then the equilibrium layer is also a static nonlinear map $\phi_{eq} : b_{w_t} \mapsto w_t$, and the REN (2.20) can be written in the form (6.1) with $f_\theta, h_\theta$ given by

$$\begin{aligned} f_\theta(x, u) &= Ax + B_1 \phi_{eq}(b_w) + B_2 u + b_x, \\ h_\theta(x, u) &= C_2 x + D_{21} \phi_{eq}(b_w) + D_{22} u + b_y. \end{aligned} \tag{6.3}$$

RENs were directly parametrised to be contracting and $(Q, S, R)$-dissipative in [136].

## 6.3 Robust Recurrent Deep Networks (R2DNs)

Our proposed R2DN models have the same structure as RENs but for two key differences – we remove the equilibrium layer (2.21) by setting $D_{11} = 0$, and we allow the static nonlinearity to be any static DNN rather than just a scalar activation function. Specifically, the model structure can be written as

$$\begin{bmatrix} x_{t+1} \\ v_t \\ y_t \end{bmatrix} = \overbrace{\left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & 0 & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right]}^{W} \begin{bmatrix} x_t \\ w_t \\ u_t \end{bmatrix} + \overbrace{\begin{bmatrix} b_x \\ b_v \\ b_y \end{bmatrix}}^{b} \tag{6.4a}$$

$$w_t = \phi_g(v_t), \tag{6.4b}$$

where $W, b$ are the learnable weights and biases of the LTI component $\boldsymbol{G}$ in Figure 6.1b (respectively), and $\phi_g$ is the static DNN. Here $v_t \in \mathbb{R}^q$, $w_t \in \mathbb{R}^l$, and $g \in \mathbb{R}^{N_g}$ are the input, output, and learnable parameters of $\phi_g$, respectively. The above system can be rewritten in the form (6.1) with

$$\begin{aligned} f_\theta(x, u) &= Ax + B_1\phi_g(b_w) + B_2u + b_x, \\ h_\theta(x, u) &= C_2x + D_{21}\phi_g(b_w) + D_{22}u + b_y, \end{aligned} \tag{6.5}$$

where $b_w = C_1x + D_{12}u + b_v$. We make the following assumption for the neural network $\phi_g$, leading to Proposition 6.1.

**Assumption 6.1.** *The DNN parametrisation $\mathcal{M}_\phi : g \mapsto \phi_g$ is 1-Lipschitz $\forall\, g \in \mathbb{R}^{N_g}$.*

**Proposition 6.1.** *Suppose that Assumption 6.1 holds, and* (6.4a) *is contracting and admits the incremental IQC defined by*

$$\tilde{Q} = \begin{bmatrix} -I & 0 \\ 0 & Q \end{bmatrix}, \quad \tilde{S} = \begin{bmatrix} 0 & 0 \\ 0 & S \end{bmatrix}, \quad \tilde{R} = \begin{bmatrix} I & 0 \\ 0 & R \end{bmatrix}. \tag{6.6}$$

*with* $0 \succ Q \in \mathbb{R}^{p\times p}$, $S \in \mathbb{R}^{m\times p}$ *and* $R = R^\top \in \mathbb{R}^{m\times m}$. *Then the system* (6.4) *is contracting and admits the incremental IQC defined by* $(Q, S, R)$.

*Proof.* Let $\xi^a = (x^a, u^a, y^a)$ and $\xi^b = (x^b, u^b, y^b)$ be a pair of solutions to (6.4). Then their difference $\Delta\xi := \xi^a - \xi^b$ satisfies the following dynamics:

$$\begin{bmatrix} \Delta x_{t+1} \\ \Delta v_t \\ \Delta y_t \end{bmatrix} = \begin{bmatrix} A & B_1 & B_2 \\ C_1 & 0 & D_{12} \\ C_2 & D_{21} & D_{22} \end{bmatrix} \begin{bmatrix} \Delta x_t \\ \Delta w_t \\ \Delta u_t \end{bmatrix} \tag{6.7a}$$

$$\Delta w_t = \phi_g(v_t^a) - \phi_g(v_t^b). \tag{6.7b}$$

From Assumption 6.1, we can obtain that

$$|\Delta v_t|^2 - |\Delta w_t|^2 \geq 0, \quad \forall t \in \mathbb{N} \tag{6.8}$$

for any $(\Delta v, \Delta w)$ satisfying (6.7b). Moreover, since (6.7a) is contracting and admits the incremental IQC defined by (6.6), then there exists a $P \succ 0$ such that

$$\begin{aligned} |\Delta x_{t+1}|_P^2 - |\Delta x_t|_P^2 &\leq \begin{bmatrix} \Delta y_t \\ \Delta u_t \end{bmatrix}^\top \begin{bmatrix} Q & S^\top \\ S & R \end{bmatrix} \begin{bmatrix} \Delta y_t \\ \Delta u_t \end{bmatrix} + |\Delta w_t|^2 - |\Delta v_t|^2 \\ &\leq \begin{bmatrix} \Delta y_t \\ \Delta u_t \end{bmatrix}^\top \begin{bmatrix} Q & S^\top \\ S & R \end{bmatrix} \begin{bmatrix} \Delta y_t \\ \Delta u_t \end{bmatrix} \end{aligned}$$

where $|\Delta x_t|_P^2 := \Delta x_t^\top P \Delta x_t$. Telescoping the sum gives the IQC (2.13) in Definition 2.2 with $\kappa(a, b) = (a - b)^\top P(a - b)$. For contraction, we take $\Delta u_t = 0$ and use the fact that $Q \prec 0$. Then for $\Delta x_0 \neq 0$, we have $|\Delta x_{t+1}|_P^2 - |\Delta x_t|_P^2 < 0$ and hence there exists an $\alpha \in [0, 1)$ such that (2.9) in Definition 2.1 holds with $\beta = \sqrt{\bar{\sigma}/\underline{\sigma}}$, where $\bar{\sigma}, \underline{\sigma}$ are the maximum and minimum singular values of $P$, respectively. □

**Remark 6.1.** *Assumption 6.1 and Proposition 6.1 are based on the fact that* $\phi_g$ *admits the incremental IQC defined by* $Q = -I, S = 0$ *and* $R = I$. *This can be further extended*

*to incorporate DNNs with general $(Q, S, R)$-type IQCs, such as incrementally passive ($Q = R = 0$ and $S = I$) or strongly monotone neural networks [179].*

Proposition 6.1 allows us to decompose the parametrisation of R2DNs into two separate parts: (a) parametrisations of 1-Lipschitz DNNs $\phi_g$; and (b) parametrisations of LTI systems $\boldsymbol{G}$ admitting the IQC defined by (6.6). We saw in Chapter 3 that many direct parametrisations of 1-Lipschitz neural networks already exist for a variety of DNN architectures [112, 170, 154, 126, 6, 128]. We therefore focus on part (b) for the remainder of this chapter.

Before introducing our direct parametrisation of R2DNs, we first note that Proposition 6.1 is not as conservative as it may seem – i.e., separating the LTI system $\boldsymbol{G}$ and nonlinear map $\phi_g$ in (6.4) and applying small-gain arguments. This is because we simultaneously learn the weights of $\boldsymbol{G}$ and $\phi_g$. We outline the flexibility of this approach in the following remark.

**Remark 6.2.** *For the purposes of illustration, assume that the dimension of $y_t, u_t$ are zero. Suppose we have an interconnection of $\boldsymbol{G}$ and $\phi_g$ which is $(Q, S, R)$-dissipative but does not satisfy Assumption 6.1 and condition* (6.6)*. A standard method to reduce the conservatism of small-gain tests is to introduce multipliers (e.g., [108]) – that is, suppose there exist invertible matrices $M \in \mathbb{R}^{q\times q}, N \in \mathbb{R}^{l\times l}$ such that*

$$\|M\boldsymbol{G}N^{-1}\|_\infty < 1, \ |N\phi(M^{-1}a) - N\phi(M^{-1}b)| \leq |a - b|$$

*for any $a, b \in \mathbb{R}^q$. Then the interconnection of $\tilde{\boldsymbol{G}} = M\boldsymbol{G}N^{-1}$ and $\tilde{\phi}_g(x) = N\phi_g(M^{-1}x)$ is stable by Proposition 6.1, since the $\mathcal{H}_\infty$ norm of the LTI system $\tilde{\boldsymbol{G}}$ is equivalent to its Lipschitz bound. Since we are simultaneously learning $\boldsymbol{G}, \phi_g$, and the representation of $v, w$, we can absorb the multipliers into the model via $\tilde{\boldsymbol{G}}$ and $\tilde{\phi}_g$, giving a transformed system of the form* (6.4) *which does satisfy the assumptions.*

## 6.4 Direct Parametrisation of R2DNs

In the following subsections, we present a direct parametrisation of LTI systems (6.4a) satisfying Proposition 6.1 by choosing specific structures of $(Q, S, R)$. This allows us to achieve contracting and Lipschitz R2DNs (6.4). We closely follow the robust parametrisation of RENs in [136, Sec.V], and provide new insight on handling input-output robustness when $D_{11} = 0$.

### 6.4.1 Robust LTI Systems

Since (6.4a) is simply an LTI system, we start by introducing sufficient conditions for robustly-stable LTI systems. Specifically, we seek LTI systems $\bar{\boldsymbol{G}} : \bar{u} \mapsto \bar{y}$ admitting the incremental IQC defined by $(\bar{Q}, \bar{S}, \bar{R})$ with

$$\bar{Q} \prec 0, \quad \bar{R} - \bar{S}\bar{Q}^{-1}\bar{S}^\top \succ 0. \tag{6.9}$$

We consider state-space realisations

$$\begin{bmatrix} x_{t+1} \\ \bar{y}_t \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x_t \\ \bar{u}_t \end{bmatrix}. \tag{6.10}$$

We over-parametrise the system by introducing an invertible matrix $E$, re-writing (6.10) as

$$\begin{bmatrix} Ex_{t+1} \\ \bar{y}_t \end{bmatrix} = \begin{bmatrix} \mathcal{A} & \mathcal{B} \\ C & D \end{bmatrix} \begin{bmatrix} x_t \\ \bar{u}_t \end{bmatrix}, \tag{6.11}$$

where $A = E^{-1}\mathcal{A}$ and $B = E^{-1}\mathcal{B}$. For convenience, we introduce the following matrix

$$H := \begin{bmatrix} E^\top + E - \mathcal{P} & \mathcal{A}^\top \\ \mathcal{A} & \mathcal{P} \end{bmatrix} \tag{6.12}$$

where $\mathcal{P} = \mathcal{P}^\top$. Then we have the following result.

**Proposition 6.2.** *Suppose that* $(\bar{Q}, \bar{S}, \bar{R})$ *satisfy* (6.9). *Then the system* (6.4a) *is contracting and admits the incremental IQC defined by* $(\bar{Q}, \bar{S}, \bar{R})$ *if* $\exists\, \mathcal{P} \succ 0$ *s.t.*

$$H \succ \begin{bmatrix} \mathcal{C}^\top \\ \mathcal{B} \end{bmatrix} \mathcal{R}^{-1} \begin{bmatrix} \mathcal{C}^\top \\ \mathcal{B} \end{bmatrix}^\top - \begin{bmatrix} C^\top \\ 0 \end{bmatrix} \bar{Q} \begin{bmatrix} C^\top \\ 0 \end{bmatrix}^\top, \tag{6.13}$$

$$\mathcal{R} := \bar{R} + \bar{S}D + D^\top \bar{S}^\top + D^\top \bar{Q} D \succ 0, \tag{6.14}$$

*where* $\mathcal{C} = (D^\top \bar{Q} + \bar{S})C$ *and* $H$ *is given by* (6.12).

**Remark 6.3.** *Proposition 6.2 is a special case of [136, Thm. 3] for RENs with* $B_1, C_1, D_{11}, D_{12}, D_{21}, b_v = 0$ *in* (2.20), *i.e., no nonlinear components. Similar results also exist in classic works such as [118]. We provide the specific construction required for our parametrisation here for completeness.*

*Proof.* We first move the terms on the right-hand side of (6.13) to the left, apply a Schur complement about the $\mathcal{R}^{-1}$ term, and use the fact that $E^\top \mathcal{P}^{-1} E \succeq E + E^\top - \mathcal{P}$ [164, Sec. V]. We are left with the matrix inequality

$$\begin{bmatrix} E^\top \mathcal{P}^{-1} E + C^\top \bar{Q} C & \mathcal{C}^\top & \mathcal{A}^\top \\ \mathcal{C} & \mathcal{R} & \mathcal{B}^\top \\ \mathcal{A} & \mathcal{B} & \mathcal{P}^\top \end{bmatrix} \succ 0.$$

Again via Schur complements, we have

$$\begin{bmatrix} E^\top \mathcal{P}^{-1} E + C^\top \bar{Q} C & \mathcal{C}^\top \\ \mathcal{C} & \mathcal{R} \end{bmatrix} - \begin{bmatrix} \mathcal{A}^\top \\ \mathcal{B}^\top \end{bmatrix} \mathcal{P}^{-1} \begin{bmatrix} \mathcal{A}^\top \\ \mathcal{B}^\top \end{bmatrix}^\top \succ 0.$$

We expand the terms $\mathcal{C}$ and $\mathcal{R}$, left-multiply $[\Delta x^\top \ \Delta \bar{u}^\top]$, and right-multiply its transpose. This leads to the incremental dissipation inequality

$$V(\Delta x_{t+1}) - V(\Delta x_t) \leq \begin{bmatrix} \Delta \bar{y}_t \\ \Delta \bar{u}_t \end{bmatrix}^\top \begin{bmatrix} \bar{Q} & \bar{S}^\top \\ \bar{S} & \bar{R} \end{bmatrix} \begin{bmatrix} \Delta \bar{y}_t \\ \Delta \bar{u}_t \end{bmatrix} \tag{6.15}$$

with storage function $V(\Delta x_t) = \Delta x_t^\top E^\top \mathcal{P}^{-1} E \Delta x_t > 0$ for all $\Delta x_t \neq 0$ since $\mathcal{P} \succ 0$. Then following similar arguments to the proof of Proposition 6.1, (6.4a) is contracting and admits the incremental IQC defined by $(\bar{Q}, \bar{S}, \bar{R})$. □

### 6.4.2 Direct Parametrisation of Contracting R2DNs

Equipped with Proposition 6.2, we are now ready to directly parametrise the LTI system (6.4a) such that its corresponding R2DN (6.4) is contracting and Lipschitz. Let us first consider the simpler case of contraction. The following parametrisation is a special case of the direct parametrisation of robust RENs from [136, Eqns. (22), (23), (29)] when the REN has no nonlinear components. We provide a detailed overview of our specific parametrisation for completeness.

Since contraction is an internal stability property, we set $\Delta u_t \equiv 0$ and ignore the output $\Delta y_t$ in (6.7a):

$$\begin{bmatrix} \Delta x_{t+1} \\ \Delta v_t \end{bmatrix} = \begin{bmatrix} A & B_1 \\ C_1 & 0 \end{bmatrix} \begin{bmatrix} \Delta x_t \\ \Delta w_t \end{bmatrix}. \tag{6.16}$$

To achieve contracting R2DNs, from Proposition 6.1 we need to construct $A, B_1, C_1$ such that the above system admits the incremental IQC defined by $\bar{Q} = -I$, $\bar{S} = 0$, and $\bar{R} = I$. To do this, we apply Proposition 6.2. First, by comparing (6.16) with (6.10) we have $D = 0$ and hence $\mathcal{R} = \bar{R}$, $\mathcal{C} = 0$ in (6.13). Condition (6.13) then becomes

$$H \succ \begin{bmatrix} C_1^\top C_1 & 0 \\ 0 & \mathcal{B}_1 \mathcal{B}_1^\top \end{bmatrix} \tag{6.17}$$

where $\mathcal{B}_1 = E B_1$. To construct a direct parametrisation satisfying (6.17), we introduce the set of free, learnable variables

$$\{X \in \mathbb{R}^{2n \times 2n},\ Y \in \mathbb{R}^{n \times n},\ \mathcal{B}_1 \in \mathbb{R}^{n \times l},\ C_1 \in \mathbb{R}^{q \times n}\}.$$

We then construct $H$ and partition it as follows

$$H = X^\top X + \epsilon I + \begin{bmatrix} C_1^\top C_1 & 0 \\ 0 & \mathcal{B}_1 \mathcal{B}_1^\top \end{bmatrix} = \begin{bmatrix} H_{11} & H_{21}^\top \\ H_{21} & H_{22} \end{bmatrix}, \tag{6.18}$$

where $H_{11}, H_{22} \in \mathbb{R}^{n \times n}$ and $\epsilon$ is a small positive scalar. Comparing (6.18) with (6.12) gives

$$E + E^\top = H_{11} + H_{22}, \quad \mathcal{P} = H_{22}, \quad \mathcal{A} = H_{21}. \tag{6.19}$$

We therefore choose

$$\begin{aligned} E &= \frac{1}{2}(H_{11} + H_{22} + Y - Y^\top), \\ A &= E^{-1} H_{21}, \quad B_1 = E^{-1} \mathcal{B}_1, \end{aligned} \tag{6.20}$$

such that the condition on $E$ in (6.19) is satisfied for any choice of $Y$. Since the remaining parameters $\{B_2, D_{12}, C_2, D_{21}, D_{22}, b\}$ in (6.4a) do not affect the contraction condition (6.17), we leave them as free learnable parameters. The above parametrisation therefore satisfies (6.17) by construction, and thus the resulting R2DN is contracting.

### 6.4.3 Direct Parametrisation of Lipschitz R2DNs

Having directly parametrised contracting R2DNs, we now seek to impose bounds on their input-output response. In this section, we directly parametrise the LTI system (6.4a) such that its corresponding R2DN is both contracting and $\gamma$-Lipschitz.

We follow a similar procedure to the previous section. First, re-write (6.7a) in the form (6.4a) with

$$B = \begin{bmatrix} B_1 & B_2 \end{bmatrix}, \quad C = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix}, \quad D = \begin{bmatrix} 0 & D_{12} \\ D_{21} & D_{22} \end{bmatrix}. \tag{6.21}$$

To achieve contracting and $\gamma$-Lipschitz R2DNs, from Proposition 6.1 we need to construct (6.21) such that (6.4a) admits the incremental IQC defined by

$$\bar{Q} = \begin{bmatrix} -I & 0 \\ 0 & -\frac{1}{\gamma} I \end{bmatrix}, \quad \bar{S} = 0, \quad \bar{R} = \begin{bmatrix} I & 0 \\ 0 & \gamma I \end{bmatrix}.$$

Once again, we seek to apply Proposition 6.2. If $D$ were dense, then we could directly apply the direct parametrisation of robust RENs from [136] by removing the nonlinear components. Instead, we must ensure that the upper-left block of $D$ in (6.21) is zero (i.e., $D_{11} = 0$). We therefore require

$$\mathcal{R} = \begin{bmatrix} I - \frac{1}{\gamma} D_{21}^\top D_{21} & -\frac{1}{\gamma} D_{21}^\top D_{22} \\ -\frac{1}{\gamma} D_{22}^\top D_{21} & \gamma I - \frac{1}{\gamma} D_{22}^\top D_{22} - D_{12}^\top D_{12} \end{bmatrix} \succ 0. \tag{6.22}$$

Parametrising $D$ in (6.21) such that $\mathcal{R} \succ 0$ is not trivial. For simplicity, we take $D_{22} = 0$ and re-write the condition as

$$\mathcal{R} = \begin{bmatrix} I - \frac{1}{\gamma} D_{21}^\top D_{21} & 0 \\ 0 & \gamma I - D_{12}^\top D_{12} \end{bmatrix} \succ 0. \tag{6.23}$$

Simple substitution and manipulation for (6.13) yields

$$H \succ \Gamma \mathcal{R}^{-1} \Gamma^\top + \frac{1}{\gamma} \begin{bmatrix} C_1^\top C_1 + C_2^\top C_2 & 0 \\ 0 & 0 \end{bmatrix} \tag{6.24}$$

with

$$\Gamma = \begin{bmatrix} -\frac{1}{\gamma} C_2^\top D_{21} & -C_1^\top D_{12} \\ \mathcal{B}_1 & \mathcal{B}_2 \end{bmatrix}. \tag{6.25}$$

To construct a direct parametrisation satisfying (6.24), we introduce the same free, learnable variables $\{X, Y, \mathcal{B}_1, C_1\}$ as the contraction case, in addition to $\{\mathcal{B}_2 \in \mathbb{R}^{n \times m}, C_2 \in \mathbb{R}^{p \times n}\}$. We then construct $H$ as

$$H = X^\top X + \epsilon I + \Gamma \mathcal{R}^{-1} \Gamma^\top + \frac{1}{\gamma} \begin{bmatrix} C_1^\top C_1 + C_2^\top C_2 & 0 \\ 0 & 0 \end{bmatrix} \tag{6.26}$$

for a small $\epsilon > 0$ and partition it similarly to (6.17), choosing $E$, $A$, and $B_1$ as per (6.20) and $B_2 = E^{-1}\mathcal{B}_2$. All that remains is to directly parametrise $D_{12}, D_{21}$ so that $\mathcal{R} \succ 0$ in (6.23). This can be achieved by choosing $D_{12} = \sqrt{\gamma}\mathcal{D}_{12}$, $D_{21} = \sqrt{\gamma}\mathcal{D}_{21}$ and directly parametrising the right-hand sides such that $\mathcal{D}_{12}^\top\mathcal{D}_{12} \prec I$, $\mathcal{D}_{21}^\top\mathcal{D}_{21} \prec I$ via the Cayley transform (see [136, Prop. 1]). We outline the process below.

Take $\mathcal{D}_{12} \in \mathbb{R}^{q\times m}$ as an example. If $q \geq m$, we introduce additional learnable parameters $\{X_{12} \in \mathbb{R}^{m\times m}, Y_{12} \in \mathbb{R}^{(q-m)\times m}, Z_{12} \in \mathbb{R}^{m\times m}\}$ and define the Cayley transform

$$\mathcal{D}_{12} = \begin{bmatrix} (I - M_{12})(I + M_{12})^{-1} \\ -2Y_{12}(I + M_{12})^{-1} \end{bmatrix} \tag{6.27}$$

where $M_{12} = X_{12} - X_{12}^\top + Y_{12}^\top Y_{12} + Z_{12}^\top Z_{12} + \epsilon I$ for some small $\epsilon > 0$. If $q < m$, then we parametrise $\mathcal{D}_{12}^\top$ via (6.27). The same approach can be used to parametrise $\mathcal{D}_{21}$.

Combining (6.20), (6.26), (6.27), the above parametrisation therefore satisfies (6.24) by construction, and thus the resulting R2DN is contracting and $\gamma$-Lipschitz.

## 6.5 Qualitative Comparison of RENs and R2DNs

Both the direct parametrisation of RENs in [136] and that of R2DNs in Section 6.4 ensure that the resulting models are contracting and Lipschitz by construction. The key design decision that separates the two is setting $D_{11} = 0$ for R2DNs. We summarize the advantages of this decision below.

**(1) Efficient GPU computation:** For RENs, solving the equilibrium layer (2.21) with general $D_{11}$ is slow and often involves iterative solvers (see [133]) which can be computationally prohibitive for large-scale problems. If $D_{11}$ is parametrised to be strictly lower-triangular as in [136], then (2.21) can be solved row-by-row, which provides a significant speed boost with minimal loss in performance. Even still, having to run a sequential solver every time the model is called is inefficient, particularly on GPUs, which are designed to leverage massive parallelism rather than sequential

computation. R2DNs do not have to solve an equilibrium layer and can take full advantage of modern GPU architectures for efficient computation.

**(2) Design flexibility:** The proposed parametrisation is flexible in that we can choose $\phi_g$ to be any 1-Lipschitz feedforward network. This opens up the possibility of using structured Lipschitz networks such as MLPs [181], CNNs [125], ResNets [6], or transformer-like architectures [128], depending on the desired application. In contrast, the existing parametrisation of contracting and Lipschitz RENs in [136] only allows for $D_{11}$ with particular structures (full or strictly lower-triangular). While, in principle, the REN (2.20) contains many of the above network architectures as special cases, it is not obvious how to parametrise a well-posed contracting and Lipschitz REN with a structured equilibrium layer. This limits its application in high-dimensional problems such as processing language or image data, where particular network structures are known to be critical for efficient training [176]. R2DN has no such restriction.

**(3) Model size and scalability:** RENs typically have many more parameters than R2DNs given the same number of neurons due to the structure of the equilibrium layer (2.21). Given $d$ neurons, the number of parameters in $\phi_{eq}$ is proportional to $d^2$. In an R2DN, the number of parameters in $\phi_g$ – parametrised by, for example, an $L$-layer MLP with $d$ neurons – is proportional to $d^2/L$. That is, the number of parameters scales linearly with the depth of the network, and if the model size is held constant between $\phi_{eq}$ for a REN and $\phi_g$ for an R2DN, then $\phi_g$ has more neurons than $\phi_{eq}$.

For problems requiring models with large state dimensions $n$, it may also be desirable for the LTI component (6.4a) to have a scalable parametrisation in addition to the nonlinear component $\phi_g$. The number of learnable parameters for the LTI component scales proportionally to $n^2$ in the parametrisations from Section 6.4 due to the $X^\top X$ terms in (6.17), (6.24). There are several options to mitigate this, two of which are:

1. **Low-rank parametrisation:** Introduce new parameters $\delta \in \mathbb{R}^{2n}$ and $\bar{X} \in \mathbb{R}^{2n\times\nu}$ with $\nu \ll n$, then replace $X^\top X$ with $\bar{X}^\top\bar{X} + \mathrm{diag}(\delta)$ in (6.17), (6.24). The

LTI system (6.4a) remains dense, but the number of learnable parameters scales linearly with $n$.

2. **Parallel components:** Replace (6.4a) with many smaller, parallel LTI systems which can be separately or jointly-parametrised. In the limit that each system is scalar, the number of parameters scales linearly with $n$. Note that parallel interconnections of 1-Lipschitz systems preserve the Lipschitz bound, hence our parametrisation remains valid.

We leave a detailed study of the effect of scalable LTI parametrisations in RENs and R2DNs to future work.

## 6.6 Numerical Experiments

In this section, we study the scalability and computational benefits of R2DNs over RENs via numerical experiments. All experiments[1] were performed in Python using JAX [22] on an NVIDIA GeForce RTX 4090. R2DN models were implemented with 1-Lipschitz MLPs constructed from Sandwich layers [181], and RENs were implemented with lower-triangular $D_{11}$. We focus our study on contracting RENs and R2DNs as a preliminary investigation.

### 6.6.1 Scalability and Expressive Power

We first show that computation time for R2DNs scales more favourably with respect to the network's expressive power (expressivity) in comparison to RENs. It is difficult to quantify the expressivity of each network architecture with simple heuristics like the total number of learnable parameters or activations. For example, we could distribute the parameters of similarly-sized networks in many different ways between the linear and nonlinear components (for RENs and R2DNs) and between the width and depth of the feedforward network (for R2DNs). Instead, we used the following heuristic.

[1] `https://github.com/nic-barbara/R2DN`

We fit the internal dynamics $f_\theta$ from (6.3), (6.5) for RENs and R2DNs (respectively) to a scalar nonlinear function

$$f(x,u) = 0.05x + 0.2\sin(x) + u + 0.05\cos(2\bar{x}) + \\ 0.05\sin(3\bar{x}) + 0.075\sin(4\bar{x})\tan^{-1}(0.1\bar{x}^2)$$

using supervised learning, where $\bar{x} := x + u$. The function is plotted in Figure 6.2 and has a maximum slope with respect to $x$ of $|\frac{\partial}{\partial x} f(x,u)| \lessapprox 0.9$. It is not in either of the REN or R2DN model classes, but can be approximated given a sufficiently large number of neurons in $\phi_{eq}$ or $\phi_g$, respectively. We then computed the Normalised Root-Mean-Square Error (NRMSE)

$$\text{NRMSE} = \frac{\|f(x,u) - f_\theta(x,u)\|}{\|f(x,u)\|} \times 100$$

for test batches of $x, u$ and took 1/NRMSE as a measure of the network's expressivity.

All models were trained over 1500 epochs using the Adam optimiser [88] with an initial learning rate of $10^{-3}$ which we decreased by a factor of 10 every 500 training epochs to ensure convergence. We uniformly sampled $x \in [-30, 30]$, $u \in [-1, 1]$ in training and

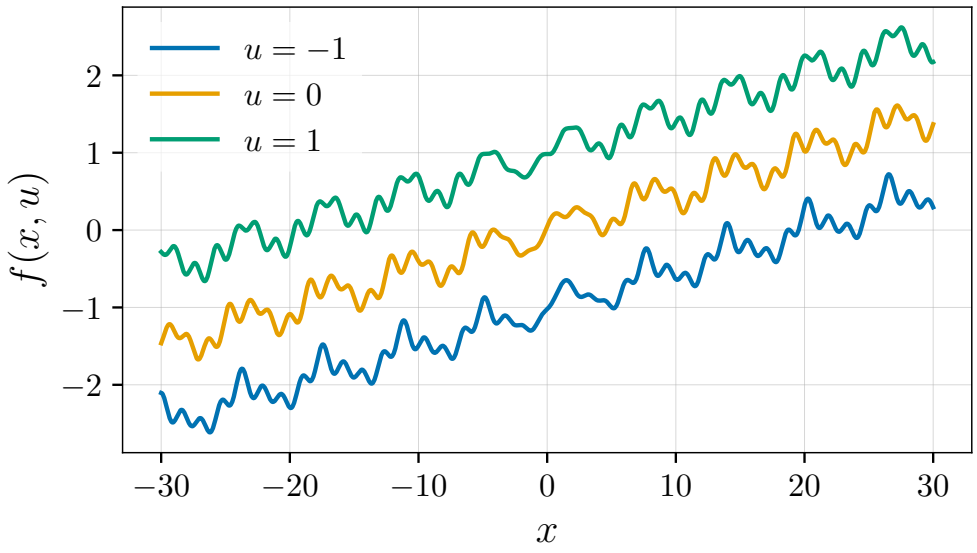
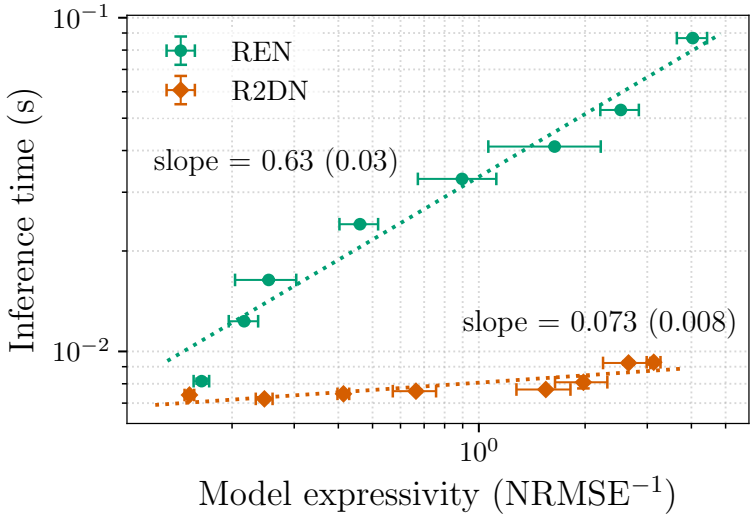
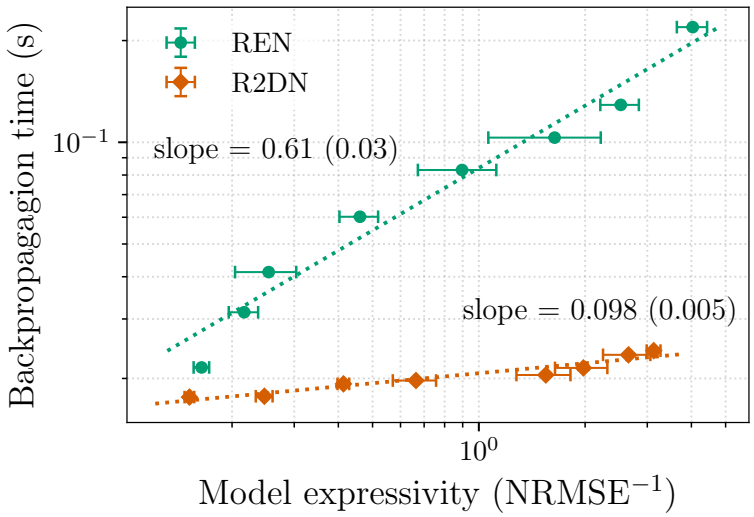



**Figure 6.2:** The function $f(x,u)$ to be fitted




**(a)** Inference scaling relations.




**(b)** Backpropagation scaling relations.

**Figure 6.3:** Computation time for the forwards (a) and backwards (b) passes as functions of model expressivity for the RENs and R2DNs. Computation time scales more favourably for the R2DN models. Error bars show one standard-deviation across 5 random model initialisations for each data point. Slope standard deviations are in parentheses.

test batches of $128 \times 512$ and 2048 samples, respectively. We trained models from each class with $n = 1$ internal states and increasingly large nonlinear components $\phi_{eq}$, $\phi_g$. For RENs, we varied the number of neurons over $q \in [20, 200]$. For R2DNs, we fixed $q = l = 16$ and designed $\phi_g$ with six layers each of width $n_h$, varying the width over $n_h \in [8, 96]$. We trained 5 models for each architecture and size, each with a different

random initialisation.

The results in Figure 6.3 show how mean computation time scales with model expressivity for each network architecture. Computation time was measured by evaluating the mean inference and backpropagation (gradient calculation) time over 1000 function calls for each model, using sequences of length 128 with a batch size of 64. We excluded the time taken for the first function call, which is typically much longer than every subsequent function call due to just-in-time compilation with JAX [22]. In both cases, computation time increases with model expressivity. However, the increase occurs at a much faster rate for the REN models, whereas R2DNs can clearly be scaled to more expressive models with minimal increase in training and inference time. This bodes well for future the application of R2DNs to large-scale machine-learning tasks which require very large recurrent models.

### 6.6.2 Training Speed and Test Performance

We now compare the performance of each model class on the three case studies introduced in [136] for RENs, which we briefly summarise below.

1. **System identification:** stable and robust nonlinear system identification on the F16 ground vibration dataset [115]. The task is to learn the vibration dynamics of the wing of an F16 aircraft from data. The dataset was collected in [115] by mounting a shaker under the right wing of a full-scale F16 aircraft. It contains measurements from accelerometers at three sites along the wing, and also the shaker voltage and force applied to generate the vibrations.

2. **Observer design:** learning nonlinear observers for Partial Differential Equations (PDEs). The task is to learn an observer of the form $\hat{x}_{t+1} = f_\theta(\hat{x}_t, u_t, y_t)$ which estimates the state of a reaction-diffusion PDE described by

$$\frac{\partial x}{\partial t}(\zeta, t) = \frac{\partial^2 x}{\partial \zeta^2} + \frac{1}{2}x(1-x)\left(x - \frac{1}{2}\right), \tag{6.28}$$

where the state $x = x(\zeta, t)$ is a function of both a spatial coordinate $\zeta \in [0, 1]$ and time $t \in \mathbb{R}^+$. Boundary conditions are set as $x(\zeta, 0) = 1$, $x(1, t) = x(0, t) = u(t)$ for some known forcing function $u(t)$, and the state is measured at $y = x(0.5, t)$.

As outlined in Chapters 4 and 5, the observer error will exponentially converge to zero over time if the observer: (a) is contracting; and (b) satisfies a correctness condition. Since contraction is already guaranteed by both REN and R2DN parametrisations, the objective is then to learn the one-step-ahead prediction error of a discretised version of (6.28) to ensure that, in the absence of initial condition error or other perturbations, the true state can be exactly generated by the observer. Further details are provided in [136, Sec. VIII].

3. **Youla control:** data-driven nonlinear feedback control design with the Youla-Kučera parametrisation. The task is to learn Youla-REN and Youla-R2DN controllers for the discrete-time LTI system

$$\begin{bmatrix} z \\ \tilde{y} \end{bmatrix} = \begin{bmatrix} \mathcal{T}_0 & \mathcal{T}_1 \\ \mathcal{T}_2 & 0 \end{bmatrix} \begin{bmatrix} w \\ \tilde{u} \end{bmatrix}$$

where $z, w, \tilde{y}, \tilde{u}$ are as defined in Chapter 4. The stable, discrete-time systems $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2$ are defined by the transfer functions

$$\mathcal{T}_0 = \mathcal{T}_1 = -\mathcal{T}_2 = \frac{0.3}{\varrho^2 - 1.6\cos(0.2\pi)\varrho + 0.8^2}$$

where $\varrho$ is the shift operator.

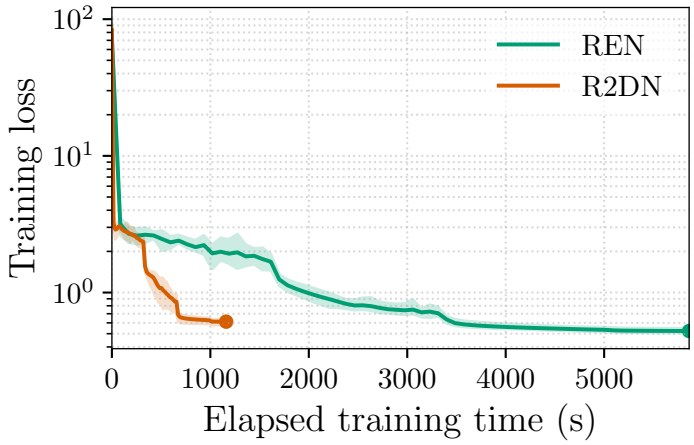
We used the same experimental setup as [136] for the first two case studies. For the third example, [136] designed Youla-REN controllers by restricting the REN to be an echo-state network [77, 192] and solving for its free parameters via convex optimisation. Instead, we took the same approach as in Chapter 5, training Youla-RENs and Youla-R2DNs with contracting models and a variant of the APG deep RL algorithm [100] (see Section 2.4.1). In each problem, we trained RENs and R2DNs with a similar number

of learnable parameters. Further training details and hyperparameters are provided in our open-source code[1].

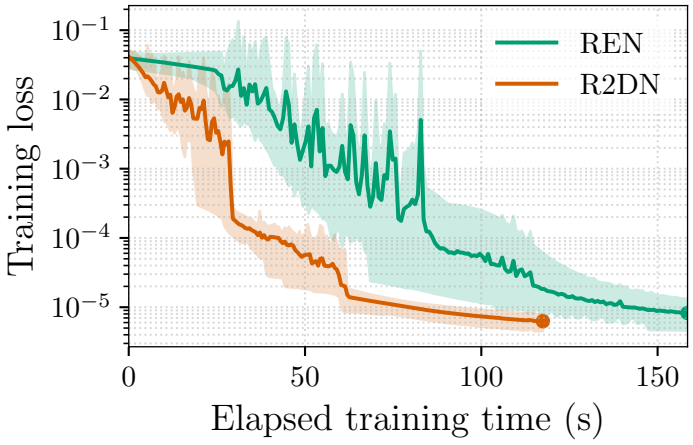
The plots in Figure 6.4 show loss curves as a function of training time for each experiment. Final test errors and mean computation time per training epoch are provided Table 6.1. It is immediately clear that the R2DN models achieve similar training and test errors to the RENs on each task, but are significantly faster to train, even though the model sizes are similar. The boost in computational efficiency is a direct benefit of not having to solve an equilibrium layer every time the model is called, which speeds up both model evaluation and backpropagation times. The benefit is most obvious for the system identification and Youla control tasks, since the models were evaluated on long sequences of data or in a closed control loop (respectively) in each training epoch. For observer design, the models were trained to minimize the one-step-ahead prediction error and so there were fewer model evaluations per epoch. The fact that R2DN matches the REN performance in each case is a clear indication that, in addition to faster training and inference, the proposed parametrisation is sufficiently expressive to capture complex nonlinear behaviour.

| **Experiment** | **Network** | **Epoch Time (s)** | **Test Error** |
|---|---|---|---|
| System ID | REN | 85.0 | 20.5 (0.22) |
| | R2DN | 16.8 | 21.7 (0.33) |
| PDE Observer | REN | 0.663 | 9.19 (1.20) |
| | R2DN | 0.565 | 8.81 (0.70) |
| Youla Control | REN | 5.66 | 1.27 (0.26) |
| | R2DN | 0.564 | 1.27 (0.47) |

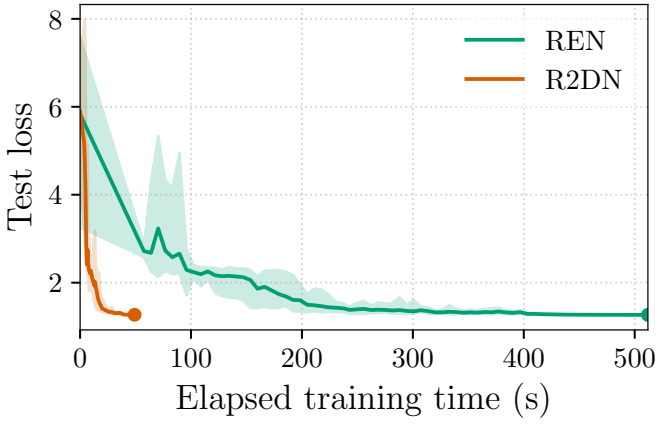
**Table 6.1:** Mean training epoch time and test error for the two network architectures. For the system identification and PDE observer examples, the test error is the final NRMSE (%). For the Youla control example, the test error is the final test cost. Test error standard deviation is in parentheses. Mean epoch time does not include compilation overhead from the first training epoch, but this is shown in Figure 6.4.




**(a)** System identification.




**(b)** PDE observer design.




**(c)** Learning-based feedback control.

**Figure 6.4:** Mean loss curves as a function of training time for each of the three benchmark problems. Bands show the loss range over 10 random model initialisations. Note that the first training step also includes the overhead from just-in-time compilation in JAX.

## 6.7 Conclusions

This chapter has introduced a parametrisation of contracting and Lipschitz R2DNs for machine learning and data-driven control. We have compared the proposed parametrisation to that of contracting and Lipschitz RENs, showing that by removing the equilibrium layer and applying small-gain arguments to the nonlinear terms, our R2DNs offer significantly more efficient computation than RENs with negligible loss in performance, and they scale more favourably with respect to model expressivity. In future work, we will remove the assumption that $D_{22} = 0$ for $\gamma$-Lipschitz R2DNs, extend the parametrisation to $(Q, S, R)$-robust R2DNs in the sense of Definition 2.2, and study the scalability of R2DNs in high-dimensional robust machine learning tasks.

# Chapter 7

# Conclusions

This thesis has studied parametrisations of neural feedback policies for learning-based control which automatically guarantee stability and robustness. We built our parametrisations on robust neural networks, which were directly parametrised to satisfy stability and robustness guarantees of their own, and which we used in place of traditional black-box neural networks for feedback control. Our main contribution was to show that with the correct choice of policy architecture, the closed-loop system can naturally inherit the stability and robustness properties of the policy network. Moreover, the parametrisation of the policy network is important – it was only by using expressive, unconstrained parametrisations that we were able to learn high-performing controllers via simple, gradient-based tools such as deep RL.

Chapter 3 began this study with an examination of Lipschitz-bounded policy networks in deep RL. We saw that the Lipschitz bound was a simple and natural heuristic with which to constrain the sensitivity of a policy to changes in its inputs, such as those induced by targeted adversarial attacks. We compared four different Lipschitz-bounded layer parametrisations in an empirical study, and found that expressive parametrisations imposing tight bounds on the Lipschitz constant were most effective in achieving both strong robustness and performance.

In Chapters 4 and 5, we shifted our focus to parametrising stabilising neural policies in the closed-loop setting. We introduced a nonlinear version of the classical Youla-Kučera parametrisation and proved that, for certain classes of nonlinear systems, it is a parametrisation of all and only incrementally stabilising controllers in the sense of contraction and Lipschitzness. Combining this with robust neural networks such as the REN [136], we proposed the Youla-REN policy class as a theoretically-motivated architecture for learning nonlinear, stabilising policies parametrised by neural networks.

We concluded our study in Chapter 6 by shifting our focus once more, this time to the scalable parametrisation of robust neural networks, to allow us to parametrise stabilising policies for large-scale control problems in future work. We introduced the R2DN as a scalable, computationally-efficient alternative to RENs by removing the dependence on an equilibrium layer. We provided a preliminary parametrisation of contracting and Lipschitz R2DNs with no direct feedthrough term, and demonstrated via numerical experiments that the parametrisation is up to an order of magnitude faster than that of contracting and Lipschitz RENs in both training and inference.

## 7.1 Future Research

We conclude this thesis with some suggestions for future research in learning-based control with stability guarantees, and the parametrisation of robust neural networks.

### Lipschitz-Bounded Policies

The empirical study of Lipschitz-bounded policy networks in Chapter 3 focused on illustrative examples. Having established the benefits of expressive parametrisations with tight bounds on the Lipschitz constant, it will be interesting to explore the impact of these policy parametrisations on the robustness of learned controllers in complex deep RL tasks, particularly those in robotics.

Specifically, we expect the Lipschitz bound to be of great help in tasks with very high-dimensional measurements spaces where the underlying state space of the system

has a much lower dimension. A common example is in vision-feedback control tasks in robotics. In these situations, it is natural to expect that small variations in the measurements (i.e., images) should not strongly affect the decision made by the policy, hence a Lipschitz-bounded policy is likely to greatly improve robustness to image noise, variations in lighting conditions, and more sinister perturbations like targeted adversarial attacks.

### The Youla Parametrisation and the Youla-REN

Our study of the Youla parametrisation in this thesis was restricted to parametrising globally stabilising controllers for smooth nonlinear systems. Three specific directions to extend our theoretical results are as follows:

1. *Local stability:* In some applications, it can be undesirable (or infeasible) to have a globally contracting nonlinear system. For example, there may be multiple set points between which a controller should move the system. In this case, a version of the parametrisation that allows for local contraction guarantees and smoothly switching between equilibria would be a useful extension.

2. *Non-smooth systems:* One of the original motivations for this thesis was learning stabilising controllers for locomotion controllers in legged robotics. To extend the Youla-REN to such problems, one would need to study the parametrisation in the context of contraction analysis for limit cycles and hybrid systems [103, 162].

3. *Prescribed robustness:* Our theoretical results guarantee closed-loop contracting and Lipschitz responses for certain classes of nonlinear systems. However, they do not allow the user to specify the resulting Lipschitz bound of the closed-loop system. A natural extension would be to follow similar procedures to $\mathcal{H}_\infty$ design [42, 196] to devise a parametrisation of controllers achieving contracting and $\gamma$-Lipschitz closed loops for some user-defined value $\gamma \in \mathbb{R}^+$.

Perhaps the greatest difficulty in applying Youla-like policies (such as the Youla-REN) in practice is the need for an observer. The observer is central to our formulation of

the Youla parametrisation. For general nonlinear systems, it is not obvious how (or whether it is possible) to construct an observer that: (1) is contracting and Lipschitz; (2) satisfies the required correctness property; and (3) has minimal (preferably zero) Lipschitz bound from the Youla augmentation to the innovations. One interesting strategy would be to learn appropriate observers directly from data such that they meet these three criteria. An approach to learning observers with RENs is outlined in [136] (see also Section 6.6.2 with R2DNs and Appendix B).

Once a suitable base controller and observer have been designed, implementing the Youla-REN is straightforward – simply connect the REN (or other robust neural network, e.g., R2DN) to the difference between the measurements and observer estimates, and add its control output to the base controller. This general structure opens possibilities of learning stabilising controllers for rather complex nonlinear tasks. Two interesting applications include trajectory-tracking in aerial robotics with highly-dynamic manoeuvres (which can be modelled as a smooth nonlinear problem, see [127] for early work), and vision-based feedback control, where the observer model may depend on a learned model of the environment allowing for novel view synthesis [110, 86].

### Scaling Up R2DN

In order to learn controllers for vision-based feedback control, further development of R2DNs will be necessary. The parametrisation of R2DNs from Chapter 6 is still in its infancy. In future work, we plan to extend the parametrisation to general $(Q, S, R)$-dissipative models with non-zero direct feedthrough, allowing for efficient network implementations with more complex robustness constraints than the Lipschitz bound. We also plan to encode scalable parametrisations of contracting and $(Q, S, R)$-dissipative LTI systems into the architecture, rather than relying on dense LTI systems whose sizes increase quadratically with the number of internal states. This will allow for further improvements in the scalability of R2DNs and, ultimately, will provide a parametrisation of robust neural networks that is suitable for learning-based control with stability guarantees in complex, high-dimensional environments.

# Appendix A

# Additional Proofs

## A.1 Contracting Systems with Inputs

This appendix contains a brief study of contracting systems with inputs, including proofs of Lemmas 2.1 and 2.2 from Section 2.2.1. We review contraction analysis from the perspective of Riemannian geometry, provide proofs of two lemmas describing the response of contracting systems to bounded, additive disturbances, and use these results to prove Lemmas 2.1 and 2.2.

### A.1.1 An Introduction to Contraction Metrics

We first recall some useful facts about contracting systems. Contraction analysis is typically conducted by examining shrinking distances between state trajectories [98]. In general, we are interested in the contraction of geodesics between trajectories in some metric space, where distances are quantified by a (possibly state- and time-dependent) Riemannian metric called the contraction metric.

Consider a smooth, time-varying, contracting nonlinear system with state $x_t \in \mathbb{R}^n$:

$$x_{t+1} = f(x_t, t). \tag{A.1}$$

As a result of [169, Thm. 15], there exists a contraction metric $M(x,t) : \mathbb{R}^n \times \mathbb{N} \to \mathbb{R}^{n\times n}$ with

$$c_1 I \preceq M(x,t) \preceq c_2 I, \quad \forall x \in \mathbb{R}^n \tag{A.2}$$

for some $c_2 \geq c_1 > 0$, such that the following inequality holds for all $t \in \mathbb{N}$:

$$\left(\frac{\partial f}{\partial x_t}\right)^\top M(x_{t+1}, t+1)\frac{\partial f}{\partial x_t} \preceq \alpha^2 M(x_t, t) \tag{A.3}$$

for some $\alpha \in [0,1)$. The contraction metric is a Riemannian metric defining distances between state trajectories. Let $x^a, x^b$ be two trajectories of (A.1). For any frozen time $t$, the contraction metric $M(x_t,t)$ induces a Riemannian distance between $x_t^a, x_t^b$, or

$$V\left(x_t^a, x_t^b, t\right) := \inf_{\xi} \int_0^1 |\Theta(\xi(s),t)\partial_s \xi|\, ds, \tag{A.4}$$

with $\Theta(x,t) \in \mathbb{R}^{n\times n}$ satisfying $M(x,t) = \Theta(x,t)^\top \Theta(x,t)$, where $\xi : [0,1] \to \mathbb{R}^n$ is a smooth path defined such that $\xi(0) = x_t^a$ and $\xi(1) = x_t^b$. The infimum $\xi^\star$ is a geodesic connecting $x_t^a$ to $x_t^b$. From (A.2) we have

$$\sqrt{c_1}\left|x_t^a - x_t^b\right| \leq V\left(x_t^a, x_t^b, t\right) \leq \sqrt{c_2}\left|x_t^a - x_t^b\right|. \tag{A.5}$$

Furthermore, (A.3) implies that the distance $V$ between $x^a, x^b$ is always decreasing with rate $\alpha$, or

$$V\left(x_{t+1}^a, x_{t+1}^b, t+1\right) \leq \alpha V\left(x_t^a, x_t^b, t\right). \tag{A.6}$$

Together with (A.5), this implies that the system (A.1) is contacting with rate $\alpha$ and overshoot $\beta = \sqrt{c_2/c_1}$ in the sense of Definition 2.1.

### A.1.2 Contracting Systems Under Additive Disturbances

Let us now consider the system (A.1) with an additive disturbance $d(x_t, t) \in \mathbb{R}^n$:

$$x_t = f(x_t, t) + d(x_t, t). \tag{A.7}$$

Let $x, x^\star$ be the perturbed and unperturbed trajectories, respectively. The bound estimation of $|x_t - x_t^\star|$ under $|d(x,t)| \leq \bar{d}$ with $\bar{d} \in \mathbb{R}^+$ was given in [98], [173, Thm. 2.8]. Here we are interested in two additional cases: exponentially-decaying disturbances, and bounds on the $\ell_p$-norm $\|x_t - x_t^\star\|_p$ of the state difference. Note that a similar statement to Lemma A.1 for continuous-time systems can be found in [157, Thm. 4].

**Lemma A.1.** *Suppose that* (A.1) *is contracting with rate* $\alpha$ *and overshoot* $\beta$*. If the disturbance satisfies* $|d(x_t, t)| \leq \bar{d}\varepsilon^t$ *with* $\bar{d} \in \mathbb{R}^+$ *and* $\varepsilon \in [0, 1)$*, then the states* $x, x^\star$ *of the disturbed and undisturbed systems* (A.7), (A.1) *(respectively) satisfy*

$$|x_t - x_t^\star| \leq \beta(|x_0 - x_0^\star| + \bar{d}\bar{C})\rho^t, \quad \forall t \in \mathbb{N} \tag{A.8}$$

*for some* $\rho \in [0, 1)$ *and* $\bar{C} \in \mathbb{R}^+$ *depending on* $\alpha, \varepsilon$*.*

*Proof.* The proof follows a similar strategy to that of [173, Thm. 2.8]. For any time $t$, let $\xi_t : [0, 1] \to \mathbb{R}^n$ be a geodesic connecting $x_t^\star$ and $x_t$ with respect to the contraction metric $M(x, t)$. We then define the path $\xi_{t+1} : [0, 1] \to \mathbb{R}^n$ by

$$\xi_{t+1}(s) = f(\xi_t(s), t) + sd(x_t, t). \tag{A.9}$$

Note that $\xi_{t+1}$ is a smooth path connecting $x_{t+1}^\star$ and $x_{t+1}$ as

$$\xi_{t+1}(0) = f(\xi_t(0), t) = f(x_t^\star, t) = x_{t+1}^\star,$$
$$\xi_{t+1}(1) = f(\xi_t(1), t) + d(x_t, t) = f(x_t, t) + d(x_t, t) = x_{t+1}.$$

By taking the derivative of (A.9) with respect to $s$, we obtain

$$\partial_s \xi_{t+1} = \frac{\partial f}{\partial \xi_t} \partial_s \xi_t + d(x_t, t). \tag{A.10}$$

To simplify notation, we denote $d(x_t, t)$, $V(x_t^\star, x_t, t)$, and $\Theta(x, t)$ by $d_t, V_t$ and $\Theta_t(x)$, respectively. We can then make the following calculation.

$$\begin{aligned}
V_{t+1} &\leq \int_0^1 |\Theta_{t+1}(\xi_{t+1}(s)) \partial_s \xi_{t+1}| \, ds \\
&= \int_0^1 \left| \Theta_{t+1}(\xi_{t+1}(s)) \left( \frac{\partial f}{\partial \xi_t} \partial_s \xi_t + d_t \right) \right| ds \\
&\leq \int_0^1 \left| \Theta_{t+1}(\xi_{t+1}(s)) \frac{\partial f}{\partial \xi_t} \partial_s \xi_t \right| ds \\
&\quad + |d(x_t, t)| \int_0^1 \|\Theta_{t+1}(\xi_{t+1}(s))\| ds \\
&\leq \alpha \int_0^1 |\Theta_t(\xi_t(s)) \partial_s \xi_t| \, ds + \sqrt{c_2} |d(x_t, t)| \\
&= \alpha V_t + \sqrt{c_2} |d(x_t, t)|.
\end{aligned} \tag{A.11}$$

Line 1 is due to (A.4) since $\xi_{t+1}$ is not necessary a geodesic, although $\xi_t$ is. Lines 2–3 follow by (A.10) and the triangle inequality. Note that $\|\Theta\|$ in Line 3 denotes the matrix 2-norm of $\Theta$, which satisfies $\|\Theta(x, t)\| \leq \sqrt{c_2}$ for all $x, t$ due to (A.2). Line 4 is due to (A.3) while Line 5 comes from the assumption that $\xi_t$ is a geodesic.

By recursively applying (A.11), we obtain

$$V_t \leq \alpha^t V_0 + \sqrt{c_2} \sum_{k=0}^{t-1} \alpha^{t-1-k} |d_k| \tag{A.12}$$

and hence

$$V_t \leq \alpha^t V_0 + \bar{d} \sqrt{c_2} \sum_{k=0}^{t-1} \alpha^{t-1-k} \varepsilon^k$$

with $t \geq 1$, since the disturbance is exponentially decaying by assumption. From (A.2)

we have

$$\begin{aligned} |x_t - x_t^\star| &\leq \sqrt{c_2/c_1}\left(|x_0 - x_0^\star|\alpha^t + \bar{d}\sum_{k=0}^{t-1}\alpha^{t-1-k}\varepsilon^k\right) \\ &\leq \beta\left(|x_0 - x_0^\star| + \bar{d}\bar{C}\right)\rho^t. \end{aligned} \tag{A.13}$$

The values of the constants $\bar{C}$ and $\rho \geq \alpha$ depend on the relative values of $\alpha, \varepsilon$. When $\alpha \neq \varepsilon$, we have $\sum_{k=0}^{t-1}\alpha^{t-1-k}\varepsilon^k = (\alpha^t - \varepsilon^t)/(\alpha - \varepsilon) \leq \bar{C}\rho^t$ with $\bar{C} = 1/|\alpha - \varepsilon|$ and $\rho = \max(\alpha, \varepsilon)$. When $\alpha = \varepsilon$, we have $\sum_{k=0}^{t-1}\alpha^{t-1-k}\varepsilon^k = t\alpha^{t-1} \leq \bar{C}\rho^t$ with any $\rho \in (\alpha, 1)$ and

$$\bar{C} = \max_{t\in\mathbb{N}} \ \alpha t\left(\frac{\alpha}{\rho}\right)^t$$

which is well-defined as $\alpha < \rho$. □

**Lemma A.2.** *Suppose that* (A.1) *is contracting with rate* $\alpha$ *and overshoot* $\beta$*. Then the states* $x, x^\star$ *of the disturbed and undisturbed systems* (A.7), (A.1) *(respectively) satisfy*

$$\|x - x^\star\|_p \leq \beta C|x_0 - x_0^\star| + \frac{\beta}{1-\alpha}\|d\|_p \tag{A.14}$$

*for* $p \in [1, \infty)$*, where* $C \in \mathbb{R}$ *is a constant that depends on the contraction rate* $\alpha$*.*

*Proof.* The proof is a discrete-time version of [173, Thm. 2.6] combined with [87, Thm. 5.1]. Follow the same proof strategy as for Lemma A.1 to arrive at (A.12). We then divide the right-hand side of (A.12) into two components

$$\begin{aligned} v_1(t) &= \alpha^t V_0 \\ v_2(t) &= \sum_{k=0}^{t} |d_k|\alpha^{t-k} \end{aligned}$$

and analyse their $\ell_p$ norms separately, noting that $V_\ell(t) \leq v_1(t) + \sqrt{c_2}v_2(t-1)$. First

observe that $\|v_1\|_p^p = V_\ell(0)^p \sum_{k=0}^{\infty} \alpha^{kp}$ so

$$\|v_1\|_p \leq C V_\ell(0) \quad \text{where} \quad C = \begin{cases} 1 & \text{if } p = \infty \\ \frac{1}{(1-\alpha^p)^{1/p}} & \text{if } p \in [1, \infty) \end{cases}. \tag{A.15}$$

Next consider $\|v_2\|_{p,T}$ for some $T \in \mathbb{N}$. We will show that for each case of $p = \infty$, $p = 1$, and $p \in (1, \infty)$, $\|v_2\|_{p,T}$ is independent of $T$ and is bounded by $\|d\|_{p,T}$. For $p = \infty$, observe that $|v_2(t)| \leq \bar{d} \sum_{k=0}^{t} \alpha^{t-k} \leq \bar{d}/(1-\alpha)$ where $\bar{d} := \sup_{x_t, t} d_t(x_t, t)$. Hence

$$\|v_2\|_{\infty,T} \leq \frac{1}{1-\alpha} \|d\|_{\infty,T}.$$

When $p = 1$, we have that for $t \leq T < \infty$, $\|v_2\|_{1,T} \leq \sum_{t=0}^{T} \sum_{k=0}^{t} \alpha^{t-k} |d_k|$. Reversing the order of summations gives

$$\|v_2\|_{1,T} \leq \sum_{k=0}^{T} \sum_{t=k}^{T} \alpha^{t-k} |d_k| = \sum_{k=0}^{T} |d_k| \sum_{t=k}^{T} \alpha^{t-k} \leq \sum_{k=0}^{T} \frac{|d_k|}{1-\alpha} = \frac{1}{1-\alpha} \|d\|_{1,T}.$$

Finally, consider $p \in (1, \infty)$. Let $m \in (1, \infty)$ be defined by $1/p + 1/m = 1$. Then for $t \leq T < \infty$ we have

$$|v_2(t)| \leq \sum_{k=0}^{t} \alpha^{t-k} |d_k| = \sum_{k=0}^{t} (\alpha^{t-k})^{\frac{1}{m}} (\alpha^{t-k})^{\frac{1}{p}} |d_k| \leq \left( \sum_{k=0}^{t} \alpha^{t-k} \right)^{\frac{1}{m}} \left( \sum_{k=0}^{t} \alpha^{t-k} |d_k|^p \right)^{\frac{1}{p}}$$

where the last inequality follows by Hölder's inequality [82, Sec. 2.2]. Hence we have

$$|v_2(t)| \leq \frac{1}{(1-\alpha)^{\frac{1}{m}}} \left( \sum_{k=0}^{t} \alpha^{t-k} |d_k|^p \right)^{\frac{1}{p}}.$$

Substituting this into the definition of the $\ell_p$-norm for $v_2$ gives

$$\begin{aligned}\|v_2\|_{p,T}^p &\leq \sum_{t=0}^{T} \frac{1}{(1-\alpha)^{\frac{p}{m}}} \sum_{k=0}^{t} \alpha^{t-k}|d_k|^p \\ &= \frac{1}{(1-\alpha)^{\frac{p}{m}}} \sum_{t=0}^{T}\sum_{k=0}^{t} \alpha^{t-k}|d_k|^p \\ &= \frac{1}{(1-\alpha)^{\frac{p}{m}}} \sum_{k=0}^{T} |d_k|^p \sum_{t=k}^{t} \alpha^{t-k} \\ &\leq \frac{1}{(1-\alpha)^{\frac{p}{m}+1}} \|d\|_{p,T}^p.\end{aligned}$$

Since $\frac{p}{m}+1 = p$ by construction, then $\|v_2\|_{p,T} \leq \frac{1}{1-\alpha}\|d\|_{p,T}$ for $p \in (1,\infty)$. Combining this with our results for $p = 1, \infty$ and (A.15) and applying the triangle inequality, we have that

$$\|V\|_p \leq CV(0) + \frac{\sqrt{c_2}}{1-\alpha}\|d\|_p \quad \forall\, p \in [1,\infty) \tag{A.16}$$

with $C$ as defined in (A.15). Applying the upper and lower bounds on $V_t$ from (A.5), we conclude that

$$\|x - x^\star\|_p \leq \beta C|x_0 - x_0^\star| + \frac{\beta}{1-\alpha}\|d\|_p \tag{A.17}$$

with $\beta = \sqrt{c_2/c_1}$ as usual. □

### A.1.3 Proofs of Lemmas 2.1 and 2.2

*Both combined.* Lemmas 2.1 and 2.2 follow directly from Lemmas A.1 and A.2 (respectively) by noting the following. Write (2.8) for $x, x^\star$ as

$$\begin{aligned}x_{t+1} &= F(x_t, u_t) \\ x_{t+1}^\star &= F(x_t^\star, u_t) + \Delta F_t\end{aligned}$$

where $\Delta F_t := F(x_t^\star, u_t^\star) - F(x_t^\star, u)$. Since $F$ is assumed to be $\gamma$-Lipschitz with respect to $u$, then $|\Delta F_t| \leq \gamma|u_t - u_t^\star|$. The results (2.10), (2.11) follow from (A.14), (A.8) by substituting $\|d\|_p \leq \gamma\|u - u^\star\|_p$ and $|d_t| \leq \gamma|u_t - u_t^\star|$, respectively. □

# Appendix B

## `RobustNeuralNetworks.jl`

We have used robust neural networks throughout this thesis for tasks in machine learning and data-driven control with certified stability and robustness. This appendix contains partial documentation for `RobustNeuralNetworks.jl`[1] – a Julia package containing implementations of the REN and Sandwich networks introduced in [136] and [181], respectively. The package relies heavily on key features of the Julia language [19] (such as multiple dispatch) for an efficient implementation of the models. The purpose of the package was to make recent research in robust machine learning at the ACFR easily accessible to users in the scientific and machine learning communities. We therefore designed it to interface directly with `Flux.jl` [76], Julia's most widely-used machine learning library, making it straightforward to incorporate robust neural networks into existing Julia code. Detailed documentation is available on GitHub[2].

`RobustNeuralNetworks.jl` was the first attempt at creating a single, unified repository for the collection of robust neural network architectures developed at the ACFR. We are currently developing a successor to the package written in Python/JAX [22] which includes not only RENs and Sandwich networks, but also newer architectures such as monotone, bi-Lipschitz, and Polyak-Łojasiewicz Networks [179] and invertible

[1] https://github.com/acfr/RobustNeuralNetworks.jl
[2] https://acfr.github.io/RobustNeuralNetworks.jl/stable/

dynamic models such as the BiLipREN [195]. An early version of the Python package is now publicly available[3]. For the remainder of this chapter, we focus on the Julia package, providing an overview of the software layout and tutorials demonstrating its use in image classification, reinforcement learning, and nonlinear observer design.

### Publications & Usage

The content in this chapter previously appeared as part of the following publication:

> [11] N. H. Barbara, M. Revay, R. Wang, J. Cheng, and I. R. Manchester, "Robust-neuralnetworks.jl: a package for machine learning and data-driven control with certified robustness," *Proceedings of the JuliaCon Conferences,* 2025.

The `RobustNeuralNetworks.jl` package was used for all experiments in Chapter 5 and [12], while earlier versions of the codebase were also used in [178, 136, 124].

## B.1 Package Overview

`RobustNeuralNetwork.jl` contains two classes of neural network models – RENs [136] and Sandwich networks [181]. Both network architectures were discussed in Section 2.3. This section provides a brief overview of how the two model architectures and their parametrisations were implemented in Julia. In the remaining sections, we use LBDN to refer to Lipschitz-bounded networks constructed specifically from Sandwich layers to remain consistent with the package implementation.

### B.1.1 Direct Parametrisations

Recall the notion of a direct parametrisation from Definition 2.4 – that is, a differentiable mapping $\varphi = \mathcal{M}(\theta)$ from the learnable or *direct parameters* $\theta \in \mathbb{R}^N$ of a neural network to its *explicit parameters* $\varphi \in \mathbb{R}^{n_\varphi}$, such as its weights and biases. When we implement direct parametrisations in practice, it is important to separately track the

[3] https://github.com/acfr/RobustNeuralNetworks

direct and explicit parameters $\theta$ and $\varphi$, respectively. We do this as follows.

RENs are defined by two abstract types in `RobustNeuralNetworks.jl`. Subtypes of `AbstractRENParams` hold all the information required to directly parametrise a REN with particular stability and/or robustness properties. For example, to initialise the direct parameters of a contracting REN with 1 input, 10 states, 20 neurons, 1 output, and a `relu` activation function, we use the following. The direct parameters $\theta$ are stored in `model_ps.direct`.

```
using Flux, RobustNeuralNetworks

T  = Float32
nu, nx, nv, ny = 1, 10, 20, 1
model_ps = ContractingRENParams{T}(nu, nx, nv, ny; nl=Flux.relu)
println(model_ps.direct) # Access direct params
```

Subtypes of `AbstractREN` represent RENs in their explicit form which can be evaluated on data. The conversion from direct to explicit parameters $\theta \mapsto \varphi$ is performed when the REN is constructed and the explicit parameters $\varphi$ are stored in `model.explicit`.

```
model = REN(model_ps)    # Create explicit model
println(model.explicit)  # Access explicit params
```

Figure B.1 illustrates this architecture. We use a similar interface for the Lipschitz-bounded Sandwich networks, basing them off the abstract types `AbstractLBDNParams` and `AbstractLBDN`.

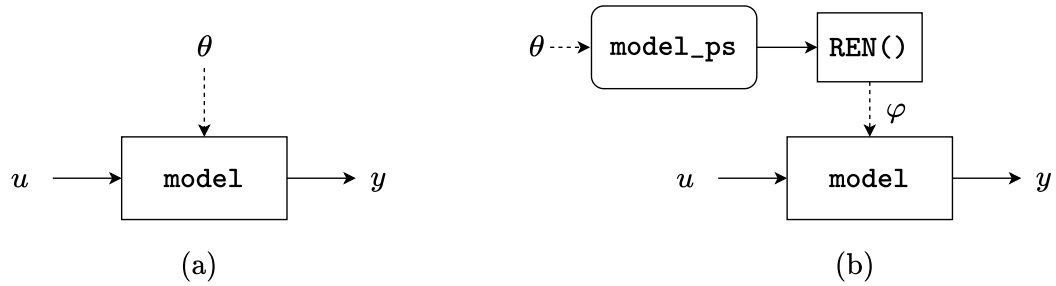



**Figure B.1:** Association of models and their parameters in (a) `Flux.jl` and (b) `RobustNeuralNetworks.jl`. In (a), model parameters $\theta$ are associated with the `model`. In (b), the direct parameters $\theta$ are associated with the parametrisation `model_ps`, and are converted to explicit parameters $\varphi$ when the `model` is constructed for evaluation with `REN()`.

There are currently four REN parametrisations implemented in this package:

1. `ContractingRENParams` parametrises contracting RENs with a user-defined upper bound on the contraction rate, as per Definition 2.1.

2. `LipschitzRENParams` parametrises RENs with a user-defined (or learnable) Lipschitz bound $\gamma \in (0, \infty)$, as per Definition 2.3

3. `PassiveRENParams` parametrises incrementally input passive RENs with user-tunable passivity parameter $\nu \geq 0$ (see below).

4. `GeneralRENParams` parametrises RENs satisfying some general behavioural constraints defined by an incremental IQC with parameters (Q,S,R), as per Definition 2.2

Note that a model is incrementally input passive if it admits the incremental IQC described by $Q = 0, R = -2\nu I, S = I$ for some $\nu \geq 0$. Passivity is a generalised notion of energy conservation [175] and is useful in, for example, learning stable dynamical systems [31].

There is currently only one LBDN parametrisation implemented in the package:

1. `DenseLBDNParams` parametrises dense (fully-connected) LBDNs with a user-defined or learnable Lipschitz bound $\gamma \in (0, \infty)$ using the Sandwich layer.

### B.1.2 Explicit Model Wrappers

When training a REN or LBDN, we learn and update the direct parameters $\theta$ and convert them to the explicit parameters $\varphi$ only for model evaluation. The main constructors for explicit models are `REN` and `LBDN`. Users familiar with `Flux.jl` will be used to creating a model once and then training it on their data. The typical workflow is as follows.

```
using Flux

# Define a model and a loss function
model = Flux.Chain(
    Flux.Dense(1 => 10, Flux.relu),
    Flux.Dense(10 => 1, Flux.relu)
)
loss(model, x, y) = Flux.mse(model(x), y)

# Training data of 20 batches
T = Float32
xs, ys = rand(T,1,20), rand(T,1,20)
data = [(xs, ys)]

# Train the model for 50 epochs
opt_state = Flux.setup(Adam(0.01), model)
for _ in 1:50
    Flux.train!(loss, model, data, opt_state)
end
```

When training a model constructed from `REN` or `LBDN`, we need to back-propagate through the mapping from direct (learnable) parameters to the explicit model. We

must therefore include the model construction as part of the loss function. If we do not, then the auto-differentiation engine has no knowledge of how the model parameters affect the loss, and will return zero gradients. Here is an example with an `LBDN`, where the `model` is defined by the direct parametrisation stored in `model_ps`.

```
using Flux, RobustNeuralNetworks

# Define model parametrisation and loss function
T = Float32
model_ps = DenseLBDNParams{T}(1, [10], 1; nl=relu)
function loss(model_ps, x, y)
    model = LBDN(model_ps)
    Flux.mse(model(x), y)
end

# Training data of 20 batches
xs, ys = rand(T,1,20), rand(T,1,20)
data = [(xs, ys)]

# Train the model for 50 epochs
opt_state = Flux.setup(Adam(0.01), model_ps)
for _ in 1:50
    Flux.train!(loss, model_ps, data, opt_state)
end
```

### B.1.3 Separating Parameters and Models

For the sake of convenience, we include the model wrappers `DiffREN` and `DiffLBDN` as alternatives to `REN` and `LBDN`, respectively. These wrappers compute the explicit parameters each time the model is called rather than just once when they are con-

structed. Any model created with these wrappers can therefore be used exactly the same way as a regular `Flux.jl` model, and there is no need for model construction in the loss function. One can simply replace the definition of the `Flux.Chain` model in the `Flux.jl` example above with

```
model_ps = DenseLBDNParams{T}(1, [10], 1; nl=relu)
model = DiffLBDN(model_ps)
```

and train the LBDN just like any other `Flux.jl` model. We use these wrappers in Appendices B.2.1 and B.2.3.

The trade-off in using `DiffREN` or `DiffLBDN` is computational efficiency in applications where a model is called many times before a training update (e.g., system identification and reinforcement learning). One of the main computational bottlenecks in training a REN or LBDN is converting from the direct to explicit parameters (mapping $\theta \mapsto \varphi$). This process involves a matrix inverse where the number of matrix elements scales quadratically with the dimension of the model in a REN or the dimension of each layer in an LBDN (see [136, 181]).

If a model is to be evaluated many times with the same direct parameters in between training updates, it is more efficient to compute the explicit parameters once, hold them fixed over many model calls, and only re-compute them once the direct parameters have been updated. This is exactly the purpose of keeping `model_ps` and `model` separate when using `REN` and `LBDN`. Note that we cannot store the direct and explicit parameters in the same `model` object since auto-differentiation in Julia does not (at the time of writing) support array mutation [75]. We therefore advise using `DiffREN` or `DiffLBDN` for convenience in applications where the model parameters are updated after just one model call (e.g., training an image classifier). The computational benefit of separating models from their parametrisations is explored numerically in Appendix B.2.2.

## B.2 Examples

This section guides the reader through a set of examples to demonstrate how to use `RobustNeuralnetworks.jl` for machine learning in Julia. We will consider three examples: image classification, reinforcement learning, and nonlinear observer design. These examples illustrate the benefits of using robust models and the reasoning behind key design decisions made in the development of the package.

We use `relu` activation functions in all examples, but other choices of activation functions satisfying Assumption 2.1 (e.g: `tanh`) are equally valid. For more examples with RENs and LBDNs , please see the package documentation[2].

### B.2.1 Image Classification

Our first example features an LBDN trained to classify the MNIST dataset [94]. We use this example to demonstrate how training image classifiers with LBDNs makes them robust to noise (and adversarial attacks) thanks to the built-in Lipschitz bound. For a detailed investigation of the effect of Lipschitz bounds on image classification robustness, see [181].

#### Loading the data

We begin by loading the training and test data from `MLDatasets.jl`[4]. To load the full dataset of 60,000 training and 10,000 test images, one would run the following code.

```
using MLDatasets: MNIST

T = Float32
x_train, y_train = MNIST(T, split=:train)[:]
x_test,  y_test  = MNIST(T, split=:test)[:]
```

[4] `https://juliaml.github.io/MLDatasets.jl/`

The feature matrices `x_train` and `x_test` are three-dimensional arrays where each $28 \times 28$ layer contains pixel data for a single handwritten number from 0 to 9 (see Figure B.2). The labels `y_train` and `y_test` are vectors containing the classification of each image as a number from 0 to 9. We convert each of these to a format better suited to training with `Flux.jl`.

```
using Flux

# Reshape features for model input
x_train = Flux.flatten(x_train)
x_test  = Flux.flatten(x_test)

# Encode categorical outputs and store
y_train = Flux.onehotbatch(y_train, 0:9)
y_test  = Flux.onehotbatch(y_test,  0:9)
data = [(x_train, y_train)]
```

Features are now stored in a $28^2 \times N$ `Matrix` where each column contains pixel data from a single image, and the labels have been converted to a $10 \times N$ `Flux.OneHotMatrix` where each column contains a 1 in the row corresponding to the image's classification (e.g., row 3 for an image showing the number 2) and a 0 otherwise.

### Defining a model

We now construct an LBDN model. The larger the model, the better the classification accuracy will be, at the cost of longer training times. The smaller the Lipschitz bound $\gamma$, the more robust the model will be to input perturbations such as image noise. If $\gamma$ is too small, however, it can restrict the model flexibility and limit the achievable performance (e.g., [181] and Chapter 3). For this example, we use a small network of two 64-neuron hidden layers and set a Lipschitz bound of $\gamma = 5.0$.

```
using RobustNeuralNetworks

# Model specification
nu = 28*28              # Inputs (size of image)
ny = 10                 # Outputs (classifications)
nh = fill(64,2)         # Hidden layers
γ  = 5.0f0              # Lipschitz bound 5.0

# Define parameters,create model
model_ps = DenseLBDNParams{T}(nu, nh, ny, γ)
model = Chain(DiffLBDN(model_ps), Flux.softmax)
```

The `model` consists of two parts. The first is a callable `DiffLBDN` model constructed from its direct parametrisation, which is defined by an instance of `DenseLBDNParams` as per Appendix B.1.1. The output is then converted to a probability distribution using a `softmax` layer. Note that all `AbstractLBDN` models can be combined with traditional neural network layers using `Flux.Chain`.

We could also construct the `model` as a chain of `SandwichFC` layers. We have designed the user interface for `SandwichFC` similarly to that of `Flux.Dense`.

```
model = Chain(
    (x) -> (sqrt(γ) * x),
    SandwichFC(nu => nh[1], relu; T),
    SandwichFC(nh[1] => nh[2], relu; T),
    SandwichFC(nh[2] => ny; output_layer=true, T),
    (x) -> (sqrt(γ) * x),
    Flux.softmax
)
```

This model is equivalent to a dense LBDN constructed with `LBDN` or `DiffLBDN`. We have included it as a convenience for users familiar with layer-wise network construction in `Flux.jl`, and recommend using it interchangeably with `DiffLBDN`.

### Defining a loss function

A typical loss function for training on datasets with discrete labels is the cross entropy loss. We can use the `crossentropy` loss function shipped with `Flux.jl`.

```
loss(model,x,y) = Flux.crossentropy(model(x), y)
```

### Training the model

We train the model over 600 epochs using a learning rate of $10^{-3}$ for the first 300 epochs, and $10^{-4}$ for the remaining 300 epochs. We use the Adam optimiser [88] and the default `Flux.train!` method for convenience. Note that the `Flux.train!` method updates the learnable parameters each time the model is evaluated on a batch of data, hence our choice of `DiffLBDN` as a model wrapper.

```
# Hyperparameters
epochs = 300
learning_rates = [1e-3,1e-4]

# Train with the Adam optimiser
opt_state = Flux.setup(Adam(learning_rates[1]), model)
for k in eachindex(learning_rates)
    for i in 1:epochs
        Flux.train!(loss, model, data, opt_state)
    end
    Flux.adjust!(opt_state, learning_rates[2])
end
```

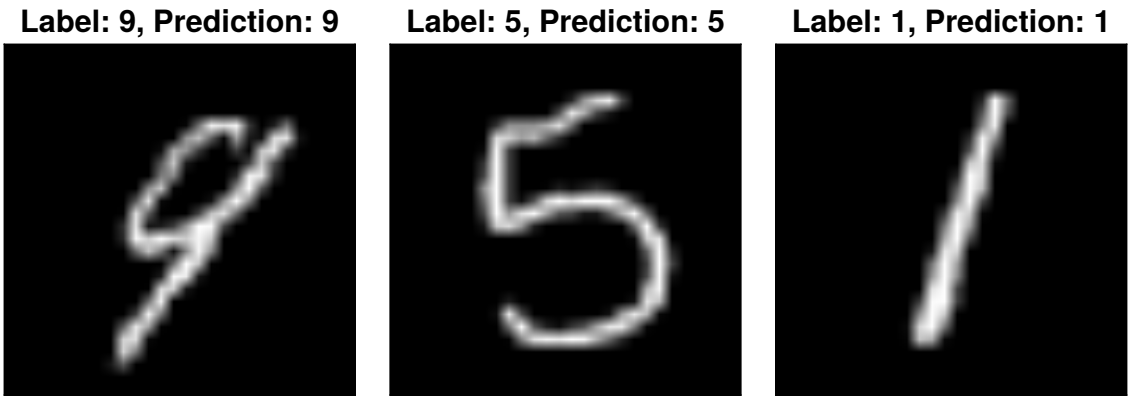



**Figure B.2:** Prediction examples from the trained LBDN model on the MNIST dataset.

### Evaluating the trained model

Our final model achieves training and test accuracies of approximately 98% and 97%, respectively, as shown in Table B.1. We could improve this further by switching to a convolutional LBDN, as in [181]. Some examples of classifications given by the trained LBDN model are presented in Figure B.2.

### Investigating robustness

The main advantage of using an LBDN for image classification is its built-in robustness to perturbations. This robustness is a direct benefit of the Lipschitz bound, which we saw in the context of deep RL in Chapter 3. The Lipschitz bound effectively defines how smooth the network is: the smaller the bound, the less the network outputs will change as the inputs vary. For example, small amounts of noise added to the image will be less likely to change its classification.

We can demonstrate the robustness of LBDNs by comparing the model to a standard

| **Model structure** | **Training accuracy (%)** | **Test accuracy (%)** |
|---|---|---|
| LBDN | 98.2 | 97.2 |
| Dense | 97.6 | 96.6 |

**Table B.1:** Training and test accuracy for the LBDN and Dense models on the MNIST dataset without perturbations.

MLP built from `Flux.Dense` layers. We first create a `dense` network with the same layer structure as the LBDN.

```
# Initialisation functions
init = Flux.glorot_normal
initb(n) = Flux.glorot_normal(n)

# Build a dense model
dense = Chain(
    Dense(nu => nh[1], relu; init, bias=initb(nh[1])),
    Dense(nh[1] => nh[2], relu; init, bias=initb(nh[2])),
    Dense(nh[2] => ny; init, bias=initb(ny)),
    Flux.softmax
)
```

Training the `dense` model with the same training loop used for the LBDN results in a model that achieves training and test accuracies of approximately 98% and 97%, respectively, as shown in Table B.1.

As a simple test of robustness, we add uniformly-sampled random noise in the range $[-\epsilon, \epsilon]$ to the test dataset for a range of noise magnitudes $\epsilon \in [0, 200/255]$. We record the test accuracy for each perturbation size and store it for plotting.

```
using Statistics

# Get test accuracy as we add noise
uniform(x) = 2*rand(T, size(x)...) .- 1
compare(y, yh) = maximum(yh, dims=1) .== maximum(y.*yh, dims=1)
accuracy(model, x, y) = mean(compare(y, model(x)))
function noisy_test_error(model, ϵ)
```

```
    noisy_xtest = x_test .+ ϵ*uniform(x_test)
    accuracy(model, noisy_xtest,  y_test)*100
end

ϵs = T.(LinRange(0, 200, 10)) ./ 255
lbdn_error  = noisy_test_error.((model,), ϵs)
dense_error = noisy_test_error.((dense,), ϵs)
```

Plotting the results in Figure B.3 clearly shows that the `dense` network, which has no guarantees on its Lipschitz bound, quickly loses its accuracy as small amounts of noise are added to the image. In contrast, the LBDN `model` maintains its accuracy even when the (maximum) perturbation size is as much as 80% of the maximum pixel values. This is an illustration of why image classification is such a promising use-case for LBDN models. For a more detailed comparison of LBDN with state-of-the-art image classification methods and optimised adversarial attacks, see [181].

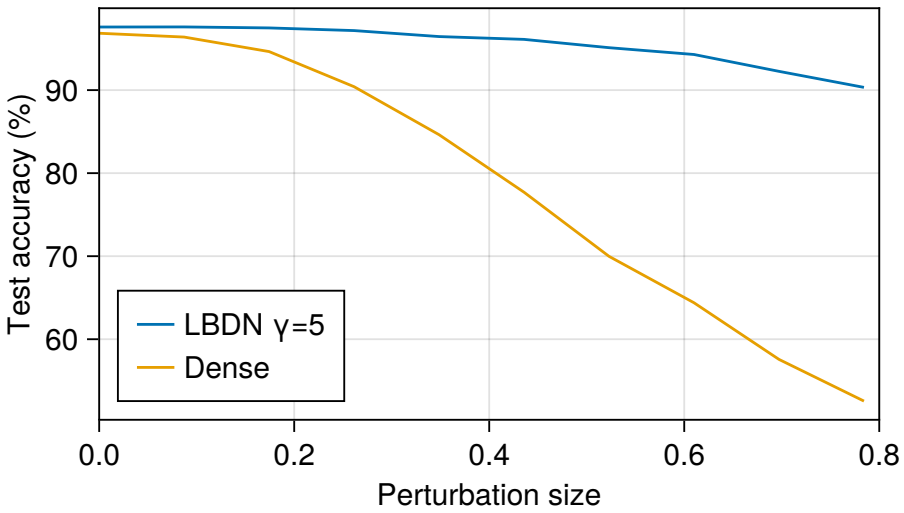



**Figure B.3:** Comparison of test accuracy on the MNIST dataset as a function of random perturbation magnitude $\epsilon$. The LBDN model is significantly more robust than a standard `Dense` network.

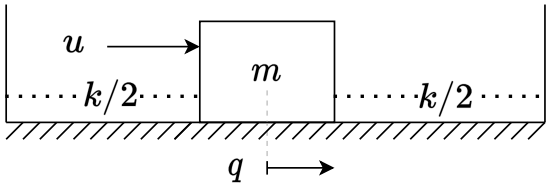



**Figure B.4:** Mechanical system to be controlled. A box sits in a tub of fluid, suspended between two springs with stiffness $k/2$. It can be pushed by a force $u$ to different positions $q$.

### B.2.2 Reinforcement Learning

In this example, we demonstrate how to train an LBDN controller with deep RL for a simple nonlinear dynamical system. This controller will not have any stability guarantees – i.e., it is in a standard feedback configuration like Chapter 3, not the Youla parametrisation from Chapters 4 and 5, and we are not applying the small-gain theorem. The purpose of this example is simply to illustrate the steps required to set up RL experiments with `RobustNeuralNetworks.jl`.

Consider the simple mechanical system shown in Figure B.4. A box of mass $m$ sits in a tub of fluid, held between the walls by two springs each with spring constant $k/2$. The box can be pushed with a force $u$. Its dynamics are

$$m\ddot{q} = u - kq - \mu\dot{q}|\dot{q}| \tag{B.1}$$

where $\mu$ is the viscous damping coefficient due to the box moving through the fluid, and $\dot{q}, \ddot{q}$ denote the velocity and acceleration of the box, respectively. We can write this as a state-space model with state $x = [q; \dot{q}]$, control input $u$, and dynamics

$$\dot{x} = f(x, u) := \begin{bmatrix} \dot{q} \\ (u - kq - \mu\dot{q}|\dot{q}|)/m \end{bmatrix}. \tag{B.2}$$

This is a continuous-time model of the dynamics. For our purposes, we need a discrete-time model. We can discretise the dynamics using a forward Euler approximation

$$x_{t+1} = f_d(x_t, u_t) := x_t + \Delta t \cdot f(x_t, u_t) \tag{B.3}$$

where $\Delta t$ is the time-step. This approximation typically requires a small time-step for numerical stability, but is sufficient for our simple example. If physical accuracy was of concern, one could use a higher-order Runge-Kutta scheme.

Our aim is to learn a controller $u = \mathcal{K}_\theta(x, q_{\text{ref}})$, parametrised by a vector of learnable parameters $\theta \in \mathbb{R}^N$, that can push the box to any goal state $x_{\text{ref}} = [q_{\text{ref}}; 0]$ within a time period $T \in \mathbb{N}$. The force required to keep the box at an equilibrium position $q_{\text{ref}}$ is $u_{\text{ref}} = kq_{\text{ref}}$. We encode these objectives into a cost function to write the RL problem

$$\min_\theta \mathbb{E}_{q_0, \dot{q}_0, q_{\text{ref}}} [J_\theta], \quad J_\theta = \sum_{t=0}^{T-1} c_1(\Delta q_t)^2 + c_2 \dot{q}_t^2 + c_3(\Delta u_t)^2, \tag{B.4}$$

where $\Delta q_t = q_t - q_{\text{ref}}$, $\Delta u_t = u_t - u_{\text{ref}}$, $c_1, c_2, c_3$ are cost function weights, and the expectation is over different initial and goal positions of the box.

### Problem setup

We first define the properties of our system. We consider a box of mass $m = 1$, spring constants $k = 5$, and a viscous damping coefficient $\mu = 0.5$. We will simulate the system over $T = 4\,\text{s}$ with a time-step of $\Delta t = 0.02\,\text{s}$.

```
m, k, μ = 1, 5, 0.5      # Mass (kg), spring (N/m), damper (kg/m)
Tmax, dt = 4, 0.02       # Simulation horizon and time step (s)
ts = 1:Int(Tmax/dt)      # Array of time indices
```

To generate the training data, we randomly sample 80 goal positions in the range $q_{\text{ref}} \in [-1, 1]$ and assume the box always starts at rest from the zero position.

```
nx, nref, batches = 2, 1, 80
x0 = zeros(nx, batches)
qref = 2*rand(nref, batches) .- 1
uref = k*qref
```

It is more efficient to simulate all batches at once, so we define our dynamics functions to operate on batches of states and controls. Each row corresponds to a different state or control, and each column corresponds to a simulation for a particular $q_{\text{ref}}$.

```
f(x::Matrix,u::Matrix) = [x[2:2,:]; (u[1:1,:] -
    k*x[1:1,:] - μ*x[2:2,:]*abs.(x[2:2,:]))/m]
fd(x::Matrix,u::Matrix) = x + dt*f(x,u)
```

We learn controllers via the APG algorithm (see Section 2.4.1). That is, we back-propagate directly through the cost and dynamics functions rather than approximating the cost gradient $\nabla J_\theta$. The simulator below takes a batch of initial states, goal positions, and a controller `model` whose inputs are $[x; q_{\text{ref}}]$. It computes trajectories of states and controls $z = \{[x_0; u_0], \dots, [x_{T-1}; u_{T-1}]\}$. To avoid the issue of unsupported array mutation when differentiating we use a `Zygote.Buffer` to store the outputs [75]. We conclude the problem setup with a function to evaluate the cost function in (B.4).

```
using Zygote: Buffer
using Statistics

function rollout(model, x0, qref)
    z = Buffer([zero([x0;qref])], length(ts))
    x = x0
    for t in ts
        u = model([x;qref])
        z[t] = vcat(x,u)
        x = fd(x,u)
    end
    return copy(z)
end
```

```
weights = [10,1,0.1]
function _cost(z, qref, uref)
    Δz = z .- [qref; zero(qref); uref]
    return mean(sum(weights .* Δz.^2; dims=1))
end
cost(z::AbstractVector, qref, uref) = mean(_cost.(z, (qref,), (uref,)))
```

### Defining a model

We will train an LBDN controller with a Lipschitz bound of $\gamma = 20$. Its inputs are the state $x_t$ and goal position $q_{\text{ref}}$, while its outputs are the control force $u_t$. We have chosen a model with two hidden layers each of 32 neurons just as an example.

```
using RobustNeuralNetworks

γ  = 20                 # Lipschitz bound
nu = nx + nref          # Inputs (x and reference)
ny = 1                  # Outputs (control action)
nh = fill(32, 2)        # Hidden layers
model_ps = DenseLBDNParams{Float64}(nu, nh, ny, γ)
```

### Defining a loss function

In constructing a loss function for this problem, we refer to Appendix B.1.2. The `model_ps` contain all information required to define a dense LBDN model. However, `model_ps` is not a model that can be evaluated on data: it is a *model parametrisation*, and contains the learnable parameters $\theta$. To train an LBDN given some data, we construct the model within the loss function using the `LBDN` wrapper so that the mapping from direct to explicit parameters is captured during back-propagation. Our loss function therefore includes the following three components.

```
function loss(model_ps, x0, qref, uref)
    model = LBDN(model_ps)            # Model
    z = rollout(model, x0, qref)      # Simulation
    return cost(z, qref, uref)        # Cost
end
```

### Training the model

Having set up the problem, all that remains is to train the policy. The function below trains a model and tracks the training loss `tl` (cost $J_\theta$) for each simulation in our batch of 80. We use the Adam optimiser over 250 epochs and a learning rate of $10^{-3}$.

```
using Flux

function train_box_ctrl!(model_ps, loss_func; epochs=250, lr=1e-3)
    costs = Vector{Float64}()
    opt_state = Flux.setup(Adam(lr), model_ps)
    for k in 1:epochs
        tl, dJ = Flux.withgradient(loss_func, model_ps, x0, qref, uref)
        Flux.update!(opt_state, model_ps, dJ[1])
        push!(costs, tl)
    end
    return costs
end
costs = train_box_ctrl!(model_ps, loss)
```

### Evaluating the trained model

We may now verify the performance of the trained model on a new set of reference positions. In the code below, we generate 60 batches of test data. In each one, the box

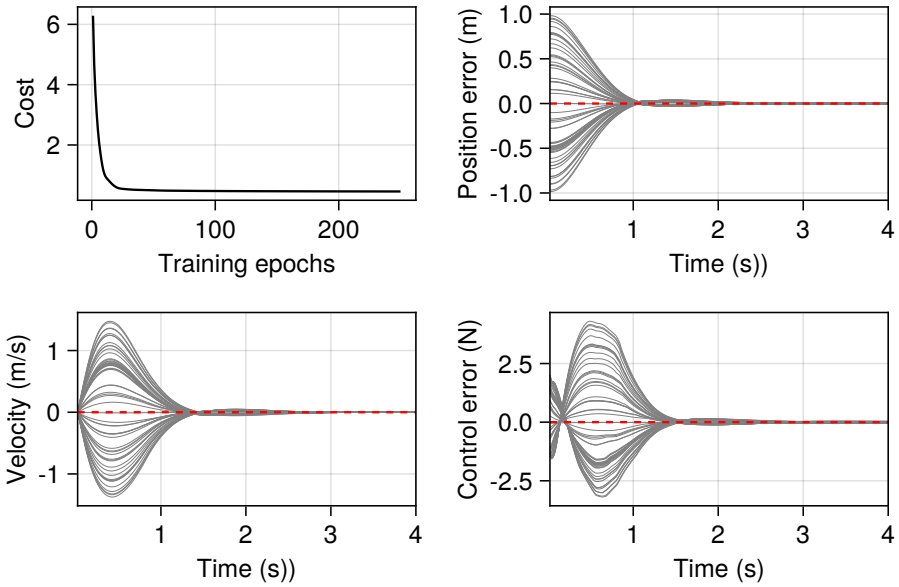



**Figure B.5:** Loss curve and simulation results from the LBDN policy controlling the box system in Figure B.4. The policy can push the box to any desired location in the domain of interest. The position and controller errors are $\Delta q$ and $\Delta u$ from (B.4), respectively.

starts at the origin at rest, and is moved through the fluid to a different (random) goal position $q_{\text{ref}} \in [-1, 1]$. We plot the states and controls alongside the loss curve from training in Figure B.5. The box clearly moves to the required position within the time frame in all cases. Plotting code is not included in the cell below.

```
model   = LBDN(model_ps)
x0_test = zeros(2,60)
qr_test = 2*rand(1, 60) .- 1
z_test  = rollout(model, x0_test, qr_test)
```

### Advantages of separate parameters and models

As discussed in Appendix B.1.3, there is a trade-off between convenience and performance in `RobustNeuralNetworks.jl`. The `DiffLBDN` and `DiffREN` wrappers exist to allow users to train robust models in a `Flux.jl`-like manner. These wrappers convert a model parametrisation to its explicit form each time they are called, hence the user

does *not* have to re-construct the model in the loss function.

```
loss2(model, x0, qref, uref) = cost(rollout(model,x0,qref), qref, uref)
```

The cost is computation speed, particular in an RL context. Careful inspection of the `rollout()` function shows that the `model` is evaluated many times within the loss function before the learnable parameters are updated with `Flux.update!()`. As discussed in the Appendix B.1.3, a computational bottleneck in training RENs and LBDNs is the conversion from learnable (direct) parameters to an explicit model. Constructing the model only when the parameters are updated therefore saves considerably on computation time, particularly for large models.

For example, suppose we train single-hidden-layer LBDNs with $n = 2, 2^2, \ldots, 2^9$ neurons over 100 epochs on our box RL problem, and log the time taken to train each model when using both `LBDN` and `DiffLBDN`.

```
function lbdn_compute_times(n; epochs=100)
    # Build model params and a model
    lbdn_ps = DenseLBDNParams{Float64}(nu, [n], ny, γ)
    diff_lbdn = DiffLBDN(deepcopy(lbdn_ps))

    # Time with LBDN vs DiffLBDN (respectively)
    t_lbdn = @elapsed (train_box_ctrl!(lbdn_ps, loss; epochs))
    t_diff_lbdn = @elapsed (train_box_ctrl!(diff_lbdn, loss2; epochs))
    return [t_lbdn, t_diff_lbdn]
end

# Evaluate computation time and run it once first for the compiler
lbdn_compute_times(2; epochs=1)
comp_times = reduce(hcat, lbdn_compute_times.(2 .^ (1:9)))
```

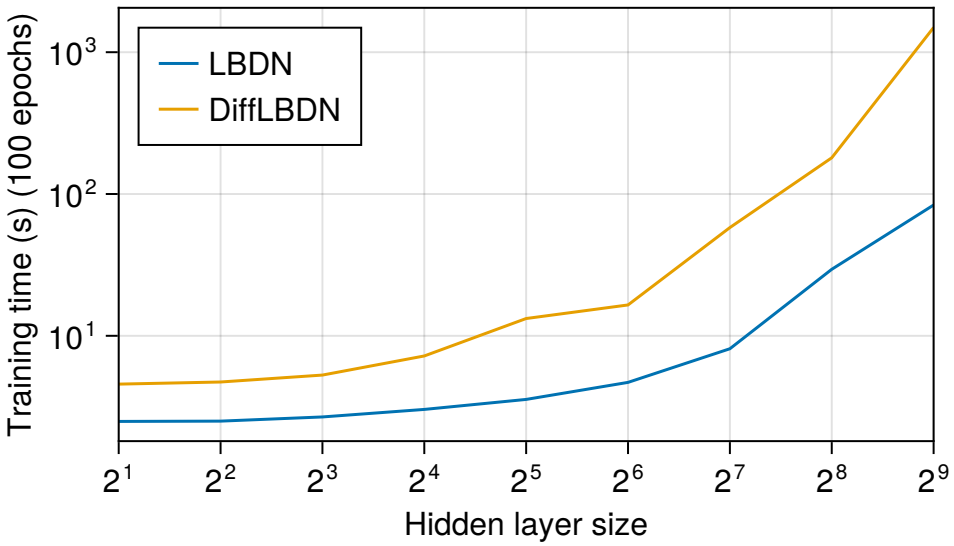



**Figure B.6:** Training time as a function of layer size for a single-hidden-layer LBDN constructed with the `LBDN` and `DiffLBDN` wrappers. Using the `LBDN` wrapper for RL is more efficient than re-constructing the explicit model at every evaluation of the `DiffLBDN` model.

The results are plotted in Figure B.6. For a single-layer LBDN with $2^9 = 512$ neurons, using `DiffLBDN` takes an order of magnitude longer to train than only constructing the `LBDN` model each time the `loss()` function is called. If we were training dynamic models with REN, the computational overhead of using `DiffREN` instead of `REN` would be even more extreme, since the conversion from direct to explicit parameters in a REN is typically more computationally expensive than for LBDNs . It is for this reason that we strongly recommend using the `LBDN` and `REN` wrappers if many evaluations of the model are required before `Flux.update!()` (or equivalent) is called, as in RL.

### B.2.3 Observer Design

In Appendix B.2.2, we designed a controller for a simple nonlinear system assuming that the controller had full state knowledge. In many practical situations, we may only be able to measure some of the system states. For example, our box may have a camera to estimate its position but not its velocity. In these cases, we need an observer to estimate the full state of the system. This example demonstrates how a contracting REN can be used to learn stable observers for dynamical systems.

We consider observer design as a supervised learning problem (Section 6.6.2). We briefly summarise some background theory from [136] relevant to this example. Suppose we have a discrete-time, nonlinear dynamical system of the form

$$x_{t+1} = f_d(x_t, u_t) \tag{B.5}$$

$$y_t = h_d(x_t, u_t) \tag{B.6}$$

with state vector $x_t$, controlled inputs $u_t$, and measured outputs $y_t$. Our aim is to estimate the sequence $\{x_0, x_1, \dots, x_T\}$ over some time period $[0, T]$ given only the measurements $y_t$ and inputs $u_t$ at each time step. Our observer takes the form

$$\hat{x}_{t+1} = f_o(\hat{x}_t, u_t, y_t) \tag{B.7}$$

where $\hat{x}_t$ is the state estimate. To estimate the true state, our observer error $(x_t - \hat{x}_t)$ must converge to zero as time progresses, or $\hat{x}_t \to x_t$ as $t \to \infty$. We studied this in detail alongside the Youla-Kučera parametrisation in Chapters 4 and 5. To summarise, our observer only has to satisfy the following two conditions to guarantee convergence.

1. The observer must be a contracting system.

2. The observer must satisfy a correctness condition which says that, given perfect knowledge of the state and inputs, the observer can exactly predict the next state. Mathematically, we write this as

$$f_o(x_t, u_t, h_d(x_t, u_t)) = f_d(x_t, u_t). \tag{B.8}$$

   It can also be shown that if the correctness condition is only approximately satisfied such that $|f_o(x_t, u_t, y_t) - f_d(x_t, u_t)| < \rho$ for some small number $\rho \in \mathbb{R}$, then the observer error will still be bounded [136, App. E].

The first condition, contraction, is already guaranteed for all RENs thanks to their direct parametrisation. Our goal is therefore to minimise the one-step-ahead pre-

diction error to approximate the correctness condition. If we have a batch of data $z = \{x_i, u_i, y_i \mid i = 1, 2, \ldots, N\}$, this corresponds to minimising the loss function

$$\mathcal{L}(z, \theta) = \sum_{i=1}^{N} |f_o(x_i, u_i, y_i) - f_d(x_i, u_i)|^2, \tag{B.9}$$

where $\theta$ contains the learnable parameters of the REN.

### Generating training data

Consider the same nonlinear box system from Appendix B.2.2, with the a change in setup so that we can only measure the box position. We introduce a measurement function `hd()` such that $y_t = q_t$.

```
m = 1                       # Mass (kg)
k = 5                       # Spring constant (N/m)
μ = 0.5                     # Viscous damping (kg/m)
nx = 2                      # Number of states

f(x::Matrix,u::Matrix) = [x[2:2,:]; (u[1:1,:] -
    k*x[1:1,:] - μ*x[2:2,:]*abs.(x[2:2,:]))/m]
fd(x,u) = x + dt*f(x,u)
hd(x::Matrix) = x[1:1,:]
```

For this example, we assume that the box always starts at rest in a random initial position between $\pm 0.5$m, after which it is released and allowed to oscillate freely with no added forces (so $u = 0$). Learning an observer typically requires a large amount of training data to fully capture the behaviour of the system, hence we consider 200 batches each simulating 10 s of motion.

```
Tmax, dt = 10, 0.01     # Simulation horizon and time step (s)
ts = 1:Int(Tmax/dt)     # Time array indices

# Generate batches of training data
batches = 200
u = fill(zeros(1, batches), length(ts)-1)
X = fill(zeros(1, batches), length(ts))
X[1] = 0.5*(2*rand(nx, batches) .- 1)
for t in ts[1:end-1]
    X[t+1] = fd(X[t],u[t])
end
```

We have stored the states of the system across each batch in `X`. To compute the one-step-ahead loss $\mathcal{L}$, we will need to separate this data into the states at the current time step `Xt` and at the next time step `Xn`, then compute the measurements. We then store the data for training, shuffling it so there is no bias towards earlier time steps.

```
using Random

# Current/next state, measurements
Xt = X[1:end-1]
Xn = X[2:end]
y  = hd.(Xt)

# Store training data
obsv_data = [[ut; yt] for (ut,yt) in zip(u, y)]
indx = shuffle(1:length(obsv_data))
data = zip(Xn[indx], Xt[indx], obsv_data[indx])
```

### Defining a model

We can construct the parametrisation for a contracting REN using the constructor `ContractingRENParams`. The inputs to the model are $[u_t; y_t]$, and its outputs are the next state estimate $\hat{x}_{t+1}$. The flag `output_map=false` sets the output map of the REN to just return its own internal state – i.e., $C_2 = I$, $D_{21} = 0$, $D_{22} = 0$, $b_y = 0$ in the LTI component of the REN (2.20a). This makes the internal state of the REN exactly the state estimate $\hat{x}_t$.

```
using RobustNeuralNetworks

T  = Float32
nv = 200
nu = size(obsv_data[1], 1)
ny = nx
model_ps = ContractingRENParams{T}(nu, nx, nv, ny; output_map=false)
model = DiffREN(model_ps)
```

### Defining a loss function

our loss function is the one-step-ahead prediction error of the REN observer. We write this as follows, noting that all subtypes of `AbstractREN` return both their updated internal state and their output (in that order).

```
using Statistics

function loss(model, xn, xt, inputs)
    xpred = model(xt, inputs)[1]
    return mean(sum((xn - xpred).^2, dims=1))
end
```

### Training the model

The function below trains the observer with the Adam optimiser over 100 epochs and decreases the maximum learning rate from $10^{-3}$ to $10^{-4}$ if the mean loss stops decreasing between epochs. The core of this function is a simple `Flux.jl` training loop, expanded out for clarity.

```
using Flux

function train_observer!(model, data; epochs=50, lr=1e-3, min_lr=1e-6)
    opt_state = Flux.setup(Adam(lr), model)
    mean_loss = [1e5]
    for epoch in 1:epochs
        # Gradient descent update
        batch_loss = []
        for (xn, xt, inputs) in data
            tloss, dJ = Flux.withgradient(loss, model, xn, xt, inputs)
            Flux.update!(opt_state, model, dJ[1])
            push!(batch_loss, tloss)
        end
        # Reduce lr if loss is stuck or growing
        push!(mean_loss, mean(batch_loss))
        if (mean_loss[end] >= mean_loss[end-1]) && (lr > min_lr)
            lr *= 0.1
            Flux.adjust!(opt_state, lr)
        end
    end
    return mean_loss
end
tloss = train_observer!(model, data)
```

### Evaluating the trained model

We have trained the REN observer to minimise the one-step-ahead prediction error, but we are yet to test whether the the observer error actually does converge to zero. We set up the following 50 batches of test data as a demonstration.

```
batches   = 50
ts_test   = 1:Int(20/dt)
u_test    = fill(zeros(1, batches),length(ts_test))
x_test    = fill(zeros(nx,batches),length(ts_test))
x_test[1] = 0.2*(2*rand(nx, batches) .-1)

for t in ts_test[1:end-1]
    x_test[t+1] = fd(x_test[t], u_test[t])
end
y_test = hd.(x_test)
obsv_in = [[u;y] for (u,y) in zip(u_test, y_test)]
```

Next, we need a function to simulate the REN observer using its own state $\hat{x}_t$ rather than the true system state $x_t$, which was used for training. We use the very neat tool `Flux.Recur` for this. We assume that the observer has no knowledge of the initial state and simply guesses $\hat{x}_0 = 0$ for all 50 batches.

```
function simulate(model::AbstractREN, x0, u)
    recurrent = Flux.Recur(model, x0)
    output = recurrent.(u)
    return output
end
x0hat = zeros(model.nx, batches)
xhat = simulate(model, x0hat, obsv_in)
```

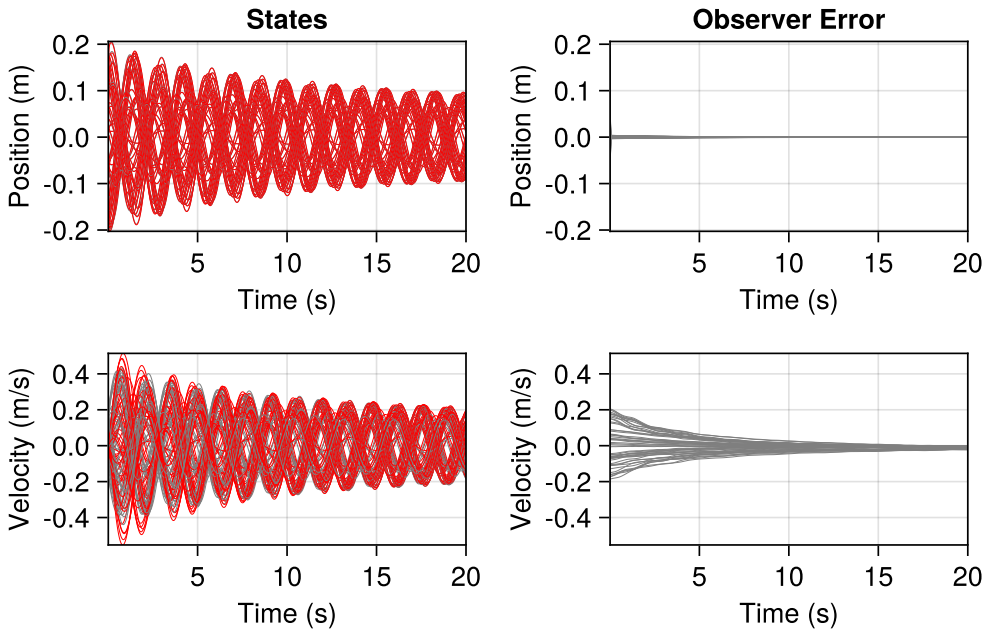



**Figure B.7:** Simulation results showing the observer predictions and observer error with the box starting at 50 different initial conditions. The left panels compare the true (grey) and estimated (red) states, while the right panels show the observer error $x - \hat{x}$ over time.

The results are plotted in Figure B.7. In the left-hand panels, the observer predictions (red) almost exactly match the true states (grey) after approximately 4 s. This is confirmed by the right-hand panels, which show the observer error $x_t - \hat{x}_t$ smoothly converging to zero as the observer estimates the correct states for all simulations.

It is worth noting that at no point did we directly train the REN to minimise the observer error. This is a natural result of using a contracting model and training it to minimise the one-step-ahead prediction error. There is still some residual observer error in the velocity in Figure B.7, since our observer was only trained to approximately satisfy the correctness condition. However, this could easily be reduced or eliminated using a larger observer model, more training data, and more training epochs.

## B.3 Conclusions

This appendix has presented `RobustNeuralNetworks.jl`, a Julia package for robust machine learning based on the REN and Sandwich network model classes. We have

outlined the package structure and its usage alongside Julia's main machine-learning library, `Flux.jl`, and have demonstrated via examples in image classification, reinforcement learning, and observer design that the package is easy to use in many common machine learning and data-driven control problems, while also offering the advantage of robustness guarantees.

We intend `RobustNeuralNetworks.jl` to be widely-applicable in the scientific and machine learning communities for learning-based problems in which robustness certificates are crucial. Some areas in which this package will be most applicable include: data-driven control and state estimation; image classification and segmentation; and privacy and security. We will continue to expand on open-source implementations of directly-parametrised robust neural networks in our new Python package, `RobustNeuralNetworks`[3].

# References


[1] Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, 2018.

[2] Brian D. O. Anderson. The small-gain theorem, the passivity theorem and their equivalence. *Journal of the Franklin Institute*, 1972.

[3] Brian D. O. Anderson. From Youla-Kucera to identification, adaptive and nonlinear control. *Automatica*, 1998.

[4] Vincent Andrieu and Christophe Prieur. Uniting two control Lyapunov functions for affine systems. *IEEE Trans. Autom. Control*, 2010.

[5] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Leonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters for on-policy deep actor-critic methods? A large-scale study. In *Proceedings of the International Conference on Learning Representations*, 2021.

[6] Alexandre Araujo, Aaron J. Havens, Blaise Delattre, Alexandre Allauzen, and Bin Hu. A unified algebraic perspective on Lipschitz neural networks. In *Proceedings of the International Conference on Learning Representations*, 2023.

[7] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 2017.

[8] Karl J Åström. *Introduction to stochastic control theory*. Courier Corporation, 2012.

[9] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, 2019.

[10] Bram Bakker. Reinforcement learning with long short-term memory. In *Advances in Neural Information Processing Systems*. MIT Press, 2001.

[11] Nicholas H. Barbara, Max Revay, Ruigang Wang, Jing Cheng, and Ian R. Manchester. RobustNeuralNetworks.jl: a package for machine learning and data-driven control with certified robustness. *Proceedings of the JuliaCon Conferences*, 2025.

[12] Nicholas H. Barbara, Ruigang Wang, and Ian R. Manchester. Learning over contracting and Lipschitz closed-loops for partially-observed nonlinear systems. *Proceedings of the IEEE Conference on Decision and Control*, 2023.

[13] Nicholas H. Barbara, Ruigang Wang, and Ian R. Manchester. On robust reinforcement learning with Lipschitz-bounded policy networks. In *Systems Theory in Data and Optimization: Proceedings of SysDO*, 2024.

[14] Nicholas H. Barbara, Ruigang Wang, and Ian R. Manchester. R2DN: Scalable parameterization of contracting and Lipschitz recurrent deep networks. *arXiv preprint arXiv:2504.01250*, 2025.

[15] Nicholas H. Barbara, Ruigang Wang, Alexandre Megretski, and Ian R. Manchester. React to surprises: Stable-by-design neural feedback control and the Youla-REN. *arXiv preprint arXiv:2506.01226*, 2025.

[16] Jan Bendtsen, Klaus Trangbaek, and Jakob Stoustrup. Plug-and-play control—modifying control systems online. *IEEE Transactions on Control Systems Technology*, 2013.

[17] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 1994.

[18] Dimitri Bertsekas. *Reinforcement Learning and Optimal Control*. Athena Scientific, 2019.

[19] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 2017.

[20] Johan Bjorck, Carla P. Gomes, and Kilian Q. Weinberger. Towards deeper deep reinforcement learning. *Advances in Neural Information Processing Systems*, 2021.

[21] Nicolas Boumal. *An introduction to optimization on smooth manifolds*. Cambridge University Press, 2023.

[22] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[23] Lukas Brunke, Melissa Greeff, Adam W. Hall, Zhaocong Yuan, Siqi Zhou, Jacopo Panerati, and Angela P. Schoellig. Safe learning in robotics: From learning-based control to safe reinforcement learning. *Annual Review of Control, Robotics, and Autonomous Systems*, 2022.

[24] Francesco Bullo. *Contraction Theory for Dynamical Systems*. Kindle Direct Publishing, 2022.

[25] Ken Caluwaerts, Atil Iscen, J. Chase Kew, Wenhao Yu, Tingnan Zhang, Daniel Freeman, Kuang-Huei Lee, Lisa Lee, Stefano Saliceti, Vincent Zhuang, et al. Barkour: Benchmarking animal-level agility with quadruped robots. *arXiv preprint arXiv:2305.14654*, 2023.

[26] Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. *IEEE Symposium on Security and Privacy Workshops*, 2018.

[27] Stephen Casper, Xander Davies, Claudia Shi, Thomas Krendl Gilbert, Jérémy Scheurer, Javier Rando, Rachel Freedman, Tomek Korbak, David Lindner, Pedro Freire, Tony Tong Wang, Samuel Marks, Charbel-Raphael Segerie, Micah Carroll, Andi Peng, Phillip J. K. Christoffersen, Mehul Damani, Stewart Slocum, Usman Anwar, Anand Siththaranjan, Max Nadeau, Eric J. Michaud, Jacob Pfau, Dmitrii Krasheninnikov, Xin Chen, Lauro Langosco, Peter Hase, Erdem Biyik, Anca Dragan, David Krueger, Dorsa Sadigh, and Dylan Hadfield-Menell. Open problems and fundamental limitations of reinforcement learning from human feedback. *Transactions on Machine Learning Research*, 2023.

[28] Veronica Centorrino, Francesco Bullo, and Giovanni Russo. Modeling and contractivity of neural-synaptic networks with Hebbian learning. *Automatica*, 2024.

[29] Thomas Chaffey, Fulvio Forni, and Rodolphe Sepulchre. Graphical nonlinear system analysis. *IEEE Transactions on Automatic Control*, 2023.

[30] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Adversarial attacks and defences: A survey. *arXiv preprint arXiv:1810.00069*, 2018.

[31] Jing Cheng, Ruigang Wang, and Ian R. Manchester. Learning stable and passive neural differential equations. In *Proceedings of the IEEE Conference on Decision and Control*, 2024.

[32] Jeremy Coulson, John Lygeros, and Florian Dörfler. Data-enabled predictive control: In the shallows of the DeePC. In *Proceedings of the European Control Conference*, 2019.

[33] Hongkai Dai, Benoit Landry, Lujie Yang, Marco Pavone, and Russ Tedrake. Lyapunov-stable neural-network control. In *Robotics: Science and Systems*, 2021.

[34] Alexander Davydov and Francesco Bullo. Perspectives on contractivity in control, optimization, and learning. *IEEE Control Systems Letters*, 2024.

[35] Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de las Casas, Craig Donner, Leslie Fritz, Cristian Galperti, Andrea Huber, James Keeling, Maria Tsimpoukelli, Jackie Kay, Antoine Merle, Jean-Marc Moret, Seb Noury, Federico Pesamosca, David Pfau, Olivier Sauter, Cristian Sommariva, Stefano Coda, Basil Duval, Ambrogio Fasoli, Pushmeet Kohli, Koray Kavukcuoglu, Demis Hassabis, and Martin Riedmiller. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 2022.

[36] Quentin Delfosse, Jannis Blüml, Fabian Tatai, Théo Vincent, Bjarne Gregori, Elisabeth Dillies, Jan Peters, Constantin A. Rothkopf, and Kristian Kersting. Deep reinforcement learning agents are not even close to human intelligence. In *Inductive biases in Reinforcement Learning Workshop, Reinforcement Learning Conference*, 2025.

[37] Charles A. Desoer and C. A. Lin. Two-step compensation of nonlinear systems. *Systems & Control Letters*, 1983.

[38] Charles A. Desoer and R.-W. Liu. Global parametrization of feedback systems with nonlinear plants. *Systems & Control Letters*, 1982.

[39] Gavin Weiguang Ding, Luyu Wang, and Xiaomeng Jin. AdverTorch v0.1: An adversarial robustness toolbox based on pytorch. *arXiv preprint arXiv:1902.07623*, 2019.

[40] John C. Doyle. Guaranteed margins for LQG regulators. *IEEE Transactions on Automatic Control*, 1978.

[41] John C. Doyle. *Matrix Interpolation Theory and Optimal Control*. PhD thesis, University of California, 1984.

[42] John C. Doyle, Keith Glover, Pramod P. Khargonekar, and Bruce A. Francis. State-space solutions to standard $\mathcal{H}_2$ and $\mathcal{H}_\infty$ control problems. *IEEE Transactions on Automatic Control*, 1989.

[43] Laurent El Ghaoui, Fangda Gu, Bertrand Travacca, Armin Askari, and Alicia Tsai. Implicit deep learning. *SIAM Journal on Mathematics of Data Science*, 2021.

[44] Mahyar Fazlyab, Alexander Robey, Hamed Hassani, Manfred Morari, and George J. Pappas. Efficient and accurate estimation of Lipschitz constants for deep neural networks. *Advances in Neural Information Processing Systems*, 2019.

[45] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *Proceedings of the International Conference on Machine Learning*. PMLR, 2016.

[46] C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax – a differentiable physics engine for large scale rigid body simulation. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2021.

[47] Kenji Fujimoto and Toshiharu Sugie. State-space characterization of Youla parametrization for nonlinear systems based on input-to-state stability. *Pro-*

*ceedings of the IEEE Conference on Decision and Control*, 1998.

[48] Kenji Fujimoto and Toshiharu Sugie. Characterization of all nonlinear stabilizing controllers via observer-based kernel representations. *Automatica*, 2000.

[49] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the International Conference on Machine Learning*, 2018.

[50] Luca Furieri, Clara Lucía Galimberti, and Giancarlo Ferrari-Trecate. Neural system level synthesis: Learning over all stabilizing policies for nonlinear systems. In *Proceedings of the IEEE Conference on Decision and Control*, 2022.

[51] Luca Furieri, Clara Lucía Galimberti, and Giancarlo Ferrari-Trecate. Learning to boost the performance of stable nonlinear systems. *IEEE Open Journal of Control Systems*, 2024.

[52] Clara Lucía Galimberti, Luca Furieri, and Giancarlo Ferrari-Trecate. Parametrizations of all stable closed-loop responses: From theory to neural network control design. *Annual Reviews in Control*, 2025.

[53] Carlos E. Garcia and Manfred Morari. Internal model control. a unifying review and some new results. *Industrial & Engineering Chemistry Process Design and Development*, 1982.

[54] Javier García and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 2015.

[55] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *Proceedings of the International Conference on Learning Representations*, 2014.

[56] Paul J. Goulart, Eric C. Kerrigan, and Jan M. MacIejowski. Optimization over state feedback policies for robust control with constraints. *Automatica*, 2006.

[57] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *Conference on Language Modeling*, 2024.

[58] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo:

Recurrent memory with optimal polynomial projections. *Advances in Neural Information Processing Systems*, 2020.

[59] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *Proceedings of the International Conference on Learning Representations*, 2022.

[60] Fangda Gu, He Yin, Laurent El Ghaoui, Murat Arcak, Peter Seiler, and Ming Jin. Recurrent neural network controllers synthesis with stability guarantees for partially observed systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

[61] Shangding Gu, Long Yang, Yali Du, Guang Chen, Florian Walter, Jun Wang, and Alois Knoll. A review of safe reinforcement learning: Methods, theories, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[62] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2017.

[63] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the International Conference on Machine Learning*, 2018.

[64] Joao P. Hespanha. *Linear systems theory*. Princeton university press, 2018.

[65] Lukas Hewing, Kim P. Wabersich, Marcel Menner, and Melanie N. Zeilinger. Learning-based model predictive control: Toward safe learning in control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2020.

[66] Dimitar Ho. A system level approach to discrete-time nonlinear systems. In *Proceedings of the American Control Conference*, 2020.

[67] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[68] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward

networks are universal approximators. *Neural Networks*, 1989.

[69] Bin Hu, Kaiqing Zhang, Na Li, Mehran Mesbahi, Maryam Fazel, and Tamer Başar. Toward a theoretical foundation of policy optimization for learning control policies. *Annual Review of Control, Robotics, and Autonomous Systems*, 2023.

[70] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. In *Proceedings of the International Conference on Learning Representations – Workshop Track Proceedings*. Proceedings of the International Conference on Learning Representations, 2017.

[71] Shengyi Huang, Rousslan Fernand Julien Dossa, Antonin Raffin, Anssi Kanervisto, and Weixun Wang. The 37 implementation details of proximal policy optimization. In *ICLR Blog Track*, 2022. https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/.

[72] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G. M. Araújo. CleanRL: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 2022.

[73] Marco Hutter, Christian Gehring, Dominic Jud, Andreas Lauber, C. Dario Bellicoso, Vassilios Tsounis, Jemin Hwangbo, Karen Bodie, Peter Fankhauser, Michael Bloesch, Remo Diethelm, Samuel Bachmann, Amir Melzer, and Mark Hoepflinger. ANYmal – a highly mobile and dynamic quadrupedal robot. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, 2016.

[74] Jun Ichi Imura and Tsuneo Yoshikawa. Parametrization of all stabilizing controllers of nonlinear systems. *Systems & Control Letters*, 1997.

[75] Michael Innes. Don't unroll adjoint: Differentiating SSA-form programs. *CoRR*, 2018.

[76] Mike Innes. Flux: Elegant machine learning with julia. *Journal of Open Source Software*, 2018.

[77] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 2004.

[78] Ming Jin and Javad Lavaei. Stability-certified reinforcement learning: A control-theoretic perspective. *IEEE Access*, 2020.

[79] Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. *Proceedings of the IEEE International Conference on Robotics and Automation*, 2019.

[80] Neelay Junnarkar, He Yin, Fangda Gu, Murat Arcak, and Peter Seiler. Synthesis of stabilizing recurrent equilibrium network controllers. *Proceedings of the IEEE Conference on Decision and Control*, 2022.

[81] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Scalable deep reinforcement learning for vision-based robotic manipulation. *Proceedings of the Conference on Robot Learning*, 2018.

[82] Leonid V. Kantorovich and Gleb P. Akilov. *Functional Analysis*, chapter IV – Normed Spaces, pages 82–126. Pergamon, second edition, 1982.

[83] Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature 2023 620:7976*, 2023.

[84] Yu Kawano, Arjan J. van der Schaft, and Jacquelien M. A. Scherpen. Image and kernel representations for variational systems. *IFAC-PapersOnLine*, 2023.

[85] Yu Kawano, Arjan J. van der Schaft, and Jacquelien M. A. Scherpen. Youla-Kučera parametrization in the contraction framework. *IEEE Transactions on Automatic Control*, 2024.

[86] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 2023.

[87] Hassan K. Khalil. *Nonlinear systems*. Prentice-Hall, third edition, 2002.

[88] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*.

Proceedings of the International Conference on Learning Representations, 2015.

[89] Leo Kozachkov, Mikael Lundqvist, Jean-Jacques E. Slotine, and Earl K. Miller. Achieving stable dynamics in neural circuits. *PLOS Computational Biology*, 2020.

[90] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2012.

[91] Aounon Kumar, Alexander Levine, and Soheil Feizi. Policy smoothing for provably robust reinforcement learning. *Proceedings of the International Conference on Learning Representations*, 2022.

[92] Vladimír Kučera. Stability of discrete linear feedback systems. *IFAC Proceedings Volumes*, 1975.

[93] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[94] Yann LeCun, Corinna Cortes, and Christopher J. C. Burges. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2010.

[95] Laurent Lessard. The analysis of optimization algorithms: A dissipativity approach. *IEEE Control Systems Magazine*, 2022.

[96] Laurent Lessard, Benjamin Recht, and Andrew Packard. Analysis and design of optimization algorithms via integral quadratic constraints. *SIAM Journal on Optimization*, 2016.

[97] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *Proceedings of the International Conference on Learning Representations*, 2016.

[98] Winfried Lohmiller and Jean-Jacques E. Slotine. On contraction analysis for non-linear systems. *Automatica*, 1998.

[99] Wei Min Lu. A state-space approach to parameterization of stabilizing controllers for nonlinear systems. *IEEE Transactions on Automatic Control*, 1995.

[100] Jing Yuan Luo, Yunlong Song, Victor Klemm, Fan Shi, Davide Scaramuzza, and Marco Hutter. Residual policy learning for perceptive quadruped control using differentiable simulation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2025.

[101] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *Proceedings of the International Conference on Learning Representations*, 2018.

[102] Imane Mahtout, Francisco Navas, Vicente Milanes, and Fawzi Nashashibi. Advances in Youla-Kucera parametrization: A review. *Annual Reviews in Control*, 2020.

[103] Ian R. Manchester and Jean-Jacques E. Slotine. Transverse contraction criteria for existence, stability, and robustness of a limit cycle. *Systems & Control Letters*, 2014.

[104] Ian R. Manchester and Jean-Jacques E. Slotine. Control contraction metrics: Convex and intrinsic criteria for nonlinear feedback design. *IEEE Transactions on Automatic Control*, 2017.

[105] Ian R. Manchester and Jean-Jacques E. Slotine. Robust control contraction metrics: A convex approach to nonlinear state-feedback $\mathcal{H}_\infty$ control. *IEEE Control Systems Letters*, 2018.

[106] Gaurav Manek and J. Zico Kolter. Learning stable deep dynamics models. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019.

[107] Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search of static linear policies is competitive for reinforcement learning. *Advances in Neural Information Processing Systems*, 2018.

[108] Alexandre Megretski and Anders Rantzer. System analysis via integral quadratic constraints. *IEEE Transactions on Automatic Control*, 1997.

[109] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 2022.

[110] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 2021.

[111] Mayank Mittal, Calvin Yu, Qinxi Yu, Jingzhou Liu, Nikita Rudin, David Hoeller, Jia Lin Yuan, Ritvik Singh, Yunrong Guo, Hammad Mazhar, Ajay Mandlekar, Buck Babich, Gavriel State, Marco Hutter, and Animesh Garg. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 2023.

[112] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *Proceedings of the International Conference on Learning Representations*, 2018.

[113] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

[114] Buqing Nie, Jingtian Ji, Yangqing Fu, and Yue Gao. Improve robustness of reinforcement learning against observation perturbations via $\ell_\infty$ Lipschitz policy networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

[115] Jean-Philippe Noël and Maarten Schoukens. F-16 aircraft benchmark based on ground vibration test data. In *Workshop on Nonlinear System Identification Benchmarks*, 2017.

[116] Tuomas Oikarinen, Wang Zhang, Mit Meche, Alexandre Megretski, Luca Daniel, and Tsui-Wei Weng. Robust deep reinforcement learning through adversarial loss. *Advances in Neural Information Processing Systems*, 2021.

[117] Yuji Okamoto and Ryosuke Kojima. Learning deep dissipative dynamics. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025.

[118] Maurıcio C. De Oliveira, José C. Geromel, and Jacques Bernussou. Extended $\mathcal{H}_2$ and $\mathcal{H}_\infty$ norm characterizations and controller parametrizations for discrete-time

systems. *International Journal of Control*, 2002.

[119] Andrew D. B. Paice and John B. Moore. On the Youla-Kucera parametrization for nonlinear systems. *Systems & Control Letters*, 1990.

[120] Andrew D. B. Paice and Arjan J. van der Schaft. Stable kernel representations and the Youla parameterization for nonlinear systems. In *Proceedings of the IEEE Conference on Decision and Control*, 1994.

[121] Andrew D. B. Paice and Arjan J. van der Schaft. The class of stabilizing nonlinear plant controller pairs. *IEEE Transactions on Automatic Control*, 1996.

[122] Anay Pattanaik, Zhenyi Tang, Shuijing Liu, Gautham Bommannan, and Girish Chowdhary. Robust deep reinforcement learning with adversarial attacks. *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2017.

[123] Patricia Pauli, Anne Koch, Julian Berberich, Paul Kohler, and Frank Allgower. Training robust neural networks using Lipschitz bounds. *IEEE Control Systems Letters*, 2022.

[124] Patricia Pauli, Ruigang Wang, Ian R. Manchester, and Frank Allgöwer. Lipschitz-bounded 1d convolutional neural networks using the Cayley transform and the controllability gramian. *Proceedings of the IEEE Conference on Decision and Control*, 2023.

[125] Patricia Pauli, Ruigang Wang, Ian R. Manchester, and Frank Allgöwer. LipKernel: Lipschitz-bounded convolutional neural networks via dissipative layers. *arXiv preprint arXiv:2410.22258*, 2024.

[126] Bernd Prach and Christoph H. Lampert. Almost-orthogonal layers for efficient general-purpose Lipschitz networks. In *European Conference on Computer Vision*. Springer, 2022.

[127] Lukas Pries and Markus Ryll. Learning robust agile flight control with stability guarantees. *arXiv preprint arXiv:2510.12611*, 2025.

[128] Xianbiao Qi, Jianan Wang, Yihao Chen, Yukai Shi, and Lei Zhang. LipsFormer: Introducing Lipschitz continuity to vision transformers. In *Proceedings of the*

*International Conference on Learning Representations*, 2023.

[129] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-Baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 2021.

[130] Jams B. Rawlings, David Q. Mayne, and Moritz Diehl. *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017.

[131] Benjamin Recht. A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2019.

[132] Max Revay. *A Behavioral Approach to Robust Machine Learning*. PhD thesis, University of Sydney, 2021.

[133] Max Revay, Ruigang Wang, and Ian R. Manchester. Lipschitz bounded equilibrium networks. *arXiv preprint arXiv:2010.01732*, 2020.

[134] Max Revay, Ruigang Wang, and Ian R. Manchester. A convex parameterization of robust recurrent neural networks. *IEEE Control Systems Letters*, 2021.

[135] Max Revay, Ruigang Wang, and Ian R. Manchester. Recurrent equilibrium networks: Unconstrained learning of stable and robust dynamical models. In *Proceedings of the IEEE Conference on Decision and Control*, 2021.

[136] Max Revay, Ruigang Wang, and Ian R. Manchester. Recurrent equilibrium networks: Flexible dynamic models with guaranteed stability and robustness. *IEEE Transactions on Automatic Control*, 2023.

[137] John W. Roberts, Ian R. Manchester, and Russ Tedrake. Feedback controller parameterizations for reinforcement learning. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, 2011.

[138] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. *Proceedings of the Conference on Robot Learning*, 2021.

[139] Alessio Russo and Alexandre Proutiere. Towards optimal attacks on reinforcement learning policies. *Proceedings of the American Control Conference*, 2021.

[140] Maarten Schoukens and Koen Tiels. Identification of block-oriented nonlinear systems starting from linear approximations: A survey. *Automatica*, 2017.

[141] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *Proceedings of the International Conference on Machine Learning*, 2015.

[142] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *Proceedings of the International Conference on Learning Representations*, 2015.

[143] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[144] Fan Shi, Chong Zhang, Takahiro Miki, Joonho Lee, Marco Hutter, and Stelian Coros. Rethinking robustness assessment: Adversarial attacks on learning-based quadrupedal locomotion controllers. In *Robotics: Science and Systems*, 2024.

[145] Guanya Shi, Xichen Shi, Michael O'Connell, Rose Yu, Kamyar Azizzadenesheli, Animashree Anandkumar, Yisong Yue, and Soon-Jo Chung. Neural lander: Stable drone landing control using learned dynamics. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2019.

[146] Jerome Sieber, Carmen Amo Alonso, Alexandre Didier, Melanie N. Zeilinger, and Antonio Orvieto. Understanding the differences in foundation models: Attention, state space models, and recurrent neural networks. In *Neural Information Processing Systems*, 2024.

[147] Hava T. Siegelmann and Eduardo D. Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 1995.

[148] Jonah Siekmann, Yesh Godse, Alan Fern, and Jonathan Hurst. Sim-to-real learning of all common bipedal gaits via periodic reward composition. *Proceedings of the IEEE International Conference on Robotics and Automation*, 2021.

[149] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham,

Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.

[150] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George Van Den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 2017.

[151] Tom Silver, Kelsey Allen, Josh Tenenbaum, and Leslie Kaelbling. Residual policy learning. *arXiv preprint arXiv:1812.06298*, 2018.

[152] Max Simchowitz, Karan Singh, and Elad Hazan. Improper learning for non-stochastic control. In *Conference on Learning Theory*. PMLR, 2020.

[153] Sumeet Singh, Benoit Landry, Anirudha Majumdar, Jean-Jacques E. Slotine, and Marco Pavone. Robust feedback motion planning via contraction theory. *The International Journal of Robotics Research*, 2023.

[154] Sahil Singla and Soheil Feizi. Skew orthogonal convolutions. In *Proceedings of the International Conference on Machine Learning*. PMLR, 2021.

[155] Jean-Jacques E. Slotine and Weiping Li. *Applied Nonlinear Control*. Prentice-Hall, 1991.

[156] Yunlong Song, Angel Romero, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Reaching the limit in autonomous racing: Optimal control versus reinforcement learning. *Science Robotics*, 2023.

[157] Eduardo D. Sontag. Contractive systems with inputs. *Lecture Notes in Control and Information Sciences*, 2010.

[158] H. J. Terry Suh, Max Simchowitz, Kaiqing Zhang, and Russ Tedrake. Do differentiable simulators give better policy gradients? *PMLR*, 2022.

[159] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[160] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *Proceedings of the International Conference on Learning Representations*, 2013.

[161] Ryoichi Takase, Nobuyuki Yoshikawa, Toshisada Mariyama, and Takeshi Tsuchiya. Stability-certified reinforcement learning control via spectral normalization. *Machine Learning with Applications*, 2022.

[162] Justin Z. Tang and Ian R. Manchester. Transverse contraction criteria for stability of nonlinear hybrid limit cycles. In *Proceedings of the IEEE Conference on Decision and Control*, 2014.

[163] T. T. Tay and John B. Moore. Left coprime factorizations and a class of stabilizing controllers for nonlinear systems. In *Proceedings of the IEEE Conference on Decision and Control*, 1988.

[164] Mark M. Tobenkin, Ian R. Manchester, and Alexandre Megretski. Convex parameterizations and fidelity bounds for nonlinear identification and reduced-order modelling. *IEEE Transactions on Automatic Control*, 2017.

[165] Mark M. Tobenkin, Ian R. Manchester, Jennifer Wang, Alexandre Megretski, and Russ Tedrake. Convex optimization in identification of stable non-linear state space models. In *Proceedings of the IEEE Conference on Decision and Control*, 2010.

[166] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, 2017.

[167] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, 2012.

[168] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, 2023.

[169] Duc N. Tran, Björn S. Rüffer, and Christopher M. Kellett. Convergence properties for discrete-time nonlinear systems. *IEEE Transactions on Automatic Control*, 2019.

[170] Asher Trockman and J. Zico Kolter. Orthogonalizing convolutional layers with the Cayley transform. *Proceedings of the International Conference on Learning Representations*, 2021.

[171] Hiroyasu Tsukamoto and Soon-Jo Chung. Learning-based robust motion planning with guaranteed stability: A contraction theory approach. *IEEE Robotics and Automation Letters*, 2021.

[172] Hiroyasu Tsukamoto and Soon-Jo Chung. Neural contraction metrics for robust estimation and control: A convex optimization approach. *IEEE Control Systems Letters*, 2021.

[173] Hiroyasu Tsukamoto, Soon Jo Chung, and Jean-Jacques E. Slotine. Contraction theory for nonlinear stability analysis and learning-based control: A tutorial overview. *Annual Reviews in Control*, 2021.

[174] Jack Umenberger and Ian R. Manchester. Specialized interior-point algorithm for stable nonlinear system identification. *IEEE Transactions on Automatic Control*, 2018.

[175] Arjan J. van der Schaft. *L2-Gain and Passivity Techniques in Nonlinear Control*. Springer International Publishing, 3 edition, 2017.

[176] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[177] Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. *Advances in Neural Information Processing Systems*, 2018.

[178] Ruigang Wang, Nicholas H. Barbara, Max Revay, and Ian R. Manchester. Learning over all stabilizing nonlinear controllers for a partially-observed linear system. *IEEE Control Systems Letters*, 2022.

[179] Ruigang Wang, Krishnamurthy Dj Dvijotham, and Ian R. Manchester. Monotone, bi-Lipschitz, and Polyak-Łojasiewicz networks. In *Proceedings of the International Conference on Machine Learning*. PMLR, 2024.

[180] Ruigang Wang and Ian R. Manchester. Youla-REN: Learning nonlinear feedback policies with robust stability guarantees. *Proceedings of the American Control Conference*, 2022.

[181] Ruigang Wang and Ian R. Manchester. Direct parameterization of Lipschitz-bounded deep networks. In *Proceedings of the International Conference on Machine Learning*. PMLR, 2023.

[182] Ruigang Wang, Roland Tóth, Patrick J. W. Koelewijn, and Ian R. Manchester. Virtual control contraction metrics: Convex nonlinear feedback design via behavioral embedding. *International Journal of Robust and Nonlinear Control*, 2024.

[183] Jiayi Weng, Min Lin, Shengyi Huang, Bo Liu, Denys Makoviichuk, Viktor Makoviychuk, Zichen Liu, Yufan Song, Ting Luo, Yukun Jiang, Zhongwen Xu, and Shuicheng Yan. EnvPool: A highly parallel reinforcement learning environment execution engine. *Advances in Neural Information Processing Systems*, 2022.

[184] Nina Wiedemann, Valentin Wüest, Antonio Loquercio, Matthias Müller, Dario Floreano, and Davide Scaramuzza. Training efficient controllers via analytic policy gradient. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2023.

[185] Daan Wierstra, Alexander Förster, Jan Peters, and Jürgen Schmidhuber. Recurrent policy gradients. *Logic Journal of the IGPL*, 2010.

[186] Jan C. Willems. Dissipative dynamical systems part I: General theory. *Archive for Rational Mechanics and Analysis*, 1972.

[187] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 1992.

[188] Ezra Winston and J. Zico Kolter. Monotone operator equilibrium networks. In *Advances in Neural Information Processing Systems*, 2020.

[189] Fan Wu, Linyi Li, Zijian Huang, Yevgeniy Vorobeychik, Ding Zhao, and Bo Li. Crop: Certifying robust policies for reinforcement learning through functional smoothing. *Proceedings of the International Conference on Learning Representations*, 2022.

[190] Lujie Yang, Hongkai Dai, Zhouxing Shi, Cho-Jui Hsieh, Russ Tedrake, and Huan Zhang. Lyapunov-stable neural control for state and output feedback: A novel formulation. In *Proceedings of the International Conference on Machine Learning*, 2024.

[191] Bowen Yi and Ian R. Manchester. On the equivalence of contraction and Koopman approaches for nonlinear stability and control. *IEEE Transactions on Automatic Control*, 2024.

[192] Izzet B. Yildiz, Herbert Jaeger, and Stefan J. Kiebel. Re-visiting the echo state property. *Neural Networks*, 2012.

[193] Dante C. Youla, Joseph J. Bongiorno, and Hamid A. Jabr. Modern wiener-hopf design of optimal controllers — part II: The multivariable case. *IEEE Transactions on Automatic Control*, 1976.

[194] George Zames. Feedback and optimal sensitivity: Model reference transformations, multiplicative seminorms, and approximate inverses. *IEEE Trans. Autom. Control*, 1981.

[195] Yurui Zhang, Ruigang Wang, and Ian R. Manchester. Robustly invertible nonlinear dynamics and the BiLipREN: Contracting neural models with contracting inverses. *arXiv preprint arXiv:2505.03069*, 2025.

[196] Kemin Zhou, John C. Doyle, and Keith Glover. *Robust and optimal control*. Prentice-Hall, Inc., USA, 1996.

[197] Kemin Zhou and Zhang Ren. A new controller architecture for high performance, robust, and fault-tolerant control. *IEEE Transactions on Automatic Control*, 2001.