UNIVERSITY OF CALIFORNIA
SANTA CRUZ

A DEEP LEARNING FRAMEWORK FOR OPTIMAL FEEDBACK CONTROL OF HIGH-DIMENSIONAL NONLINEAR SYSTEMS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

APPLIED MATHEMATICS AND STATISTICS

by

Tenavi Erin Nakamura-Zimmerer

June 2022

The Dissertation of Tenavi Erin Nakamura-Zimmerer is approved:

Professor Qi Gong, Chair

Professor Wei Kang

Professor Daniele Venturi

Professor Abhishek Halder

Peter Biehl
Vice Provost and Dean of Graduate Studies

Contents

List of Figures

List of Tables

Abstract

A Deep Learning Framework for Optimal Feedback Control of High-Dimensional Nonlinear Systems

by

Tenavi Erin Nakamura-Zimmerer

Designing optimal feedback controllers for nonlinear dynamical systems requires solving Hamilton-Jacobi-Bellman equations, which are notoriously difficult when the state dimension is large. Existing strategies for optimal feedback design are usually not valid for high-dimensional problems, may rely on restrictive problem structures, or are valid only locally around some nominal trajectory. On the other hand, mature numerical methods exist for solving open loop optimal control problems, and these have been successfully deployed in a number of settings including the International Space Station. It is well-known, however, that open loop controls are not robust to model uncertainty or disturbances, so for real-time applications we need a closed loop feedback controller.

In this dissertation we develop a deep learning-based framework to synthesize optimal feedback controllers for high-dimensional nonlinear systems. The core idea is to train a neural network feedback controller on data generated by solving a set of open loop optimal control problems, which does not require state space discretization and is thus applicable in high dimensions. We also introduce several specialized neural network architectures which guarantee local stability by construction and can still accurately approximate the optimal control over large domains. Training is made more effective and data-efficient by leveraging the known physics of the problem and using the partially-trained neural network to aid in adaptive data generation. Data generation and model training, while

computationally demanding, are performed offline. Once trained, the neural network can be evaluated online at minimal cost, thus delivering real-time optimal feedback control.

We demonstrate the feasibility and effectiveness of the proposed control design framework on several nonlinear high-dimensional examples, including stabilization of unstable Burgers'-like partial differential equations, attitude control of a rigid body satellite with momentum wheels, and altitude and course tracking for an unmanned aircraft.

To my family: Helga, Q, Bobby, Terry, Cady, Perry, Joey.

Acknowledgements

Most of the work presented in this dissertation is adapted from previously published material [96, 97, 98, 99, 100, 101], with permission of the coauthors. The relevant papers are cited at the beginning of each chapter.

Anything I have accomplished during the course of graduate school - and in my life in general - is thanks to the support, inspiration, and love of many others.

First I would like to thank my advisor, Qi Gong, for six years of mentorship, support, and guidance. You have been patient, encouraging, and a great source of wisdom. I sincerely appreciate all that you have taught me and all that you have done to give me a joyous graduate school experience.

Thank you to my unofficial almost-co-advisor, Wei Kang, for many insightful discussions and words of encouragement. Thank you to Daniele Venturi for showing me the meaning of passion and curiosity for research. Thank you to Abhishek Halder for keeping my on my toes about the rigor of my work, and for your courses on control theory. Thank you also to Pascale Garaud for your caring and inspiration (both in science and on the rock).

I've been inspired and supported by so many great classmates in Applied Mathematics (and Statistics). I will not try to list them all for fear of failure, but let say a special thanks to Jacob Noone Wade, Isabelle Grenier, Lia Gianfortone, Paul Wintz, Tayler Quist, Bethany Johnson, George Labaria, and Yuanran Zhu. I look fondly on the beers, rumors, complaints, jokes, beers, and occasional math discussions we shared throughout the years. Thank you for making Applied Math a welcoming place to be. Of course the special-est thank you goes to legendary Sara Nasab. You have been a truly one of a kind friend and I am so lucky to have gotten to know you.

It would be a crime not to acknowledge the support and friendship of my roommates and quarantine companions, Alexandra Turmon, Allie Hyatt, and Corey Dickson. Thank you for keeping the house upright when I was submerged in work, for entertaining my demands to exercise together, and for making my house feel like a home.

Thank you to the wonderful friends I made before graduate school, Theo Muller, Ana Lisa Sutherland, Zav Hershfield, Anna Kenney, and Madeline Thompson, for continuing to put up with me all these years, keeping me humble, and reminding me how to have fun.

For almost the entirety of my time in graduate school I've had the incredible companionship of Van Lau. Thank you for the excitement, candor, and love you have brought to my life. Anh yêu em!

To my wacky extended nuclear family, Jane Naimark, Terry Erickson, Cady Shadwick, Perry Shadwick, and Joseph Stevenson: thank you so much for your love and support. You have all inspired me in one way or another and I'm so glad to have you in my life.

Finally I would like to thank my parents, Tomassa Nakamura and Helga Zimmerer. Qちゃん、深く愛して、色々教えてもらって、 素晴らしロールモデルをしてくれて、とても感謝する。愛しているよ。 Helga, du hast mir gezeigt was Liebe bedeutet und du warst immer meine größte Cheerleader. Ohne dich wäre ich nirgends hingekommen. Ich danke dir, und ich habe dich sehr lieb.

Chapter 1

Introduction

Control engineering is the practice of designing inputs to a dynamical system so that it behaves in some specified way. Controller design is important for just about every engineering problem. To name a few examples, control engineering is used to design autopilots for manned and unmanned aircraft; orbital transfer trajectories to take a spacecraft to the moon; and building heating and air conditioning systems which maintain a specified temperature.

Within control engineering, *optimal control* is a system for designing controllers by optimizing a performance metric. Optimal control laws guarantee stability and the best possible performance according to the chosen performance metric. Returning to the previous examples, optimal control could be used to design an autopilot which produces a smooth ride or enables acrobatic maneuvers; spacecraft trajectories which are fuel-efficient; or temperature regulators which balance energy efficiency with comfort. Besides these clear performance and efficiency benefits, optimal control can account for nonlinearities in the system dynamics,

unlike many commonly used control design strategies which require linearization.

A fairly generic optimal control problem can be posed mathematically as follows (more detailed and specific problem settings are introduced in Section 1.1):

$$\begin{cases} \underset{\mathbf{u}(\cdot)}{\text{minimize}} & \mathcal{J}\left[\mathbf{u}(\cdot)\right], \\ \text{subject to} & \dot{\mathbf{x}}(t) = \mathbf{f}(t, \mathbf{x}, \mathbf{u}), \\ & \boldsymbol{\pi}_{\min} \leq \boldsymbol{\pi}\left(t, \mathbf{x}(t), \mathbf{u}(t)\right) \leq \boldsymbol{\pi}_{\max}, \\ & \mathbf{e}_{\min} \leq \mathbf{e}\left(\mathbf{x}\left(t_f\right), \mathbf{u}\left(t_f\right)\right) \leq \mathbf{e}_{\max}. \end{cases} \tag{1.1}$$

Here $t \in [0, t_f]$ denotes time, $\mathbf{x} : [0, t_f] \to \mathbb{R}^n$ is the state, $\mathbf{u} : [0, t_f] \to \mathbb{R}^m$ is the control, and $\mathbf{f} : [0, t_f] \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is a nonlinear system of ordinary differential equations (ODEs) describing the system dynamics. We seek to optimize the cost functional $\mathcal{J}\left[\mathbf{u}(\cdot)\right]$ subject to these dynamics, path constraints $\boldsymbol{\pi}\left(\cdot\right)$, and endpoint constraints $\mathbf{e}\left(\cdot\right)$.

For any particular initial condition $\mathbf{x}(0) = \mathbf{x}_0$, the solution of (1.1) is written as $\mathbf{u}(t) = \mathbf{u}^*\left(t; \mathbf{x}_0\right)$. Observe that this optimal control is a function of time and independent of the current state. Such a control law is called *open loop*: the control input is pre-calculated for a given $\mathbf{x}_0$ and then implemented for a certain amount of time in order to achieve the stated objective. On the other hand, suppose we can obtain a solution of (1.1) for *all* initial conditions so that the control becomes a function of the state: $\mathbf{u}(t) = \mathbf{u}^*\left(t; \mathbf{x}(t)\right)$. This is called a *closed loop* or *feedback* control. In this case the controller takes measurements of the current system state and returns the optimal control input, allowing it to react to unexpected changes.

Well-developed numerical methods exist for solving open loop optimal control problems (OCPs), and these have been used successfully in numerous applications [121]. However, when an open loop control is applied to the real system, the state will deviate from the optimal path due to uncertainty and disturbances.

In principle, it may be possible to implement closed loop optimal control by repeatedly re-solving the open loop OCP [120, 123, 119]. Unfortunately, numerical methods for open loop optimal control are often too slow to deploy in real-time applications. The techniques of model predictive control (MPC) [53, 125] seek to make this approach practical in some settings by trading optimality for computational feasibility. The main idea behind MPC is, at each sampling time, to solve a *simplified* open loop OCP over a relatively short time horizon; under some conditions this yields system stability. MPC has traditionally only been useful for slow moving systems like chemical process control, but has more recently been applied to some faster dynamics such as autonomous driving [53, 125].

Despite these advances, one would prefer an *optimal* control in *explicit* feedback form, as feedback controllers can react to fast changes in the state and are inherently more robust to disturbances. In principle, optimal feedback controllers can be synthesized by solving a (discretized) *Hamilton-Jacobi-Bellman* (HJB) equation, a partial differential equation (PDE) in n spatial dimensions plus time. But the size of the discretized problem increases exponentially with n, making direct solution intractable for even moderately large problems. This is the so-called "curse of dimensionality" [10]. As a consequence, designing optimal feedback controllers for high-dimensional nonlinear systems remains an outstanding challenge for the control community.

In this dissertation we present a machine learning-based framework for solving HJB equations and designing optimal feedback controllers for high-dimensional nonlinear systems. The core idea is to solve a set of open loop OCPs at different points in space. These OCPs can be solved independently of one another without any spatial mesh, raising the possibility for use in high-dimensional systems. Through *Pontryagin's Minimum Principle* (PMP), the solutions of these

open loop OCPs are related to the characteristics of the solution to the HJB PDE, called the *value function*. Hence they serve as a data set which we use to train a neural network (NN) to approximate the value function, its gradient, or the optimal feedback control policy.

This *supervised learning* approach is inspired by the sparse grid characteristics method [66, 69, 67], in which open loop solutions are computed at sparse grid points, with interpolation being used to compute the closed loop solution. The use of NNs is motivated by their ability to approximate high-dimensional nonlinear functions, thus significantly increasing the feasible problem dimension as compared to sparse grids. The computationally expensive processes, data generation and NN training, are performed *offline*. Once trained, the NN can be evaluated *online* at minimal cost, thus delivering real-time optimal feedback control. In addition, with the development of modern specialized computing hardware, NNs can be implemented on small onboard processors. Through numerical simulations we demonstrate that the proposed framework yields nearly optimal feedback controllers and requires an order of magnitude less data than the sparse grid characteristics method to obtain equivalent results.

We enhance the core supervised learning algorithm with specialized NN architectures that guarantee, at a minimum, local asymptotic stability (LAS) of the desired objective states. These architectures smoothly combine a linear quadratic regulator (LQR), which provides local stability and optimality, with an NN that learns the nonlinear optimal feedback over a semi-global domain. These architectures make the learning algorithm more reliable and hence more practically useful.

Throughout the dissertation we demonstrate the feasibility and effectiveness of the proposed control design framework on several examples of dimension up to

$n = 64$, including stabilization of unstable Burgers'-like PDEs, attitude control of a rigid body satellite with momentum wheels, and altitude and course tracking for a six degree-of-freedom (6DoF) fixed-wing UAV. It is worth noting that it has only recently become possible to synthesize optimal feedback controllers for non-trivial, nonlinear, and high-dimensional OCPs such as these.

The proposed methods are *data-driven* but also *physics-informed*. At each step of optimal control synthesis we take advantage of ideas from control theory and knowledge of problem structure. At no point do we argue that machine learning is meant to replace rigorous control engineering practices. Instead, the proposed framework uses machine learning as a tool to enable the application of optimal control theory to problems which were historically computationally intractable.

The remainder of Chapter 1 is organized as follows. Section 1.1 introduces the OCP, PMP, the HJB equation, and LQR. Next, Section 1.2 contextualizes this work by discussing related efforts to solve the HJB equation and design optimal feedback controllers. Lastly Section 1.3 highlights the contributions of this dissertation and outlines the organization of remaining chapters.

1.1 Optimal control

In this dissertation we consider fixed final time OCPs for dynamical systems described by ODEs:

$$\begin{cases} \underset{\mathbf{u}(\cdot)}{\text{minimize}} & \mathcal{J}\left[\mathbf{u}(\cdot); \mathbf{x}_0\right] = F(\mathbf{x}(t_f)) + \displaystyle\int_0^{t_f} \mathcal{L}(t, \mathbf{x}, \mathbf{u}) dt, \\ \text{subject to} & \dot{\mathbf{x}}(t) = \mathbf{f}(t, \mathbf{x}, \mathbf{u}), \\ & \mathbf{x}(0) = \mathbf{x}_0, \\ & \mathbf{u}(t) \in \mathbb{U}. \end{cases} \tag{1.2}$$

Here $\mathbf{x} : [0, t_f] \to \mathbb{R}^n$ is the state, $\mathbf{u} : [0, t_f] \to \mathbb{U} \subseteq \mathbb{R}^m$ is the control, and $\mathbf{f} : [0, t_f] \times \mathbb{R}^n \times \mathbb{U} \to \mathbb{R}^n$ is a vector field which is continuously differentiable ($\mathcal{C}^1$) in $\mathbf{x}$ and $\mathbf{u}$. We consider box control constraints which can be expressed as

$$\mathbb{U} = \left\{ \mathbf{u} \in \mathbb{R}^m \,|\, u_{\min,i} \leq u_i \leq u_{\max,i}, i = 1, \ldots, m \right\}. \tag{1.3}$$

We seek to optimize the cost functional $\mathcal{J}\left[\mathbf{u}(\cdot); \mathbf{x}_0\right]$ which is composed of $F : \mathbb{R}^n \to \mathbb{R}$, the terminal cost, and $\mathcal{L} : [0, t_f] \times \mathbb{R}^n \times \mathbb{U} \to \mathbb{R}$, the running cost. We assume that the cost functional is smooth and convex in $\mathbf{x}$ and $\mathbf{u}$.

Much of the work in this dissertation concerns infinite horizon OCPs. Infinite horizon OCPs can be considered a special case of (1.2) with time-independent dynamics and running cost and where the final time $t_f \to \infty$. Infinite horizon optimal control provide a natural framework for solving stabilization or set-point tracking problems. We will consider problems of the form

$$\begin{cases} \underset{\mathbf{u}(\cdot)}{\text{minimize}} & \mathcal{J}\left[\mathbf{u}(\cdot); \mathbf{x}_0\right] = \displaystyle\int_0^\infty \mathcal{L}(\mathbf{x}, \mathbf{u}) dt, \\ \text{subject to} & \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}, \mathbf{u}), \\ & \mathbf{x}(0) = \mathbf{x}_0, \\ & \mathbf{u}(t) \in \mathbb{U}. \end{cases} \tag{1.4}$$

Notice that there is no terminal cost $F(\cdot)$, the running cost and dynamics are time-independent, and $t_f \to \infty$. We specifically consider running costs $\mathcal{L} : \mathbb{R}^n \times \mathbb{U} \to [0, \infty)$ of the form

$$\mathcal{L}(\mathbf{x}, \mathbf{u}) = q\left(\mathbf{x} - \mathbf{x}_f\right) + r\left(\mathbf{u} - \mathbf{u}_f\right), \tag{1.5}$$

where $(\mathbf{x}_f, \mathbf{u}_f) \in \mathbb{R}^n \times \mathbb{U}$ is a (possibly unstable) equilibrium of the dynamics such that $\mathbf{f}\left(\mathbf{x}_f, \mathbf{u}_f\right) = \mathbf{0}$. The running cost consists of a state cost $q : \mathbb{R}^n \to [0, \infty)$, which is a smooth, positive semi-definite function with $q(\mathbf{0}) = 0$, and a control

cost $r : \mathbb{R}^m \to [0, \infty)$, which is a smooth, positive definite function with $r(\mathbf{0}) = 0$. We make the standard assumptions that $\mathbf{u}_f$ is contained in an open subset of $\mathbb{U}$.

Assumption 1 (Well-posed OCPs)**.** *In this work we assume that the OCP* (1.2) *is well-posed and admits a unique solution*

$$\mathbf{u}(t) = \mathbf{u}^*(t; \mathbf{x}_0) \tag{1.6}$$

for each given initial condition $\mathbf{x}(0) = \mathbf{x}_0$*. We similarly assume that the infinite horizon OCP* (1.4) *is well-posed and has a unique solution. That is, there exists a unique* $\mathbf{u}^* : [0, \infty) \to \mathbb{U}$ *such that* $\mathcal{J}\left[\mathbf{u}^*(\cdot)\right] < \infty$ *and* $\lim_{t\to\infty} \mathcal{L}\left(\mathbf{x}^*(t), \mathbf{u}^*(t)\right) = 0$.

1.1.1 Pontryagin's Minimum Principle

In this section we present the well-known PMP, which gives necessary conditions for optimality of open loop OCPs. Derivations and further explanation of PMP can be found in e.g. [114, 83]. For the OCPs (1.2) and (1.4), PMP takes the form of a two-point ODE boundary value problem (BVP). This BVP must be simultaneously solved forward for the (locally) optimal state trajectory, $\mathbf{x}^* : [0, t_f] \to \mathbb{R}^n$, and *backwards* from $t = t_f$ to $t = 0$ for the *costate* $\boldsymbol{\lambda} : [0, t_f] \to \mathbb{R}^n$.

Let us start by defining the control *Hamiltonian*,

$$\mathcal{H}(t, \mathbf{x}, \boldsymbol{\lambda}, \mathbf{u}) \coloneqq \mathcal{L}(t, \mathbf{x}, \mathbf{u}) + \boldsymbol{\lambda}^T \mathbf{f}(t, \mathbf{x}, \mathbf{u}). \tag{1.7}$$

For any fixed initial condition, $\mathbf{x}_0$, it is known that the optimal control must satisfy the Hamiltonian minimization condition for almost all $t \in [0, t_f]$:

$$\mathbf{u}^*\left(t; \mathbf{x}_0\right) = \mathbf{u}^*\left(t; \mathbf{x}\left(t; \mathbf{x}_0\right), \boldsymbol{\lambda}\left(t; \mathbf{x}_0\right)\right) = \arg\min_{\mathbf{u}\in\mathbb{U}} \mathcal{H}\left(t, \mathbf{x}, \boldsymbol{\lambda}, \mathbf{u}\right). \tag{1.8}$$

Substituting this back into (1.7) yields the optimized Hamiltonian,

$$\mathcal{H}^*(t, \mathbf{x}, \boldsymbol{\lambda}) \coloneqq \mathcal{H}\left(t, \mathbf{x}, \boldsymbol{\lambda}, \mathbf{u}^*(t; \mathbf{x}, \boldsymbol{\lambda})\right). \tag{1.9}$$

Now according to PMP [114, 83], a necessary condition for optimality of the open loop control is that the state and costate satisfy the following two-point BVP:

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathcal{H}^*_{\boldsymbol{\lambda}}(t, \mathbf{x}, \boldsymbol{\lambda}) = \mathbf{f}(t, \mathbf{x}, \mathbf{u}^*(t, ; \mathbf{x}, \boldsymbol{\lambda})), & \mathbf{x}(0) = \mathbf{x}_0, \\ \dot{\boldsymbol{\lambda}}(t) = -\mathcal{H}^*_{\mathbf{x}}(t, \mathbf{x}, \boldsymbol{\lambda}, \mathbf{u}^*(t, ; \mathbf{x}\boldsymbol{\lambda})), & \boldsymbol{\lambda}\left(t_f\right) = F_{\mathbf{x}}\left(\mathbf{x}\left(t_f\right)\right). \end{cases} \tag{1.10}$$

Here and throughout this dissertation we often denote the gradient of a scalar-valued function $f(\cdot)$ as $f_{\mathbf{z}} \coloneqq [\partial f / \partial \mathbf{z}]^T$, where $f(\cdot)$ is a function of some $\mathbf{z}$ and possibly other variables.

For the infinite horizon OCP (1.4) the Hamiltonian becomes time-independent:

$$\mathcal{H}(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{u}) \coloneqq \mathcal{L}(\mathbf{x}, \mathbf{u}) + \boldsymbol{\lambda}^T \mathbf{f}(\mathbf{x}, \mathbf{u}). \tag{1.11}$$

In this setting the optimal control satisfies

$$\mathbf{u}^*\left(t; \mathbf{x}_0\right) = \mathbf{u}^*\left(\mathbf{x}\left(t; \mathbf{x}_0\right), \boldsymbol{\lambda}\left(t; \mathbf{x}_0\right)\right) = \arg\min_{\mathbf{u} \in \mathbb{U}} \mathcal{H}\left(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{u}\right). \tag{1.12}$$

and the two-point BVP is solved over the infinite interval [114]:

$$\lim_{t_f \to \infty} \begin{cases} \dot{\mathbf{x}}(t) = \mathcal{H}^*_{\boldsymbol{\lambda}}(\mathbf{x}, \boldsymbol{\lambda}) = \mathbf{f}(\mathbf{x}, \mathbf{u}^*(\mathbf{x}, \boldsymbol{\lambda})), & \mathbf{x}(0) = \mathbf{x}_0, \\ \dot{\boldsymbol{\lambda}}(t) = -\mathcal{H}^*_{\mathbf{x}}(\mathbf{x}, \boldsymbol{\lambda}), & \boldsymbol{\lambda}\left(t_f\right) = \mathbf{0}. \end{cases} \tag{1.13}$$

In general, both BVPs (1.10) and (1.13) admit multiple solutions, some of which may be locally optimal in the space of admissible controls. In many problems it is also possible for BVP solutions to intersect in finite time, giving rise

to non-smooth solutions of the HJB equation and a number of computational challenges.

Optimality of solutions to (1.10) and (1.13) can be guaranteed under some convexity conditions (see e.g. [92]). Unfortunately for most dynamical systems it is difficult to verify such conditions globally, but we can guarantee optimality locally around an equilibrium point [88]. Fully addressing this challenge is beyond the scope of the present work, so we assume that solutions of (1.10) and (1.13) are globally optimal. Under this assumption, the relationship between PMP and the value function discussed in Section 1.1.3 holds everywhere.

Assumption 2 (Solutions to PMP are globally optimal)**.** *If* $(\mathbf{x}(t), \boldsymbol{\lambda}(t))$ *satisfy* (1.10) *then* $\mathbf{u}^*(t) = \mathbf{u}^*(t; \mathbf{x}(t), \boldsymbol{\lambda}(t))$ *is the global minimizer of* (1.2)*. Likewise if* $(\mathbf{x}(t), \boldsymbol{\lambda}(t))$ *satisfy* (1.13)*, then* $\mathbf{u}^*(t) = \mathbf{u}^*(\mathbf{x}(t); \boldsymbol{\lambda}(t))$ *is the global minimizer of* (1.4)*.*

We note that supervised learning approaches based on PMP can still be applied even when Assumption 2 cannot be verified. In such cases PMP remains the prevailing tool for finding *candidate optimal* solutions [119]. From these the proposed methods yield stabilizing feedback controllers which satisfy necessary conditions for local optimality.

1.1.2 The Hamilton-Jacobi-Bellman equation

The open loop optimal control (1.6) which solves (1.2) is valid for all $t \in [0, t_f]$, but only for the fixed initial condition $\mathbf{x}(0) = \mathbf{x}_0$. Due to various sources of disturbance and real-time application requirements, for practical implementation

one typically desires a control in closed loop feedback form,

$$\mathbf{u}(t) = \mathbf{u}^*(t, \mathbf{x}(t)). \tag{1.14}$$

Such a control policy can be evaluated online at any $t \in [0, t_f]$ and for any measurement of $\mathbf{x} \in \mathbb{R}^n$, irrespective of the initial condition $\mathbf{x}_0$. In other words, the same function $\mathbf{u}^*(\cdot)$ can be used even if the trajectory $\mathbf{x}(t; \mathbf{x}_0)$ deviates from the optimal path $\mathbf{x}^*(t; \mathbf{x}_0)$. For the infinite horizon OCP (1.4), it is easy to see that the optimal feedback control $\mathbf{u}^*(\cdot)$ must also be time-invariant, i.e.

$$\mathbf{u}(t) = \mathbf{u}^*(\mathbf{x}(t)). \tag{1.15}$$

The mathematical framework for designing an optimal feedback control policy (1.14) is the HJB PDE. Following the standard procedure in optimal control (see e.g. [83]) we define the *value function* $V : [0, t_f] \times \mathbb{R}^n \to \mathbb{R}$ as the optimal cost-to-go of (1.2) starting at (with some abuse of notation) $\mathbf{x}(t) = \mathbf{x}$. That is,

$$V(t, \mathbf{x}) \coloneqq \mathcal{J}[\mathbf{u}^*(\cdot); \mathbf{x}] = \begin{cases} \inf\limits_{\mathbf{u}(\cdot)} & F(\mathbf{y}(t_f)) + \displaystyle\int_t^{t_f} \mathcal{L}(s, \mathbf{y}, \mathbf{u}) ds, \\ \text{s.t.} & \dot{\mathbf{y}}(s) = \mathbf{f}(s, \mathbf{y}, \mathbf{u}), \\ & \mathbf{y}(t) = \mathbf{x}, \\ & \mathbf{u}(s) \in \mathbb{U}. \end{cases} \tag{1.16}$$

Under appropriate conditions, the value function is the unique viscosity solution [26] of the HJB PDE,

$$\begin{cases} -V_t(t, \mathbf{x}) - \min_{\mathbf{u} \in \mathbb{U}} \mathcal{H}(t, \mathbf{x}, V_{\mathbf{x}}, \mathbf{u}) = 0, \\ V(t_f, \mathbf{x}) = F(\mathbf{x}). \end{cases} \tag{1.17}$$

If (1.17) can be solved (in the viscosity sense), then it provides both necessary

and sufficient conditions for optimality. Furthermore, the optimal feedback control (1.14) is computed from the Hamiltonian minimization condition

$$\mathbf{u}^*(t,\mathbf{x}) = \mathbf{u}^*\left(t,\mathbf{x};V_{\mathbf{x}}(t,\mathbf{x})\right) = \arg\min_{\mathbf{u}\in\mathbb{U}} \mathcal{H}\left(t,\mathbf{x},V_{\mathbf{x}},\mathbf{u}\right). \tag{1.18}$$

This means that with $V_{\mathbf{x}}(\cdot)$ available, the optimal feedback control is obtained as the solution of an (ideally straightforward) optimization problem. The challenge, as alluded to earlier, is solving (1.17) in the first place.

For the infinite horizon OCP (1.4) the value function is defined as

$$V(\mathbf{x}) \coloneqq \mathcal{J}\left[\mathbf{u}^*(\cdot);\mathbf{x}\right] = \begin{cases} \inf\limits_{\mathbf{u}(\cdot)} & \int_t^{t_f} \mathcal{L}(\mathbf{y},\mathbf{u})ds, \\ \text{s.t.} & \dot{\mathbf{y}}(s) = \mathbf{f}(\mathbf{y},\mathbf{u}), \\ & \mathbf{y}(t) = \mathbf{x}, \\ & \mathbf{u}(s) \in \mathbb{U}. \end{cases} \tag{1.19}$$

This time-independent value function satisfies the following steady state HJB PDE:

$$\begin{cases} \min\limits_{\mathbf{u}\in\mathbb{U}} \mathcal{H}\left(\mathbf{x},V_{\mathbf{x}},\mathbf{u}\right) = 0, \\ V\left(\mathbf{x}_f\right) = 0, \end{cases} \tag{1.20}$$

where the Hamiltonian is defined as in (1.11). As with the finite horizon case, if (1.20) can be solved (in the viscosity sense), then it provides both necessary and sufficient conditions for optimality, and the optimal feedback control is obtained from the Hamiltonian minimization condition

$$\mathbf{u}^*(\mathbf{x}) = \mathbf{u}^*\left(\mathbf{x};V_{\mathbf{x}}(\mathbf{x})\right) = \arg\min_{\mathbf{u}\in\mathbb{U}} \mathcal{H}\left(\mathbf{x},V_{\mathbf{x}},\mathbf{u}\right). \tag{1.21}$$

Besides the curse of dimensionality, a well known challenge when solving the HJB equations (1.17) and (1.20) is non-smoothness of the value function. In

general (1.17) and (1.20) may only have viscosity solution [26]. For the kinds of problems we have studied in this work non-smoothness does not appear to be a serious concern, but it can be hard to determine *a priori* whether solutions will be smooth. For this reason we make the following common assumption.

Assumption 3 (Existence of classical solutions)**.** *The HJB equations* (1.17) *and* (1.20) *admit unique* $\mathcal{C}^1$ *solutions* $V : [0, t_f] \times \mathbb{R}^n \to \mathbb{R}$ *and* $V : \mathbb{R}^n \to \mathbb{R}$*, respectively.*

1.1.3 Relationship between PMP and HJB

The critical idea underlying our supervised learning approach is that the solutions of the open loop OCP obtained via PMP are related to the closed loop optimal control and value function. In particular, under Assumptions 2 and 3, the controlled trajectory $\mathbf{x}^*(t; \mathbf{x}_0)$ is a *characteristic* of the value function $V(\cdot)$, and at each point $\mathbf{x} = \mathbf{x}^*(t; \mathbf{x}_0)$ along the characteristic we have

$$\begin{cases} V(t, \mathbf{x}) = F\left(\mathbf{x}^*\left(t_f; \mathbf{x}_0\right)\right) + \displaystyle\int_t^{t_f} \mathcal{L}\left(s, \mathbf{x}^*(s), \mathbf{u}^*(s)\right) ds, \\ V_{\mathbf{x}}(t, \mathbf{x}) = \boldsymbol{\lambda}\left(t; \mathbf{x}_0\right), \\ \mathbf{u}^*(t, \mathbf{x}) = \mathbf{u}^*\left(t; \mathbf{x}_0\right). \end{cases} \tag{1.22}$$

Likewise in the infinite horizon setting, along the trajectory $\mathbf{x} = \mathbf{x}^*\left(t; \mathbf{x}_0\right)$ we have

$$\begin{cases} V(\mathbf{x}) = \displaystyle\int_t^{\infty} \mathcal{L}\left(\mathbf{x}^*(s), \mathbf{u}^*(s)\right) ds, \\ V_{\mathbf{x}}(\mathbf{x}) = \boldsymbol{\lambda}\left(t; \mathbf{x}_0\right), \\ \mathbf{u}^*(\mathbf{x}) = \mathbf{u}^*\left(t; \mathbf{x}_0\right). \end{cases} \tag{1.23}$$

In the finite horizon case, (1.10) describes the evolution of the characteristics of the time-dependent HJB equation (1.17). While the stationary HJB equation (1.20) does not have characteristics in the same sense, by viewing (1.20) as the

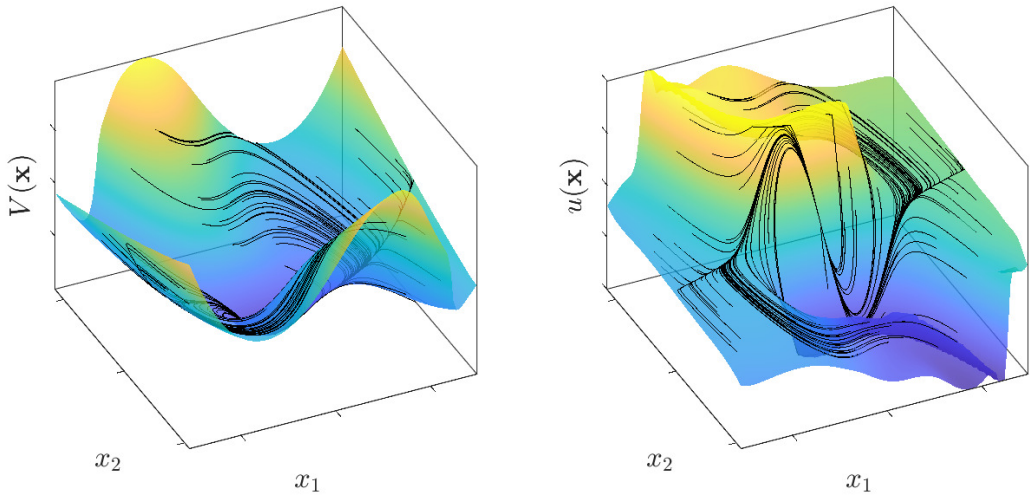


Figure 1.1: Solutions of PMP (black curves) superimposed on the value function (left) and optimal control (right) for a prototypical OCP.

infinite horizon limit of the usual time-dependent HJB equation (1.17), we can see that it maintains the same relationship with the infinite horizon BVP (1.13). This relationship is depicted for a prototypical OCP in Figure 1.1.

Eqs. (1.22) and (1.23) contain an important relationship between PMP and the value function: if the solution to PMP is optimal then the costate is equal to the value gradient along the characteristic, i.e. $V_{\mathbf{x}}(t, \mathbf{x}) = \boldsymbol{\lambda}(t)$. We will exploit this relationship when modeling the value function.

1.1.4 The linear quadratic case

The LQR problem is a well-known special case of the (infinite horizon) OCP (1.4) with linear dynamics and a quadratic cost function. The LQR problem commonly arises from the linearization of the nonlinear dynamics about a desired equilibrium. This linear control design technique yields a simplified form of the HJB equation (1.20) and an LAS and locally optimal *linear* control law. Due

to the difficulty in solving the full nonlinear HJB equation, control design by linearization is a standard engineering solution. LQR also forms a key component of the NN-based controllers developed in this dissertation. For these reasons in this section we provide a brief overview of LQR design. We restrict our discussion to the infinite horizon case, though one can also design a time-dependent LQR. Accessible derivations of the time-independent and time-dependent LQR can be found in e.g. [83, 130].

Given a nonlinear dynamical system $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ with equilibrium $(\mathbf{x}_f, \mathbf{u}_f)$, to compute the LQR controller we linearize the dynamics about the equilibrium to obtain

$$\begin{cases} \dot{\mathbf{x}} \approx \mathbf{A}(\mathbf{x} - \mathbf{x}_f) + \mathbf{B}(\mathbf{u} - \mathbf{u}_f), \\ \mathbf{A} \coloneqq \dfrac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}_f, \mathbf{u}_f), \\ \mathbf{B} \coloneqq \dfrac{\partial \mathbf{f}}{\partial \mathbf{u}}(\mathbf{x}_f, \mathbf{u}_f). \end{cases} \tag{1.24}$$

Similarly, we make a quadratic approximation of the running cost (1.5):

$$\begin{cases} \mathcal{L}(\mathbf{x}, \mathbf{u}) \approx (\mathbf{x} - \mathbf{x}_f)^T \mathbf{Q}(\mathbf{x} - \mathbf{x}_f) + (\mathbf{u} - \mathbf{u}_f)^T \mathbf{R}(\mathbf{u} - \mathbf{u}_f), \\ \mathbf{Q} \coloneqq \dfrac{\partial^2 q}{\partial \mathbf{x}^2}(\mathbf{0}), \\ \mathbf{R} \coloneqq \dfrac{\partial^2 r}{\partial \mathbf{u}^2}(\mathbf{0}). \end{cases} \tag{1.25}$$

Under the standard conditions that $(\mathbf{A}, \mathbf{B})$ is controllable and $\left(\mathbf{A}, \mathbf{Q}^{1/2}\right)$ is observable [130], we get a quadratic value function approximation

$$V^{\mathrm{LQR}}(\mathbf{x}) = (\mathbf{x} - \mathbf{x}_f)^T \mathbf{P} (\mathbf{x} - \mathbf{x}_f) \tag{1.26}$$

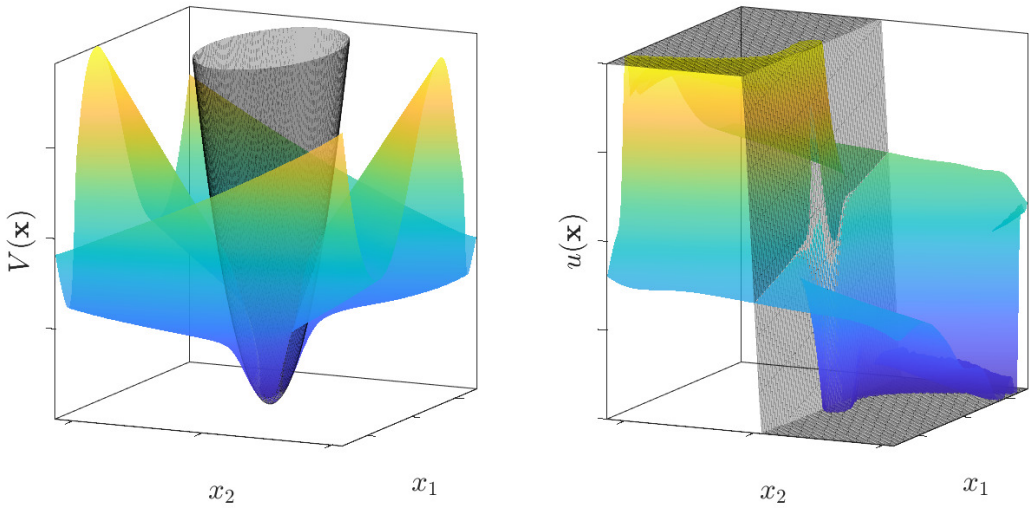


Figure 1.2: Prototypical example of LQR approximation (black) and full nonlinear solution (color). Left: value function. Right: optimal feedback policy. Notice the close agreement near the origin.

and a linear control law

$$\mathbf{u}^{\mathrm{LQR}}(\mathbf{x}) = \mathbf{u}_f - \mathbf{K}(\mathbf{x} - \mathbf{x}_f), \quad \mathbf{K} = \mathbf{R}^{-1}\mathbf{B}^T\mathbf{P}, \tag{1.27}$$

where $\mathbf{P} \in \mathbb{R}^{n \times n}$ is a positive definite matrix satisfying the following continuous algebraic Riccati equation:

$$\mathbf{Q} + \mathbf{A}^T\mathbf{P} + \mathbf{P}\mathbf{A} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P} = \mathbf{0}. \tag{1.28}$$

Mature numerical methods for solving the Riccati equation (1.28) have been implemented in a variety of software packages. In this dissertation we use *SciPy*'s `linalg.solve_continuous_are` [140, 80, 138, 11].

Sufficiently near the equilibrium $\mathbf{x}_f$, the LQR value function $V^{\mathrm{LQR}}(\cdot)$ and linear controller $\mathbf{u}^{\mathrm{LQR}}(\cdot)$ are good approximations of the true value function $V(\cdot)$ and optimal control $\mathbf{u}^*(\cdot)$. In fact it can be shown (see Lemma 3 adapted from [88])

$\frac{\partial \mathbf{u}^*}{\partial \mathbf{x}}(\mathbf{x}_f) = -\mathbf{K}$ and $\frac{\partial^2 V}{\partial \mathbf{x}^2}(\mathbf{x}_f) = 2\mathbf{P}$; a prototypical case is illustrated in Figure 1.2. . Furthermore, LQR has excellent gain and phase margins [145, 130, 44] making it highly robust to certain system perturbations. These properties make LQR an effective and popular linear control design tool.

But further away from $\mathbf{x}_f$, the LQR control is suboptimal and in some cases may even fail to stabilize the nonlinear dynamics. This motivates us to combine the LAS and local optimality of LQR with deep learning to construct the full *nonlinear* optimal feedback control $\mathbf{u}^*(\cdot)$ over a semi-global domain.

1.2 Related work

The longstanding bottleneck for designing optimal feedback controllers is the need to solve a high-dimensional HJB PDE (1.17) or (1.20). Unfortunately, traditional finite discretization approaches do not work for even moderately large systems. This is because the size of the discretized problem increases exponentially with the state dimension, n, making such direct solution methods computationally intractable. As such, the HJB equation is traditionally considered very difficult in just four or more dimensions. Indeed, Richard Bellman coined the term "curse of dimensionality" in reference to the difficulty of solving high-dimensional dynamic programming problems [10].

For this reason, there is an extensive literature on numerical methods of finding approximate solutions for nonlinear HJB equations. Outside of mesh-based methods, some key examples include series expansions [4, 88, 63, 102, 16], level set methods [107], patchy dynamic programming [103, 20], semi-Lagrangian methods [15, 42, 5], method of characteristics and Hopf formula-based algorithms [30, 149, 25], bespoke NN architectures for certain restricted kinds of OCPs

[28, 29], low rank tensor methods [109, 37], and model predictive control based on trajectory databases [110]. Unfortunately, most of these existing approaches suffer from one or more of the following drawbacks: they can only handle problems of limited dimension; the solution may be valid only in a small region; or the system model must have certain special algebraic structure.

The present work is largely inspired by the sparse grid characteristics method developed in [66, 69, 67]. In this approach, semi-global solutions to HJB equations (1.17) are computed by constructing a sparse discretization of the state space, then solving a two-point BVP (1.10) at each point on the grid. Once the solution at each grid point is available, feedback control is computed online by interpolation. Since these BVPs can be solved independently of one another we call them *causality-free* [66]. Causality-free algorithms are attractive because the computation does not depend on a grid, and hence they can be applied to high-dimensional problems. They are also embarrassingly parallelizable so can be used to generate large data sets offline. Such data sets can then be used to construct faster solutions such as sparse grid interpolation [66, 69, 67]. However, for this approach the final solution requires a BVP to be solved for each point in the sparse grid, the size of which grows like $O\left(N(\log N)^{n-1}\right)$, where n is the state dimension and N is the number of grid points in each dimension [67]. Consequently, one may have to solve a prohibitively large number of BVPs for higher-dimensional problems, limiting these methods to problems of moderately high dimension. This motivates us to replace the sparse grids with NNs.

The framework outlined in this dissertation is based on supervised learning. The main idea is to generate data by solving many open loop OCPs and then fit a model to this data set, thus obtaining an approximate optimal feedback controller. Around the same time that the first main results in this work were

obtained, similar research was published by another group [56, 55]. Other related efforts circumvent solving the HJB equation and directly learn the optimal control policy based on data [124, 133, 82]. Some of the algorithms presented in this dissertation also take this route. The main distinctions between these works and our methodology lie in data generation and in NN architecture. Notably, our NN architectures are specifically designed to guarantee at least LAS. Other authors have proposed variations of the supervised learning approach. For example, [21] propose an an active learning method based on [97] to learn the value gradient (instead of the value function); [6] and [34] use sparse polynomials and tensor trains, respectively, instead of NNs to approximate the value function; and [3, 34] use supervised learning to design suboptimal feedback laws based on state-dependent Riccati equations.

Using NNs as a basis for solving HJB equations and optimal feedback design is not by itself a new idea, though specific architectures training algorithms vary greatly. For example, [58] train an NN to approximate the control in small regions around a nominal trajectory. Many NN-based methods attempt to solve the HJB PDE in the least-squares sense by minimizing the residual of the PDE and boundary conditions at randomly sampled collocation points [23, 134, 129, 90, 93]. This approach has had some success, but requires the expensive computation of PDE residuals during NN training and, often, the creation of artifical boundary conditions. The least squares method can be considered a kind of *physics-informed neural network* (PINN) [116], which can often be rather challenging to train [75, 144, 143]. More recently, [51, 52, 105, 78] have proposed methods to solve the HJB equation along its characteristics without generating data. These approaches are closely related to promising recent research on deep learning for *explicit MPC* [35, 94, 128]. All of these learning-based methods could be classified as *self-*

supervised: they avoid generating data by taking on a harder learning problem.

Lastly we mention the method of successive approximations (SA) for continuous systems [9, 2, 77], see also e.g. [5, 62, 108, 33, 12] for recent implementations. SA is a well-studied approach based on iterative updates of a value function model and/or control policy by approximately solving a series of Lyapunov equations. These methods are equipped with convergence guarantees but they often depend on specific problem dynamics, exponentially discounted future costs, *a priori* access to a semi-globally stabilizing controller, or polynomial model structures whose size grow exponentially with the problem dimension. The method of SA forms the core of the related field of approximate dynamic programming (ADP), also known as adaptive dynamic programming, neuro-dynamic programming, and reinforcement learning [13, 95, 115, 84]. Some ADP methods have the impressive property that they can be used when the dynamics are unknown, partially known, or uncertain. But due to various prohibitive assumptions or exponential growth of the model size, we believe the claim that ADP "[solves] the curses of dimensionality" [115] is exaggerated. In addition, in this work we are more concerned with situations where no globally stabilizing controller is initially available, performance requirements are strict, and the dynamic model is good but nonlinear and high-dimensional.

1.3 Contributions of this dissertation

In this dissertation we develop a computational framework for solving HJB equations and synthesizing optimal feedback controllers for high-dimensional nonlinear systems. The proposed framework is based around supervised learning, where we fit an NN model to open loop optimal control data generated through PMP. Data

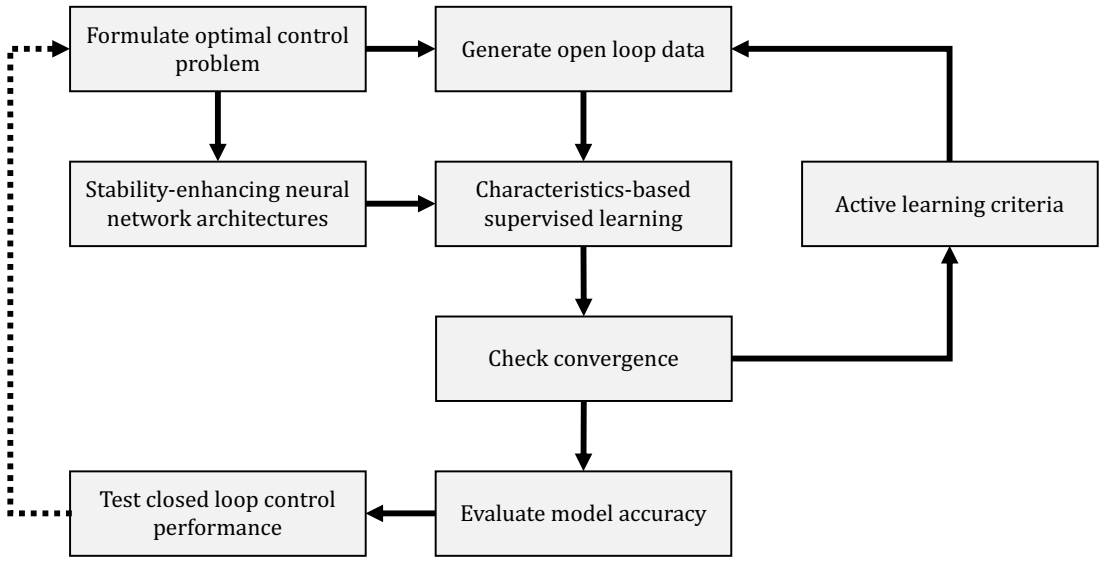


Figure 1.3: Summary of the of the steps and components of the proposed computational framework. Note that the outer loop connecting closed loop testing with problem formulation is a standard part of any control design process.

generation and NN training are performed offline; once trained the NN can be loaded onto onboard systems to implement real-time optimal control. A block diagram summarizing the steps and components of the proposed framework is presented in Figure 1.3. We will revisit this chart at the beginning of each chapter to orient the reader as to the focus of that chapter.

The proposed computational framework uniquely marries deep learning with control theory. We take full advantage of a known system model and existing algorithms for open loop optimical control to generate data. Deep learning bridges the gap between PMP and HJB, providing a powerful computational tool to implement the method of characteristics. Knowledge of the problem structure is used here to make learning more data-efficient. Finally, control theory again comes into play when we modify NN architectures to incorporate LQR controllers, locally recovering the LQR feedback and ensuring LAS.

Throughout the dissertation we demonstrate the potential capabilities of the

framework through numerical examples of several challenging nonlinear OCPs of dimension up to $n = 64$. This suggests that the proposed method scales to high-dimensional settings where optimal feedback control was traditionally thought to be infeasible. Furthermore, any computational burden associated with an increase in dimensionality or complexity is incurred entirely offline. In fact, due to the structure of NNs, increasing the dimension has a negligible impact on the speed of online control evaluation.

In Chapter 2 we outline the core supervised learning algorithm: data set construction, NN training, model evaluation on independent test data, and implementation in the closed loop system. We can choose to model the value function $V(\cdot)$, its gradient $V_{\mathbf{x}}(\cdot)$, or directly approximate the optimal control $\mathbf{u}^*(\cdot)$. A key detail here is that when training an NN value function model, the same model simultaneously learns the value gradient and/or optimal control. This extracts more information from the same data set, as well as helps the NN learn the shape of the value function rather than just fitting points. To illustrate this, in Chapter 2 we apply the core method to design an attitude controller of a rigid-body satellite equipped with momentum wheels. This is a highly nonlinear problem with $n = 6$ spatial dimensions and $m = 3$ control inputs. With the proposed method, we can obtain a model of the value function with accuracy comparable to the sparse grid characteristics method [67] while requiring an order of magnitude fewer data points.

Data is perhaps the most important aspect of supervised learning. A common saying goes "a model is only as good as the data it is trained on." With this in mind we devote Chapter 3 to discussing strategies for solving open loop OCPs for the purpose of data generation. We review the two important classes of open loops solvers, *direct* and *indirect* methods. Both solvers - in particular indirect methods

- benefit from good initial guesses, and we give several strategies for generating such guesses: *time marching* [66, 69, 67], *LQR warm start*, and *NN warm start*. In addition, we propose a numerical test to estimate the number of data points needed for model convergence. Together with NN warm start and a heuristic for selecting new initial points, this forms a framework for *adaptive* NN training. We conclude Chapter 3 by applying the proposed methods to the attitude control problem and OCPs of dimension $n = 10$, 20, and 30, arising from pseudospectral (PS) discretization of an unstable Burgers'-like PDE studied by [62].

In Chapter 4 we demonstrate that, despite these promising results, incorporating an NN controller into a nonlinear system can give rise to unexpected and destabilizing behavior. We use numerical simulations and some theoretical analysis to show that standard machine learning test accuracy metrics do not fully characterize closed loop stability and performance. Thus for the purpose of controller evaluation we apply several numerical tests of closed loop stability and optimality. Through these tests we see that NNs which are stable provide excellent performance commensurate with their approximation accuracy, but good approximation accuracy does not always imply that that the controller will be stabilizing.

In Chapter 5 we consider one approach for enhancing closed loop stability by building this into the NN architecture itself. Observing that much of the instability observed in Chapter 4 is a result of local instability, we propose specialized NN architectures that smoothly combine an LQR with an NN. This is in contrast to common practice in which linear and nonlinear parts of the control are often treated separately and joined at a later stage. We collectively refer to the proposed architectures as *QRnet*. Some of these novel architectures guarantee at least LAS by construction. We also prove that they retain the approximation capabilities

of standard NNs, thus allowing them to learn the full nonlinear optimal feedback over semi-global domains. We evaluate the approximation capacity, training time, stability, and optimality of the proposed controllers on two example problems: a 64 dimensional OCPs derived from PS discretization of an unstable Burgers' PDE, and an eleven dimensional model of a 6DoF fixed-wing UAV with nonlinear aerodynamics.

We summarize the proposed framework and its potential capabilities in Chapter 6. Here we also discuss outstanding challenges, limitations, promising areas for future research, and closely-related ideas.

Proof of concept software packages for implementing the proposed computational framework are made publicly available in the following online repositories:

- https://github.com/Tenavi/HJB_NN

  Implements methods for solving finite horizon HJB equations and active learning (see Chapters 2 and 3).

- https://github.com/Tenavi/QRnet

  A more refined and complete package for solving infinite horizon HJB equations and designing optimal feedback controllers with LAS guarantees (see Chapters 2, 4 and 5).

- https://github.com/Tenavi/PyLGR

  A python implementation of a Legendre-Gauss-Radau PS method [120, 41] for solving open loop infinite horizon OCPs (see Chapters 3 and 5).

Chapter 2

Solving Hamilton-Jacobi-Bellman Equations with Deep Learning

Computing optimal feedback controls for nonlinear systems generally requires solving HJB equations, which are notoriously difficult when the state dimension is large. Existing strategies for high-dimensional problems often rely on specific, restrictive problem structures, or are valid only locally around some nominal trajectory. In this chapter we introduce a *physics-informed* supervised learning algorithm for solving HJB equations. This algorithm serves as the core of the optimal feedback design methodology developed in this dissertation.

Figure 2.1 highlights the content of Chapter 2 within the greater computational framework. This chapter is structured as follows. In Section 2.1 we give an overview of the core algorithm and some of its potential advantages. In Section 2.2 we present a high level discussion of the data generation process; practical details are given later in Chapter 3. In Section 2.3 we introduce the concept of a

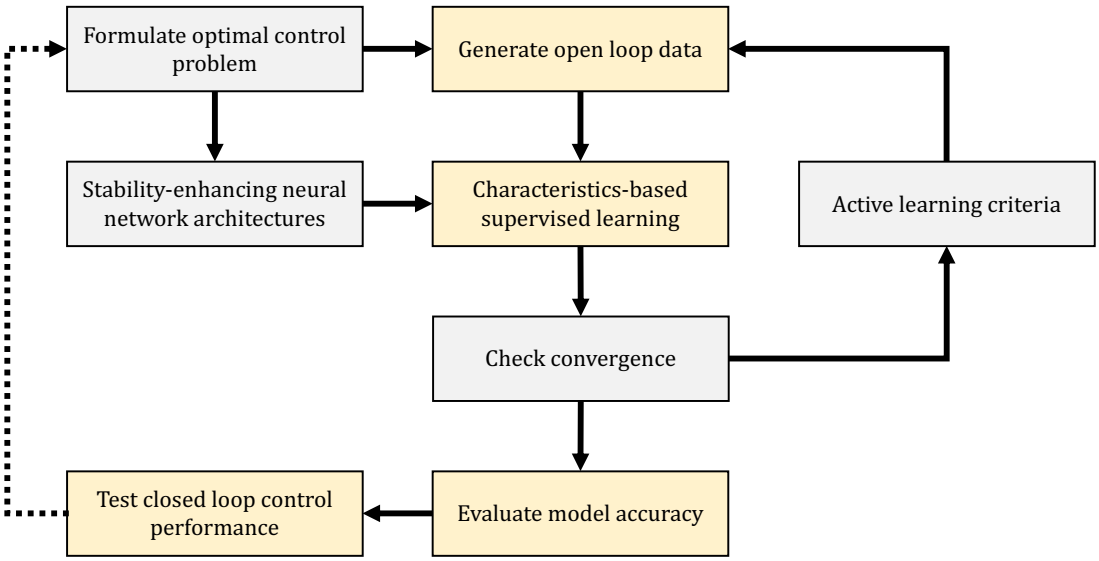


Figure 2.1: Summary of the of the steps and components of the proposed computational framework, highlighting the focus of Chapter 2.

feedforward NN and explain how to fit them to the open loop OCP data. Finally in Section 2.4 compare the proposed algorithm to the sparse grid characteristics method [67] on an attitude control design problem. A summary is given in Section 2.5.

The material presented in this chapter is drawn largely from [96, 97, 98]. It is presented in the context of the finite horizon OCP (1.2), but is equally applicable to the infinite horizon problem (1.4) which are the focus of Chapters 4 and 5.

2.1 Overview

Inspired by the promising results of the sparse grid characteristics method [66, 69, 67] we seek to avoid the curse of dimensionality by applying PMP to generate data along the characteristics of the value function. By replacing sparse grids with NNs we completely remove the dependence on a spatial grid, thus drastically reducing the number of data points needed to synthesize a value function approximation.

A further motivation for this approach is that NNs are effective for approximating very high-dimensional functions, further mitigating the curse of dimensionality. In later chapters we will demonstrate that this indeed greatly increases the feasible problem dimension relative to sparse grids.

The core algorithm consists of three steps. The first is to generate *two* independent data sets of open loop OCP solutions through PMP (1.10). These data sets are constructed by solving open loop OCPs for independently sampled sets of initial conditions. The next main step is to construct an NN model of the value function,

$$\widehat{V}(t, \mathbf{x}) = \mathcal{N}(t, \mathbf{x}; \boldsymbol{\theta}) \approx V(t, \mathbf{x}), \tag{2.1}$$

where $\mathcal{N} : [0, t_f] \times \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}$ is an NN (see Section 2.3.1) parameterized by $\boldsymbol{\theta} \in \mathbb{R}^p$. We train (2.1) by regression on the first of the two data sets (called the *training data*). Training is made more effective and data-efficient by leveraging the known physics of the problem. Specifically, we simultaneously learn the value function and its gradient based on the costate data, exploiting the relationship $V_{\mathbf{x}}(t, \mathbf{x}) = \boldsymbol{\lambda}(t)$ along the characteristics. This helps the NN to learn the shape of the value function instead of just fitting point data. Finally, the second data set (called the *test data*) is used to evaluate the model's *generalization performance* at unseen points. Good test accuracy indicates that the model did not overfit to the training data.

Although data generation, model training, and thorough testing are computationally expensive, these steps are all performed offline. Once trained, the NN can be deployed to make very fast control evaluations, potentially enabling use of the proposed methods in real-time applications. This is accomplished inserting the gradient of the value function model, $\widehat{V}_{\mathbf{x}}(\cdot)$, into (1.18) to obtain an approximate

optimal feedback law:

$$\widehat{\mathbf{u}}(t,\mathbf{x}) = \mathbf{u}^*\left(t,\mathbf{x};\widehat{V}_{\mathbf{x}}(t,\mathbf{x})\right) \approx \mathbf{u}^*(t,\mathbf{x}). \tag{2.2}$$

The accuracy of (2.2) depends on the approximation accuracy of the value gradient, $\widehat{V}_{\mathbf{x}}(t,\mathbf{x}) \approx V_{\mathbf{x}}(t,\mathbf{x})$. Here it should be emphasized that because we model the value function with an NN, the gradients are computed using automatic differentiation and are therefore *exact*.

To illustrate the core algorithm, the method is applied to design an attitude controller of a rigid body satellite equipped with momentum wheels. This is a highly nonlinear problem with $n = 6$ spatial dimensions and $m = 3$ control inputs. With the proposed method, we obtain a model of the value function with accuracy comparable to that obtained by sparse grid characteristics [67], but require far fewer sample trajectories to do so. This first example highlights several advantages of the proposed framework, notably the ability to solve high-dimensional HJB equations over semi-global domains, efficient use of data, the ability to empirically validate model accuracy, and computationally efficient nonlinear feedback control for real-time applications.

2.1.1 Value gradient and optimal control models

Note that besides value function models, we can alternatively train an NN to approximate the value gradient,

$$\widehat{\boldsymbol{\lambda}}(t,\mathbf{x}) = \boldsymbol{\mathcal{N}}(t,\mathbf{x};\boldsymbol{\theta}) \approx V_{\mathbf{x}}(t,\mathbf{x}) \tag{2.3}$$

where $\boldsymbol{\mathcal{N}} : [0, t_f] \times \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^n$ is an NN; or even directly learn the optimal control,

$$\widehat{\mathbf{u}}(t, \mathbf{x}) = \boldsymbol{\mathcal{N}}(t, \mathbf{x}; \boldsymbol{\theta}) \approx \mathbf{u}^*(t, \mathbf{x}), \tag{2.4}$$

for where $\boldsymbol{\mathcal{N}} : [0, t_f] \times \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^m$. Section 2.3.2 will detail how we modify (2.4) for control-constrained problems. In the case of the value gradient model (2.3) the feedback control is computed as

$$\widehat{\mathbf{u}}(t, \mathbf{x}) = \mathbf{u}^* \left(t, \mathbf{x}; \widehat{\boldsymbol{\lambda}}(t, \mathbf{x}) \right), \tag{2.5}$$

analogously to (2.2). Optimal control models are used directly for control without solving (1.18).

Throughout the text we refer to NN value function models as V-NN, value gradient models as λ-NN, and optimal control models as u-NN. Numerical example in Chapters 2 and 3 are carried out with V-NN models. Other model types are explored in more detail in Chapters 4 and 5.

2.2 Causality-free data generation

Before we can train any models we need to construct a data set. In this section we give a very brief overview of the data generation procedure. Details, practical considerations, and extensions are discussed later in Chapter 3. Here we focus on finite horizon OCPs (1.2); we discuss the infinite horizon case in Section 2.2.1.

Suppose that we want to model the value function, its gradient, or the optimal control policy over some compact domain $\mathbb{X} \subset \mathbb{R}^n$. We then (uniformly) randomly

sample a set of N_{OCP} initial conditions

$$\left\{\mathbf{x}_0^{(j)}\right\}_{j=1}^{N_{\text{OCP}}} \subset \mathbb{X}, \tag{2.6}$$

where the superscript (j) denotes the sample index. For each initial condition $\mathbf{x}_0^{(j)}$ we solve the corresponding OCP (1.2) to obtain an optimal trajectory, $\mathbf{x}^*\left(t; \mathbf{x}_0^{(j)}\right)$, where t ranges from $t = 0$ to $t = t_f$. It is worth emphasizing that the open loop OCPs for each initial condition can be solved independently, and thus we are free to sample them as desired without the need for a grid. This also makes data generation embarassingly parallelizable, so we can leverage distributed computing resources to greatly increase the efficiency of this step [67].

Now we evaluate each trajectory at $N_t^{(j)}$ sample time instances

$$t_1^{(j)}, \ldots, t_{N_t^{(j)}}^{(j)} \in [0, t_f], \qquad t_1^{(j)} = 0, \quad t_{N_t^{(j)}}^{(j)} = t_f, \tag{2.7}$$

which can be chosen by the numerical OCP solver or according to some other criteria, see e.g. [55, 21]. By (1.22), each sample time $t^{(j,k)} := t_k^{(j)}$ is associated with a sampled state $\mathbf{x}^{(j,k)} := \mathbf{x}^*\left(t_k^{(j)}; \mathbf{x}_0^{(j)}\right)$ and corresponding optimal cost $V^{(j,k)} := V\left(t_k^{(j)}, \mathbf{x}^*\left(t_k^{(j)}; \mathbf{x}_0^{(j)}\right)\right)$, costate $\boldsymbol{\lambda}^{(j,k)} := \boldsymbol{\lambda}\left(t_k^{(j)}; \mathbf{x}_0^{(j)}\right)$, and optimal control $\mathbf{u}^{(j,k)} := \mathbf{u}^*\left(t_k^{(j)}; \mathbf{x}_0^{(j)}\right)$. This yields a data set

$$\mathcal{D} = \left\{ \left\{ t^{(j,k)}, \mathbf{x}^{(j,k)}, V^{(j,k)}, \boldsymbol{\lambda}^{(j,k)}, \mathbf{u}^{(j,k)} \right\}_{k=1}^{N_t^{(j)}} \right\}_{j=1}^{N_{\text{OCP}}}. \tag{2.8}$$

For our purposes we will not need to keep track of which points are associated with which trajectory. Hence with some abuse of notation we can reindex the data for all $k = 1, \ldots, N_t^{(j)}, j = 1, \ldots, N_{\text{OCP}}$ into a common index $i = 1, \ldots, N_{\text{data}}$,

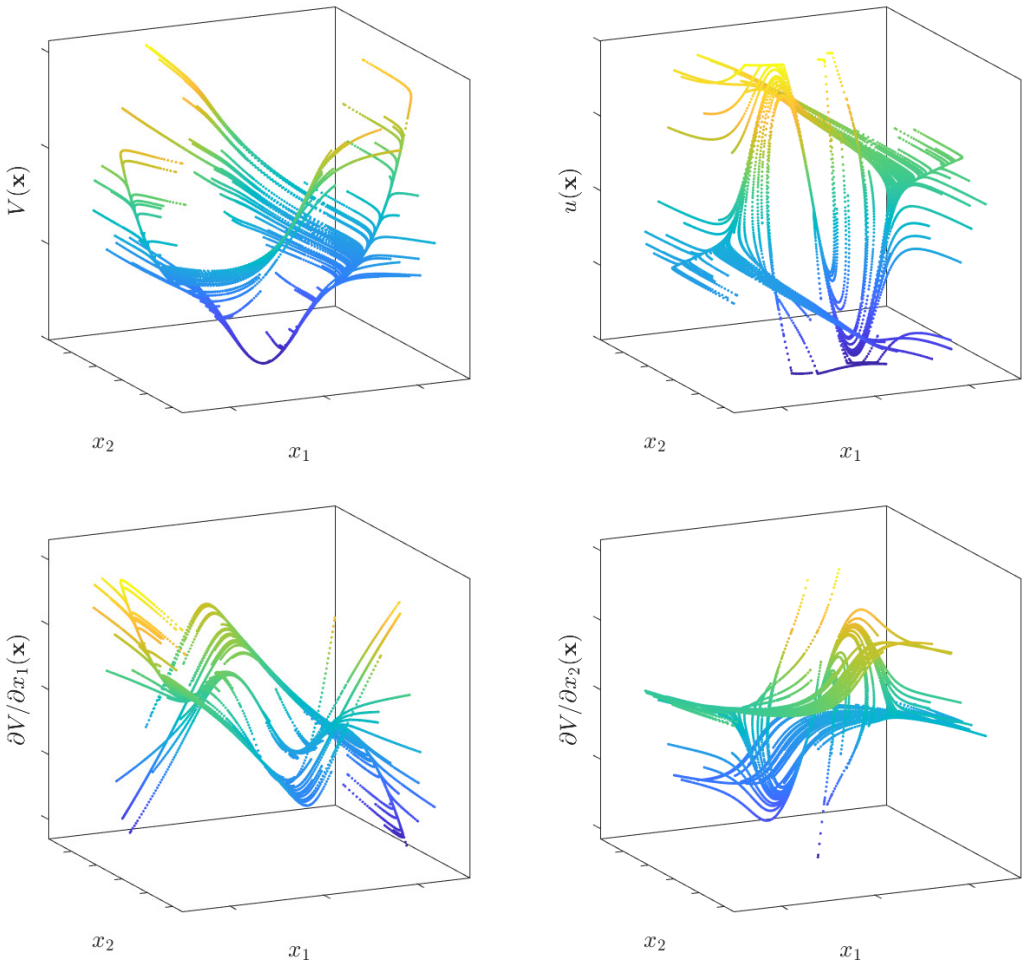


Figure 2.2: Example of a data set for a prototypical (infinite horizon) OCP. Top left: value function. Top right: optimal control. Bottom: value gradients or costates.

where $N_{\text{data}} = \sum_{j=1}^{N_{\text{OCP}}} N_t^{(j)}$ is the total number of data. Hence we rewrite (2.8) as

$$\mathcal{D} = \left\{ t^{(i)}, \mathbf{x}^{(i)}, V^{(i)}, \boldsymbol{\lambda}^{(i)}, \mathbf{u}^{(i)} \right\}_{i=1}^{N_{\text{data}}}. \tag{2.9}$$

We visualize such a combined data set in Figure 2.2. Notice that $\mathcal{D}$ consists of associated input-output pairs, namely inputs $\left(t^{(i)}, \mathbf{x}^{(i)}\right)$ and outputs $\left(V^{(i)}, \boldsymbol{\lambda}^{(i)}, \mathbf{u}^{(i)}\right)$. This gives rise to a regression problem as we discuss in Section 2.3.3.

We use the same data generation process to construct a training data set $\mathcal{D}_{\text{train}}$, as well as a test data set $\mathcal{D}_{\text{test}}$. The training data set is, of course, used for model training, while the test data set is constructed from *independently sampled* initial conditions and thus provides a means for an unbiased evaluation of the NN's approximation accuracy after training.

2.2.1 Data for infinite horizon problems

In general we approximate infinite horizon OCPs by finite horizon OCPs with a very large final time t_f, but we assume that the difference is negligible for the variables of interest, $\mathbf{x}^{(i)}, V^{(i)}, \boldsymbol{\lambda}^{(i)}$, and $\mathbf{u}^{(i)}$.

Recall that for infinite horizon OCPs (1.4) the value function, value gradient, and optimal control are time-invariant. This means that the open loop data we obtain by solving infinite horizon OCPs must also be time-invariant, and hence for such problems we can ignore the $t^{(i)}$ data. In contrast to learning the initial time value function of a finite horizon problem as in Section 2.4, we can still use the entirety of all open loop optimal trajectories.

While generating data in this way is efficient because we extract a lot of data from each successful OCP solution, it has the side effect of concentrating a large amount of data near the equilibrium. On the other hand, we are interested in designing controllers which are effective over large regions of the state space and consequently we need data sets which support learning far from the equilibrium.

Thus in order to not bias the data set too severely we typically do not include the whole trajectory, taking only points corresponding to times $t_{(j,k)} \leq T^{(j)}$ for some $T^{(j)} < t^{(j,N_t)}$. Depending on the problem we might choose the maximum time to simply be $T^{(j)} = Mt_{(j,N_t)}$ for some constant $M \in (0,1)$, or we might take

$T^{(j)}$ to be the time when the system reaches equilibrium. In the latter case this means that we set a tolerance $\varepsilon > 0$ and pick

$$T^{(j)} = \min_k t^{(j,k)}, \qquad \text{subject to } \left\| \mathbf{f}\left(\mathbf{x}^{(j,k)}, \mathbf{u}^{(j,k)}\right) \right\| < \varepsilon. \tag{2.10}$$

2.3 Neural network modeling

NNs have become a popular tool for modeling high-dimensional functions, with recent applications in numerous scientific domains. In theory they can approximate any continuous nonlinear function without discretizing the input space. Recent advances in computing hardware, in particular graphical processing units (GPUs), along with the development of robust deep learning software packages like *TensorFlow* [1] and *PyTorch* [111], have contributed to making NNs a viable modeling approach for many challenging problems.

The crux of the proposed control design framework depends on modeling the value function, its gradient, or the optimal control policy. This section is devoted to explaining this process. In Section 2.3.1 we review the basic structure of feedforward NNs and the NN universal approximation theorem. In Section 2.3.2 we propose a method to smoothly incorporate control saturation constraints into an NN controller. Then in Section 2.3.3 we discuss the mechanics of supervised learning. Here we also propose a simple way to incorporate information about the known solution structure into training value function and value gradient models. Section 2.3.4 lists the accuracy metrics we use for model evaluation. Finally in Section 2.3.5 we demonstrate how to use the trained NN for feedback control.

2.3.1 Artificial neural networks

In this dissertation we use fully-connected multilayer feedforward NNs. More sophisticated architectures have been developed for other applications, but we find this basic architecture to be mostly than adequate for our purposes[1]. NNs are well-known for their ability to approximate arbitrary continuous functions [54] and have been observed to work for extremely high-dimensional problems [81]. This motivates us to try to use them to approximate high-dimensional value functions and optimal controls.

Suppose we want to approximate a k times continuously differentiable ($\mathcal{C}^k$) function $\mathbf{f} : \mathbb{X} \to \mathbb{R}^d$ over the compact subset $\mathbb{X} \subset \mathbb{R}^n$. The input can of course be augmented with time t or any other variables of interest. Feedforward NNs approximate such complicated nonlinear functions by a composition of simpler functions, namely

$$\mathbf{f}(\mathbf{x}) \approx \mathcal{N}(\mathbf{x}) = \mathbf{g}_L \circ \mathbf{g}_{L-1} \circ \cdots \circ \mathbf{g}_\ell \circ \cdots \circ \mathbf{g}_1(\mathbf{x}), \tag{2.11}$$

where each layer is defined as

$$\mathbf{g}_\ell(\mathbf{y}) = \mathbf{g}_\ell\left(\mathbf{y}; \mathbf{W}_\ell, \mathbf{b}_\ell\right) = \sigma_\ell\left(\mathbf{W}_\ell \mathbf{y} + \mathbf{b}_\ell\right), \qquad \ell = 1, \ldots, L. \tag{2.12}$$

Here $\mathbf{W}_\ell$ and $\mathbf{b}_\ell$ are called *weight matrices* and *bias vectors*, respectively. $\sigma_\ell : \mathbb{R} \to \mathbb{R}$ is a nonlinear *activation function* applied component-wise to its argument; popular choices include $\mathrm{ReLU}(\cdot)$, $\tanh(\cdot)$, and other similar functions. The final layer, $\mathbf{g}_L(\cdot)$, is typically linear, so $\sigma_L(\cdot)$ is the identity function. In this dissertation

[1] In Chapter 5 we introduce modified architectures to guarantee LAS. These still incorporate standard feedforward NNs as their central function approximation component.

we always use activation functions which are at least $\mathcal{C}^1$, such as $\tanh(\cdot)$.

The weights $\mathbf{W}_\ell$ and biases $\mathbf{b}_\ell$ can be any size as long the matrix dimensions are compatible. The network depth[2] L, layer width w (corresponding to the number of rows in the weight matrices), and activation functions $\sigma_\ell(\cdot)$ are hyperparameters which need to be tuned to achieve good approximation performance.

2.3.1.1 Classical neural network universal approximation theorem

It has been widely observed that NNs are able to learn complicated, nonlinear, high-dimensional functions in practice [81]. Although the reasons for this are not well understood, a commonly cited theoretical foundation for this property is the *universal approximation theorem*. Many variations of this exist but we present a seminal result from [54]. This theorem says that for any $\mathcal{C}^k$ function $f(\cdot)$ on a compact domain, there exists an NN with $\mathcal{C}^k$ activation functions that approximates $f(\cdot)$ and its derivatives up to arbitrary tolerance.

In the statement of Theorem 1 let $\mathcal{C}^k(\mathbb{X})$ be the vector space of k times continuously differentiable functions from $\mathbb{X}$ to $\mathbb{R}$. Let $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ be a multi-index of integers $\alpha_i \leq k$ and denote $|\boldsymbol{\alpha}| \coloneqq \sum_{i=1}^n \alpha_i$. We write partial derivatives with this multi-index as follows:

$$D^{\boldsymbol{\alpha}} f(\mathbf{x}) \coloneqq \frac{\partial^{|\boldsymbol{\alpha}|} f}{\partial x_1^{\alpha_1} \cdots \partial x_n^{\alpha_n}}(\mathbf{x}). \tag{2.13}$$

Theorem 1 (Universal approximation theorem [54])**.** *Suppose $\sigma \in \mathcal{C}^k(\mathbb{R})$ is non-constant with bounded derivatives up to order k. Then for all functions $f \in C^k(\mathbb{X})$*

[2] An NN with many layers, i.e. $L \gg 1$, is called a "deep NN", hence the term "deep learning."

and all $\epsilon > 0$*, there exists a function* $\mathcal{N} \in \mathcal{C}^k(\mathbb{X})$ *of the form*

$$\mathcal{N}(\mathbf{x}) = \mathbf{W}_2 \left[\sigma\left(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1\right)\right] + \mathbf{b}_2, \tag{2.14}$$

for some $w \in \mathbb{N}$*,* $\mathbf{W}_1 \in \mathbb{R}^{w \times n}$*,* $\mathbf{b}_1 \in \mathbb{R}^w$*,* $\mathbf{W}_2 \in \mathbb{R}^{d \times w}$*, and* $\mathbf{b}_2 \in \mathbb{R}^d$*, such that*

$$\max_{|\boldsymbol{\alpha}|} \sup_{\mathbf{x} \in \mathbb{X}} |D^{\boldsymbol{\alpha}} f(\mathbf{x}) - D^{\boldsymbol{\alpha}} \mathcal{N}(\mathbf{x})| < \epsilon. \tag{2.15}$$

For clarity Theorem 1 is stated for real-valued functions, single layer NNs, and the L^2 function norm. However, it extends straightforwardly to functions taking values in $\mathbb{R}^d$, multilayer NNs, and L^p spaces equipped with finite measures [54]. Note that this theorem and others like it tell us nothing about how to *find* such an NN in practice – only that one exists. Nevertheless, such universal approximation theorems are needed as basic theoretical justification for using NNs for regression tasks. We will use Theorem 1 in Chapter 5 to prove analogous approximation theorems for the NN architectures developed in this dissertation.

2.3.2 Smooth constraints for optimal control models

If a problem has control saturation constraints – which essentially all real world problems do – then our NN controller must obviously not exceed these constraints. If the NN in question is a V-NN or λ-NN then constraints are accounted for when solving (1.18) to get the NN feedback (2.2) or (2.5).

For u-NNs which directly approximate the optimal control, we have two options. The first option is to train the model while ignoring the constraints, and

then at runtime apply the saturation function, $\text{sat} : \mathbb{R}^m \to \mathbb{U}$:

$$\text{sat}\left(\mathbf{u}\right) = \begin{pmatrix} \text{sat}\left(u_1\right) \\ \vdots \\ \text{sat}\left(u_m\right) \end{pmatrix}, \qquad \text{sat}\left(u_i\right) \coloneqq \begin{cases} u_{\min,i}, & u_i < u_{\min,i}, \\ u_i, & u_{\min,i} \leq u_i \leq u_{\max,i}, \\ u_{\max,i}, & u_{\max,i} < u_i \ . \end{cases} \tag{2.16}$$

Alternatively, we can incorporate constraints directly into the model by modifying the final layer of the NN. We prefer the latter option, with the intuition that this approach lets the NN learn to adjust its predictions to account for saturation.

For this purpose we want a *smooth* saturation function since this makes learning easier by preventing vanishing gradients when the control becomes saturated during training. To this end assume that $(\mathbf{x}_f, \mathbf{u}_f)$ is an equilibrium of the dynamics. Now let $\boldsymbol{\mathcal{N}} : [0, t_f] \times \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^m$ be the NN component and construct

$$\widehat{\mathbf{u}}\left(t, \mathbf{x}\right) = \sigma\left[\boldsymbol{\mathcal{N}}\left(t, \mathbf{x}; \boldsymbol{\theta}\right)\right], \tag{2.17}$$

where we choose $\sigma : \mathbb{R}^m \to \mathbb{U}$ to be a generalized logistic function:

$$\sigma\left(\mathbf{u}\right) \coloneqq \mathbf{u}_{\min} + \frac{\mathbf{u}_{\max} - \mathbf{u}_{\min}}{\mathbf{1} + \mathbf{c}_1 \exp\left[-\mathbf{c}_2\left(\mathbf{u} - \mathbf{u}_f\right)\right]}. \tag{2.18}$$

Here $\mathbf{c}_1, \mathbf{c}_2 \in \mathbb{R}^m$ are constants, multiplication and division are performed elementwise, and we must also clip the value of the exponent $-\mathbf{c}_2\left(\mathbf{u} - \mathbf{u}_f\right)$ to prevent numerical overflow when evaluating the gradient during training.

We set the values of the constants as

$$\mathbf{c}_1 = \frac{\mathbf{u}_{\max} - \mathbf{u}_f}{\mathbf{u}_f - \mathbf{u}_{\min}}, \quad \mathbf{c}_2 = \frac{\mathbf{u}_{\max} - \mathbf{u}_{\min}}{\left(\mathbf{u}_{\max} - \mathbf{u}_f\right)\left(\mathbf{u}_f - \mathbf{u}_{\min}\right)}. \tag{2.19}$$

It is straightforward to verify that these choices of $\mathbf{c}_1, \mathbf{c}_2$ satisfy $\sigma(\mathbf{u}_f) = \mathbf{u}_f$ and $\frac{\partial \sigma}{\partial \mathbf{u}}(\mathbf{u}_f) = 1$. Consequently, $\sigma(\cdot)$ smoothly imposes saturation constraints while preserving the unsaturated control behavior near $\mathbf{u}_f$.

2.3.3 Model training

Suppose that we want to learn a model of the value function, $\widehat{V}(t, \mathbf{x}) \approx V(t, \mathbf{x})$. Let $\boldsymbol{\theta} \in \mathbb{R}^p$ denote the collection of the parameters of the NN model,

$$\boldsymbol{\theta} \coloneqq \{\mathbf{W}_\ell, \mathbf{b}_\ell\}_{\ell=1}^{L}, \tag{2.20}$$

concatenated in a single p-dimensional vector. We often write $\widehat{V}(t, \mathbf{x}) = \widehat{V}(t, \mathbf{x}; \boldsymbol{\theta})$ to make the dependence on parameters explicit. The NN is trained by optimizing over the parameters $\boldsymbol{\theta}$ to best approximate $V(t, \mathbf{x})$. Specifically, given a training data set $\mathcal{D}_{\text{train}}$, we can train an NN value function model by solving a nonlinear least squares regression problem,

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \left\{ \underset{V}{\text{loss}}\left(\boldsymbol{\theta}; \mathcal{D}_{\text{train}}\right) \right\}, \tag{2.21}$$

where

$$\underset{V}{\text{loss}}\left(\boldsymbol{\theta}; \mathcal{D}_{\text{train}}\right) \coloneqq \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \left[V^{(i)} - \widehat{V}\left(t^{(i)}, \mathbf{x}^{(i)}; \boldsymbol{\theta}\right) \right]^2. \tag{2.22}$$

The function we seek to minimize is called the *loss function* or *objective function*. In this naïve setting it is a mean square error (MSE) loss, which is commonly used in regression problems.

2.3.3.1 Physics-informed learning

Motivated by the development of PINNs [116] we expect that we can improve on the rudimentary loss function (2.22) by incorporating information about the underlying problem structure. In [116], and in particular in the context of HJB equations in [134, 129, 90], the known underlying PDE and boundary conditions are imposed by minimizing a residual loss over spatio-temporal collocation points. In these approaches, no data is gathered: the PDE is solved directly in the least-squares sense. But this residual must be evaluated over a large number of collocation points and can be rather expensive to compute. Thus we propose a simpler approach of modeling the costate $\boldsymbol{\lambda}(\cdot)$ along with the value function itself, taking full advantage of the ability to gather data along the characteristics of the value function.

Specifically, we know that when the two-point BVP solutions are optimal that value gradient must be equal to the costate. Thus we train the NN to simultaneously minimize the value loss (2.22) and

$$\underset{\boldsymbol{\lambda}}{\text{loss}}\left(\boldsymbol{\theta}; \mathcal{D}_{\text{train}}\right) \coloneqq \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \left\| \boldsymbol{\lambda}^{(i)} - \widehat{V}_{\mathbf{x}}\left(t^{(i)}, \mathbf{x}^{(i)}; \boldsymbol{\theta}\right) \right\|^2. \tag{2.23}$$

The gradient of the NN value function model, $\widehat{V}_{\mathbf{x}}(\cdot)$, is calculated using automatic differentiation. In machine learning, automatic differentiation is usually used to compute gradients with respect to the model parameters, but is just as easy to apply to computing gradients with respect to inputs. This gradient is *exact* so no finite difference approximations are needed, and the computational graph is pre-compiled so these computations incur relatively little additional cost.

We can also add a third MSE loss term to directly penalize deviating from the

optimal control data:

$$\underset{\mathbf{u}}{\text{loss}}\left(\boldsymbol{\theta}; \mathcal{D}_{\text{train}}\right) \coloneqq \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \left\| \mathbf{u}^{(i)} - \widehat{\mathbf{u}}\left(t^{(i)}, \mathbf{x}^{(i)}; \boldsymbol{\theta}\right) \right\|^2. \tag{2.24}$$

It goes without saying that this loss term, as well as (2.23), contribute directly to improving closed loop control performance. Recall for V-NN and λ-NN that $\widehat{\mathbf{u}}(t, \mathbf{x}) = \mathbf{u}^*\left(t, \mathbf{x}; \widehat{V}_{\mathbf{x}}(t, \mathbf{x})\right)$. For these models learning from (2.24) requires that we are able to solve (1.18) analytically to get $\mathbf{u}^*(\cdot)$ in terms of t, $\mathbf{x}$, and $V_{\mathbf{x}}(\cdot)$.

We now arrive at the following *physics-informed* learning problem,

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \left\{ \mu_V \underset{V}{\text{loss}}\left(\boldsymbol{\theta}; \mathcal{D}_{\text{train}}\right) + \mu_{\boldsymbol{\lambda}} \underset{\boldsymbol{\lambda}}{\text{loss}}\left(\boldsymbol{\theta}; \mathcal{D}_{\text{train}}\right) + \mu_{\mathbf{u}} \underset{\mathbf{u}}{\text{loss}}\left(\boldsymbol{\theta}; \mathcal{D}_{\text{train}}\right) \right\}, \tag{2.25}$$

where $\mu_V, \mu_{\boldsymbol{\lambda}}, \mu_{\mathbf{u}} \geq 0$ are scalar weights. An NN trained to minimize (2.25) learns not just to fit the value function data, but it is rewarded for doing so in a way which respects the underlying problem structure. Gradient and control regularization take this known structure into account; and are therefore preferable to the usual ℓ^1 or ℓ^2 regularization, which are based on the (heuristic) principle that simpler models are likely to generalize better. As noted above, these regularization terms also contribute directly to the ultimate goal of achieving learning an optimal control policy. We demonstrate in Section 2.4 that the physics-informed learning problem (2.25) produces more accurate results with less data then models trained on value function data only.

2.3.3.2 Training value gradient and optimal control models

The procedure for training a model of the value gradient, $\widehat{V}_{\mathbf{x}}(t, \mathbf{x}; \boldsymbol{\theta}) \approx V_{\mathbf{x}}(t, \mathbf{x})$, or optimal control, $\widehat{\mathbf{u}}(t, \mathbf{x}; \boldsymbol{\theta}) \approx \mathbf{u}^*(t, \mathbf{x})$, is much the same as for a value function

model V-NN. All that is different in these cases is which loss terms can be included. In particular, λ-NN does not predict the value function directly[3], so we cannot use (2.22). Hence the optimization problem we solve to train a λ-NN is

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \left\{ \mu_{\boldsymbol{\lambda}} \underset{\boldsymbol{\lambda}}{\text{loss}} \left(\boldsymbol{\theta}; \mathcal{D}_{\text{train}} \right) + \mu_{\mathbf{u}} \underset{\mathbf{u}}{\text{loss}} \left(\boldsymbol{\theta}; \mathcal{D}_{\text{train}} \right) \right\}, \tag{2.26}$$

where we replace $\widehat{V}_{\mathbf{x}}(\cdot)$ by $\widehat{\boldsymbol{\lambda}}(\cdot)$. Similarly, u-NN directly approximates the optimal control and so is not able to use the value loss (2.22) or the gradient loss (2.23). Therefore for such models we simply solve

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \left\{ \underset{\mathbf{u}}{\text{loss}} \left(\boldsymbol{\theta}; \mathcal{D}_{\text{train}} \right) \right\}. \tag{2.27}$$

Unfortunately (2.27) does not benefit from physics-informed regularization, but we will see in later chapters that u-NN still performs about as well as V-NN and λ-NN.

2.3.3.3 Learning for infinite horizon problems

Adjusting the models and loss functions for infinite horizon problems (1.4) is as simple as removing t from the model inputs, i.e. we learn $\widehat{V}(\mathbf{x}; \boldsymbol{\theta}) \approx V(\mathbf{x})$, $\widehat{V}_{\mathbf{x}}(\mathbf{x}; \boldsymbol{\theta}) \approx V_{\mathbf{x}}(\mathbf{x})$, and $\widehat{\mathbf{u}}(\mathbf{x}; \boldsymbol{\theta}) \approx \mathbf{u}^*(\mathbf{x})$. As we point out in Section 2.2 when dealing with infinite horizon problems the data becomes time invariant, so we combine data from all trajectories and ignore the time data.

[3] If desired for e.g. controller analysis purposes, the cost $\mathcal{J}[\widehat{\mathbf{u}}(\cdot); \mathbf{x}_0]$ of a particular initial condition $\mathbf{x}_0$ can be recovered by closed loop integration.

2.3.3.4 Numerical optimization

Nonlinear regression problems (2.25–2.27) cannot be solved analytically. Thus training an NN requires the use of numerical optimization techniques. Here we give a brief overview of this process, details and further explanation can be found in [104, 17].

Suppose that we have a training data set $\mathcal{D} = \left\{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right\}_{i=1}^{N}$, where $\left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right)$ are samples of an input-output relationship which we want to learn. Consider a typical regression problem of the form

$$\underset{\boldsymbol{\theta}}{\text{minimize}}\ \ell\left(\boldsymbol{\theta}; \mathcal{D}\right), \qquad \ell\left(\boldsymbol{\theta}; \mathcal{D}\right) := \frac{1}{N}\sum_{i=1}^{N} \ell\left(\boldsymbol{\theta}; \mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right). \tag{2.28}$$

Here $\ell\left(\cdot\right)$ is a sufficiently smooth loss function as in (2.25–2.27) and $\boldsymbol{\theta}$ denote the model parameters. In optimization terminology these parameters are referred to as *decision variables*.

The most fundamental algorithm for solving (2.28) is *gradient descent*. Given an initial estimate for $\boldsymbol{\theta} = \boldsymbol{\theta}_1$, we update the parameters in an iterative process by moving in the direction of the gradient of the loss function. That is, at each optimization iteration k we set

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha_k \nabla_{\boldsymbol{\theta}} \ell\left(\boldsymbol{\theta}_k; \mathcal{D}\right) = \boldsymbol{\theta}_k - \frac{\alpha_k}{N}\sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} \ell\left(\boldsymbol{\theta}_k; \mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right), \tag{2.29}$$

where $\alpha_k > 0$ is an appropriate step size chosen by line search [104]. Assuming $\ell\left(\cdot\right)$ is bounded from below (which is typically the case by design), then $\lim_{k\to\infty} \nabla_{\boldsymbol{\theta}} \ell\left(\boldsymbol{\theta}_k; \mathcal{D}\right) = \mathbf{0}$ [104]. That is, the weights $\boldsymbol{\theta}_k$ approach a stationary point $\boldsymbol{\theta}^*$. We generally assume that $\boldsymbol{\theta}^*$ is a local minimizer of (2.28).

The gradient descent algorithm (2.29) is referred to as a full batch method

since it uses the entire data set to compute each parameter update. A popular alternative in deep learning with massive data sets is *stochastic gradient descent* (SGD), computes weight updates using only single samples or small subsets (called mini-batches) of the data. These subsets are drawn *randomly* from $\mathcal{D}$, and the step size α_k is usually specified *a priori*. If we let $\mathcal{S}_k \subset \mathcal{D}$ be the mini-batch at the kth step then the gradient descent update becomes

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha_k \nabla_{\boldsymbol{\theta}} \ell\left(\boldsymbol{\theta}_k; \mathcal{S}_k\right) = \boldsymbol{\theta}_k - \frac{\alpha_k}{|\mathcal{S}_k|} \sum_{\left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right) \in \mathcal{S}_k} \nabla_{\boldsymbol{\theta}} \ell\left(\boldsymbol{\theta}_k; \mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right). \quad (2.30)$$

The stochastic formulation saves significant computational resources compared to full batch methods since only a small subset of the data is used at each step. One can also derive probabilistic convergence results for (2.30) which recover the same convergence rate as the full batch method (2.29). Researchers have developed a number of variations of SGD with various favorable properties, and SGD variants have become the de facto standard for machine learning applications. We refer the reader to [17] for a review.

Gradient descent and SGD are called *first order* methods because they depend only on the gradient of the loss function. On the other hand, in the context of deep learning we are dealing with relatively small NNs and data sets. This opens up the possibility of using *second order* methods which obtain much faster convergence rates by incorporating information about the curvature of the loss function in the form of the (approximate) Hessian. The most popular second order optimizer is perhaps the limited memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS, [85]) algorith. This is a quasi-Newton method which computes update steps of the form

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha_k \mathbf{h}_k, \qquad \mathbf{h}_k \approx \left[\nabla^2_{\boldsymbol{\theta}} \ell\left(\boldsymbol{\theta}_k; \mathcal{D}\right)\right]^{-1} \nabla_{\boldsymbol{\theta}} \ell\left(\boldsymbol{\theta}_k; \mathcal{D}\right). \quad (2.31)$$

Observe that the update vectors $\mathbf{h}_k$ approximate the matrix-vector product of the inverse Hessian and the gradient, thus approximately minimizing a second order Taylor series approximation of the loss at each step. Remarkably, $\mathbf{h}_k$ can be computed without ever forming or inverting the Hessian matrix $\nabla^2_{\boldsymbol{\theta}} \ell\left(\boldsymbol{\theta}_k; \mathcal{D}\right)$, using only information about a few previous parameter updates $\boldsymbol{\theta}_k, \boldsymbol{\theta}_{k-1}, \boldsymbol{\theta}_{k-2}, \ldots$ and gradient vectors $\nabla_{\boldsymbol{\theta}} \ell\left(\boldsymbol{\theta}_k; \mathcal{D}\right), \nabla_{\boldsymbol{\theta}} \ell\left(\boldsymbol{\theta}_{k-1}; \mathcal{D}\right), \nabla_{\boldsymbol{\theta}} \ell\left(\boldsymbol{\theta}_{k-2}; \mathcal{D}\right), \ldots$ In this work we prefer L-BFGS as we find it provides significantly faster convergence without needing to manually tune step sizes α_k. We only ever use Adam [73], a popular SGD variant, when dealing with very large data sets.

It is important to mention that the gradient $\nabla_{\boldsymbol{\theta}} \ell\left(\cdot\right)$ is calculated *exactly* and efficiently by means of *automatic differentiation*. Automatic differentiation keeps track of mathematical operations during each forward pass through the computational graph of $\ell\left(\boldsymbol{\theta}; \mathcal{D}\right)$, and then backpropagates through the graph performing the chain rule automatically. This is a key feature of all modern machine learning software frameworks, and removes the need to manually implement gradient calculations.

2.3.4 Model accuracy evaluation

The last step of model building is to test its approximation accuracy. During training, the loss functions and gradients are calculated with respect to the training data $\mathcal{D}_{\text{train}}$. Afterwards we evaluate the performance of the NN against a separate test data $\mathcal{D}_{\text{test}}$, which it did not observe during training. Good test performance indicates that the NN generalizes well, i.e. it did not overfit the training data. In common machine learning practice, one randomly partitions a given data set into training and test sets $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$. Since we have the freedom to generate data

we make the test more stringent by generating $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ from *independently drawn* initial conditions, so that the two data sets do not share any part of the same trajectories.

We consider the following error metrics for testing. First, the relative mean absolute error (RMAE) of value function prediction, which is defined as

$$\text{RMAE}\left(\boldsymbol{\theta}; \mathcal{D}_{\text{test}}\right) \coloneqq \frac{\sum_{i=1}^{N_{\text{test}}} \left| V^{(i)} - \widehat{V}\left(t^{(i)}, \mathbf{x}^{(i)}; \boldsymbol{\theta}\right)\right|}{\max_{i=1,\dots N_{\text{test}}} \left|V^{(i)}\right|}. \tag{2.32}$$

Here N_{test} denotes the number of data points in $\mathcal{D}_{\text{test}}$. We also measure the relative mean ℓ^2 error (RMℓ^2) of gradient prediction, which is defined as

$$\text{gradient RM}\ell^2\left(\boldsymbol{\theta}; \mathcal{D}_{\text{test}}\right) \coloneqq \frac{\sum_{i=1}^{N_{\text{test}}} \left\| \boldsymbol{\lambda}^{(i)} - \widehat{V}_{\mathbf{x}}\left(t^{(i)}, \mathbf{x}^{(i)}; \boldsymbol{\theta}\right)\right\|_2}{\max_{i=1,\dots N_{\text{test}}} \left\|\boldsymbol{\lambda}^{(i)}\right\|_2}. \tag{2.33}$$

If we are evaluating a value gradient model then of course we replace $\widehat{V}_{\mathbf{x}}(\cdot)$ by $\widehat{\boldsymbol{\lambda}}(\cdot)$. Similarly, define the RMℓ^2 of control prediction:

$$\text{control RM}\ell^2\left(\boldsymbol{\theta}; \mathcal{D}_{\text{test}}\right) \coloneqq \frac{\frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \left\|\mathbf{u}^{(i)} - \widehat{\mathbf{u}}\left(t, \mathbf{x}^{(i)}; \boldsymbol{\theta}\right)\right\|_2}{\max_{i=1,\dots N_{\text{test}}} \left\|\mathbf{u}^{(i)}\right\|_2}, \tag{2.34}$$

We consider these relative error metrics instead of pointwise relative errors in order to emphasize predictive accuracy in regions where a lot of control effort is needed. This is important because we are interested in designing nonlinear controllers which are effective and efficient far away from the equilibrium.

Although less commonly reported in machine learning, we also consider the maximum ℓ^2 control error:

$$\max \ell^2 \coloneqq \max_{i \in \{1,\dots,N_{\text{test}}\}} \left\|\mathbf{u}^{(i)} - \widehat{\mathbf{u}}\left(\mathbf{x}^{(i)}\right)\right\|_2. \tag{2.35}$$

The maximum error can be more relevant and convenient in the context of determining system stability. However, even with low (maximum) test error, there is a chance that the NN could still perform poorly when implemented in the closed loop system. This is demonstrated in Chapter 4. For this reason we believe that test metrics like (2.32–2.35) are insufficient in the context of control design; we should instead focus on rigorous closed loop stability and performance tests such as those presented in Chapter 4.

2.3.5 Neural network in the closed loop system

While data generation and model training require considerable computing resources, once the NN is trained it is very fast to evaluate. For u-NN which directly approximates the optimal control, $\widehat{\mathbf{u}}\left(t, \mathbf{x}; \boldsymbol{\theta}\right) \approx \mathbf{u}^*\left(t, \mathbf{x}\right)$, implementation in the closed loop system is straightforward. Otherwise we need to use (2.2) or (2.5) to compute the approximate optimal control as a function of $\widehat{V}_{\mathbf{x}}\left(t, \mathbf{x}; \boldsymbol{\theta}\right)$ or $\widehat{\boldsymbol{\lambda}}\left(t, \mathbf{x}; \boldsymbol{\theta}\right)$. For the rest of this section we just write $\widehat{V}_{\mathbf{x}}\left(t, \mathbf{x}\right)$ to indicate the approximate value gradient, which can refer to either V-NN or λ-NN.

If we do not directly approximate the control then in general we need to evaluate $\widehat{V}_{\mathbf{x}}\left(t, \mathbf{x}\right)$ and then solve (1.18) to get $\widehat{\mathbf{u}}\left(t, \mathbf{x}\right) = \mathbf{u}^*\left(t, \mathbf{x}; \widehat{V}_{\mathbf{x}}\left(t, \mathbf{x}\right)\right)$. Of course this is not ideal for closed loop implementation because solving this optimization problem might add considerable computational delay. Fortunately, for many problems of interest (1.18) admits an easy or analytic solution in terms of the value gradient. In particular, for the important class of control affine systems with running cost convex in $\mathbf{u}$, we can solve (1.18) analytically.

Suppose that the system dynamics are in the form

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}) + \mathbf{g}(t, \mathbf{x})\mathbf{u}, \tag{2.36}$$

where $\mathbf{f} : [0, t_f] \times \mathbb{R}^n \to \mathbb{R}^n$, $\mathbf{g} : [0, t_f] \times \mathbb{R}^n \to \mathbb{R}^{n \times m}$, and the control is unconstrained. Let $(\mathbf{x}_f, \mathbf{u}_f)$ be an equilibrium point such that $\mathbf{f}(t, \mathbf{x}_f) + \mathbf{g}(t, \mathbf{x}_f)\,\mathbf{u}_f = \mathbf{0}$ for all $t \geq 0$ and suppose that the running cost is of the form

$$\mathcal{L}(t, \mathbf{x}, \mathbf{u}) = q(t, \mathbf{x}) + (\mathbf{u} - \mathbf{u}_f)^T \mathbf{R}\,(\mathbf{u} - \mathbf{u}_f)\,, \tag{2.37}$$

for any $q : [0, t_f] \times \mathbb{R}^n \to \mathbb{R}$ and a positive definite weight matrix $\mathbf{R} \in \mathbb{R}^{m \times m}$. Then the Hamiltonian is

$$\mathcal{H}(t, \mathbf{x}, \boldsymbol{\lambda}, \mathbf{u}) = h(t, \mathbf{x}) + (\mathbf{u} - \mathbf{u}_f)^T \mathbf{R}\,(\mathbf{u} - \mathbf{u}_f) + \boldsymbol{\lambda}^T \mathbf{f}(t, \mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{g}(t, \mathbf{x})\mathbf{u}. \tag{2.38}$$

Now we apply the Hamiltonian minimization condition (1.18), which for unconstrained control requires

$$\mathbf{0} = \frac{\partial \mathcal{H}}{\partial \mathbf{u}}(t, \mathbf{x}, \boldsymbol{\lambda}, \mathbf{u}^*) = 2\,(\mathbf{u}^* - \mathbf{u}_f)^T \mathbf{R} + \boldsymbol{\lambda}^T \mathbf{g}\,(t, \mathbf{x})\,. \tag{2.39}$$

Solving for $\mathbf{u}^*$ yields the optimal feedback control law in explicit form:

$$\mathbf{u}^*\,(t, \mathbf{x}; \boldsymbol{\lambda}) = \mathbf{u}_f - \frac{1}{2}\mathbf{R}^{-1}\mathbf{g}^T(t, \mathbf{x})\boldsymbol{\lambda}. \tag{2.40}$$

Finally we substitute $\boldsymbol{\lambda} = \widehat{V}_{\mathbf{x}}\,(t, \mathbf{x})$ to get the NN feedback law,

$$\widehat{\mathbf{u}}(t, \mathbf{x}) = \mathbf{u}^*\left(t, \mathbf{x}; \widehat{V}_{\mathbf{x}}\,(t, \mathbf{x})\right) = \mathbf{u}_f - \frac{1}{2}\mathbf{R}^{-1}\mathbf{g}^T(t, \mathbf{x})\widehat{V}_{\mathbf{x}}\,(t, \mathbf{x})\,. \tag{2.41}$$

Computing the feedback control based on (2.41) typically adds little additional cost compared to evaluating the NN itself. Hence in cases like this where we can solve (1.18) analytically, all of the NN types can provide real time control. By

optimizing a loss function with gradient and/or control learning terms, (2.23) and (2.24), the NN learns to mimic the optimal control. Thus with sufficient high quality data and the right NN architecture (see Chapter 5) we expect the NN controller to be close to optimal.

2.4 Example: Rigid body attitude control

To illustrate the capabilities of the core algorithm, we consider an $n = 6$ state rigid body model of a satellite with $m = 3$ momentum wheels studied by Kang and Wilcox [66, 67]; see also [27]. With the sparse grid characteristics method, [66, 67] interpolate the value function at initial time, $V(t = 0, \mathbf{x})$, and use this for moving horizon feedback control of the nonlinear system. We use their successful results as a baseline for evaluating our core supervised learning algorithm.

Let $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ be an inertial frame of orthonormal vectors and let $\{\mathbf{e}'_1, \mathbf{e}'_2, \mathbf{e}'_3\}$ be a body frame. The attitude of the satellite, i.e. the rotation from inertial to body frames, can be described by Euler angles

$$\mathbf{v} = \begin{pmatrix} \phi & \theta & \psi \end{pmatrix}^T, \tag{2.42}$$

in which ϕ, θ, and ψ are the angles of rotation around $\mathbf{e}'_1$, $\mathbf{e}'_2$, and $\mathbf{e}'_3$, respectively, in the order $(1, 2, 3)$. These are commonly called roll, pitch, and yaw. The angular rates expressed in the body frame are denoted as

$$\boldsymbol{\omega} = \begin{pmatrix} \omega_1 & \omega_2 & \omega_3 \end{pmatrix}^T. \tag{2.43}$$

For a detailed explanation we refer the reader to [32, 7]. The state dynamics are

given by [27]

$$\begin{pmatrix} \dot{\mathbf{v}} \\ \dot{\boldsymbol{\omega}} \end{pmatrix} = \begin{pmatrix} \mathbf{E}(\mathbf{v})\boldsymbol{\omega} \\ \mathbf{J}^{-1}\left[-\boldsymbol{\omega} \times \mathcal{R}_{\mathbf{v}}(\mathbf{h}) + \mathbf{B}\mathbf{u}\right] \end{pmatrix}. \tag{2.44}$$

Here $\mathbf{E}(\mathbf{v}) : \mathbb{R}^3 \to \mathbb{R}^{3\times 3}$ is defined as

$$\mathbf{E}(\mathbf{v}) \coloneqq \begin{pmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{pmatrix}, \tag{2.45}$$

and $\mathcal{R}_{\mathbf{v}}(\mathbf{h}) : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}^3$ is the rotation from inertial to body frames, parameterized in terms of Euler angles $\mathbf{v}$:

$$\mathcal{R}_{\mathbf{v}}(\mathbf{h}) \coloneqq \begin{pmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \cos\theta\sin\phi \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\theta\cos\phi \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \end{pmatrix}, \tag{2.46}$$

where $\mathbf{h} \in \mathbb{R}^3$ is the total (constant) angular momentum of the system. The total inertia matrix $\mathbf{J} \in \mathbb{R}^{3\times 3}$ is a combination of the inertia matrices of the momentum wheels and the rigid body without wheels, and $\mathbf{B} \in \mathbb{R}^{3\times m}$ is a constant matrix projecting the momentum wheel torques onto the body axis. To control the system, we apply a torque $\mathbf{u} : [0, t_f] \times \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}^m$.

Observe that due to the momentum wheels the dynamics are more complicated than when using gas jet actuators [27]. In particular, in the simpler case of gas jet actuators the term $\mathcal{R}_{\mathbf{v}}(\mathbf{h})$ is replaced by $\mathbf{J}\boldsymbol{\omega}$.

We consider the fully-actuated case where $m = 3$. Let

$$\mathbf{B} = \begin{pmatrix} 1 & 1/20 & 1/10 \\ 1/15 & 1 & 1/10 \\ 1/10 & 1/15 & 1 \end{pmatrix}, \qquad \mathbf{J} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix}, \qquad \mathbf{h} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}. \tag{2.47}$$

We solve the following OCP from [67]:

$$\begin{cases} \underset{\mathbf{u}(\cdot)}{\text{minimize}} & \mathcal{J}\left[\mathbf{u}(\cdot)\right] = F\left(\mathbf{v}\left(t_f\right), \boldsymbol{\omega}\left(t_f\right)\right) + \int_0^{t_f} \mathcal{L}(\mathbf{v}, \boldsymbol{\omega}, \mathbf{u}) dt, \\ \text{subject to} & \dot{\mathbf{v}} = \mathbf{E}(\mathbf{v})\boldsymbol{\omega}, \\ & \dot{\boldsymbol{\omega}} = \mathbf{J}\left[-\boldsymbol{\omega} \times \mathcal{R}_{\mathbf{v}}(\mathbf{h}) + \mathbf{B}\mathbf{u}\right], \end{cases} \tag{2.48}$$

with

$$F(\mathbf{v}, \boldsymbol{\omega}) = \frac{W_1}{2}\|\mathbf{v}\|^2 + \frac{W_2}{2}\|\boldsymbol{\omega}\|^2, \tag{2.49}$$

$$\mathcal{L}(\mathbf{v}, \boldsymbol{\omega}, \mathbf{u}) = \frac{W_3}{2}\|\mathbf{v}\|^2 + \frac{W_4}{2}\|\boldsymbol{\omega}\|^2 + \frac{W_5}{2}\|\mathbf{u}\|^2 \tag{2.50}$$

and

$$W_1 = 1, \quad W_2 = 1, \quad W_3 = 1, \quad W_4 = 10, \quad W_5 = \frac{1}{2}, \quad t_f = 20. \tag{2.51}$$

Finally, we consider initial conditions in the domain

$$\mathbb{X} = \left\{ \mathbf{v}, \boldsymbol{\omega} \in \mathbb{R}^3 \middle| -\frac{\pi}{3} \leq \phi, \theta, \psi \leq \frac{\pi}{3} \text{ and } -\frac{\pi}{4} \leq \omega_1, \omega_2, \omega_3 \leq \frac{\pi}{4} \right\}. \tag{2.52}$$

In [67], to avoid discretizing time the value function is approximated only at initial time $t = 0$. To facilitate a fair comparison we do the same. This means that we model $\widehat{V}(\mathbf{v}, \boldsymbol{\omega}) \approx V(0, \mathbf{v}, \boldsymbol{\omega})$, i.e. the NN does *not* take time as an input variable. Consequently the control is implemented with a time-independent moving horizon: at each time t when we evaluate the control, we assume $t = 0$ and return $\mathbf{u}(t) = \widehat{\mathbf{u}}\left(\mathbf{v}(t), \boldsymbol{\omega}(t)\right)$. Controlling the system using moving horizon feedback is standard practice. It is also reasonable for the present case because the problem dynamics are time-invariant and the time horizon is relatively long.

Because of this we observe near-optimal performance from the moving horizon controller.

2.4.1 Learning the value function

In this section, we present numerical results of our implementation of an NN for approximating the initial-time value function of the rigid body attitude control problem (2.48). To obtain data, we solve the BVP (1.10) for each initial condition $\left(\mathbf{v}_0^{(i)}, \boldsymbol{\omega}_0^{(i)}\right)$ uniformly sampled from the domain $\mathbb{X}$ defined in (2.52). Each integrated trajectory contains around 100 data points on average, but we use only initial time data, $V\left(0, \mathbf{v}_0^{(i)}, \boldsymbol{\omega}_0^{(i)}\right)$. For testing, we generate a data set containing $N_{\text{test}} = 1000$ data points at $t = 0$. As a baseline, the sparse grid characteristics method with $44,698$ grid points achieves a test RMAE of 2.93×10^{-4}.

We implement a standard feedforward NN in TensorFlow 1.11 [1] and train it to approximate $V(0, \mathbf{v}, \boldsymbol{\omega})$. The NN has $L = 3$ hidden layers with $w = 64$ neurons in each, but many alternate configurations of depth and width also work. For optimization, we use the *SciPy* [140] interface for the L-BFGS optimizer [85]. Figure 2.3 displays the results of a series of tests in which we vary the weight $\mu_{\boldsymbol{\lambda}}$ on the value gradient loss term (2.23) and the size of the training data set. No penalty is placed on control accuracy (i.e. $\mu_{\mathbf{u}} = 0$). Results are compared to those obtained in [67].

We highlight that with just $N_{\text{train}} = 512$, we can train NNs with better accuracy than the sparse grid characteristics method with $44,698$ points. Thus for this problem, the proposed method can be at least 90 times as data-efficient. With $N_{\text{train}} = 8192$ data points, the NN can be almost four times as accurate as the sparse grid characteristics method.

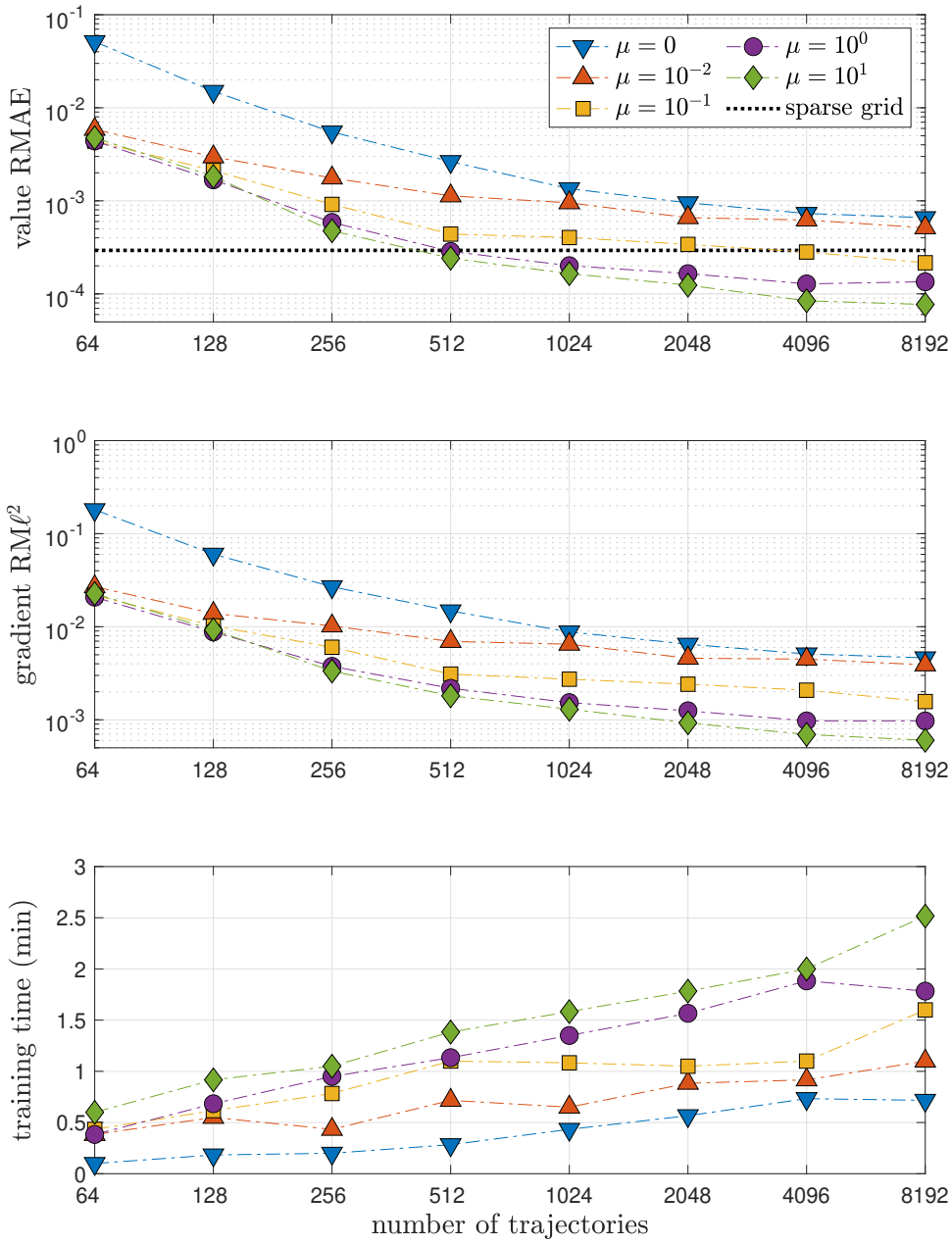


Figure 2.3: Test accuracy and training time of NNs for modeling the initial time value function $V(0, \mathbf{v}, \boldsymbol{\omega})$ of the optimal attitude control problem (2.48).

This level of accuracy with small data sets is obtained only with physics-informed learning. In particular, NNs trained by pure regression (2.25) cannot match the accuracy of the sparse grid characteristics method. This is observed in Figure 2.3 for the case with $\mu_{\boldsymbol{\lambda}} = 0$. Accuracy improves as we increase μ but with diminishing returns for $\mu_{\boldsymbol{\lambda}} \geq 10$. While physics-informed learning is more costly, it facilitates the use of much smaller data sets, and the increased training time is still quite short. Both data efficiency and short training time are important in an iterative control design process where one might have to generate many data sets and train many NNs to obtain a satisfactory controller.

2.4.2 Closed-loop simulation

In this section we demonstrate the use of a trained NN feedback controller for closed loop control. Using (2.41) we calculate the optimal feedback control law

$$\widehat{\mathbf{u}}\left(\mathbf{v}, \boldsymbol{\omega}; \boldsymbol{\theta}\right) = -\frac{1}{W_5}\left[\mathbf{J}^{-1}\mathbf{B}\right]^T \widehat{V}_{\boldsymbol{\omega}}(\mathbf{v}, \boldsymbol{\omega}; \boldsymbol{\theta}). \tag{2.53}$$

Recall that because we are using a time-independent value function model, the control is implemented as time-independent moving horizon feedback. Since $\mathbf{J}$ and $\mathbf{B}$ are constant matrices, we pre-compute the product $-\left[\mathbf{J}^{-1}\mathbf{B}\right]^T/W_5$. Hence evaluation of the control requires only a forward pass through the computational graph of $\widehat{V}_{\boldsymbol{\omega}}(\cdot)$ and a matrix multiplication. As a result each evaluation takes only a couple milliseconds on both an NVIDIA RTX 2080Ti GPU and a 2019 MacBook Air (see Table 5.1). This short computation is critical for feedback implementation in real systems.

In Figure 2.4, we plot a typical closed-loop trajectory starting from a randomly sampled initial condition. To make the simulation more realistic, we implement

the controller using a zero-order hold [44] with a sample rate of 10 [Hz]. In addition, we corrupt inputs to the controller with Gaussian white noise with standard deviation $\sigma = 0.01\pi$. That is, for all $t \in [t_k, t_k + 0.1)$, we apply the control

$$\mathbf{u}(t) = \widehat{\mathbf{u}}\left(\widetilde{\mathbf{v}}\left(t_k\right), \widetilde{\boldsymbol{\omega}}\left(t_k\right)\right), \qquad t_k \leq t_k \leq t + 0.1 \tag{2.54}$$

where

$$\begin{pmatrix} \widetilde{\mathbf{v}}\left(t_k\right) \\ \widetilde{\boldsymbol{\omega}}\left(t_k\right) \end{pmatrix} \coloneqq \begin{pmatrix} \mathbf{v}(t_k) \\ \boldsymbol{\omega}(t_k) \end{pmatrix} + \mathbf{n}(t_k), \qquad \mathbf{n}(t_k) \sim \mathcal{N}\left(\mathbf{0}, \sigma^2 \mathbf{I}\right). \tag{2.55}$$

In spite of these additional challenges, the NN controller successfully stabilizes the system. The total cost of this closed loop trajectory is $\mathcal{J}\left[\widehat{\mathbf{u}}(\cdot); \mathbf{v}_0, \boldsymbol{\omega}_0\right] = 12.67$, about 1% more than the optimal cost $V\left(0, \mathbf{v}_0, \boldsymbol{\omega}_0\right) = \mathcal{J}\left[\mathbf{u}^*(\cdot); \mathbf{v}_0, \boldsymbol{\omega}_0\right] = 12.52$. For comparison, an LQR for (2.48) accumulates a total cost of $\mathcal{J}\left[\mathbf{u}^{\mathrm{LQR}}(\cdot); \mathbf{v}_0, \boldsymbol{\omega}_0\right] = 15.95$, which is 27% more than the optimal cost.

This result is chosen to be representative, but simulating a single trajectory is of course not a rigorous test of the NN's closed loop performance. We provide more thorough stability and performance tests in Chapters 4 and 5.

2.5 Summary

In this chapter we have presented the core supervised learning methodology for our feedback control design framework. For clarity we have focused the presentation on finite horizon OCPs and value function models, but infinite horizon OCPs and other model types are treated in an almost identical fashion. To focus on the core methodology we have kept the discussion on data generation brief, leaving details for Chapter 3.

We have demonstrated the possibility for use of the core algorithm in a mod-

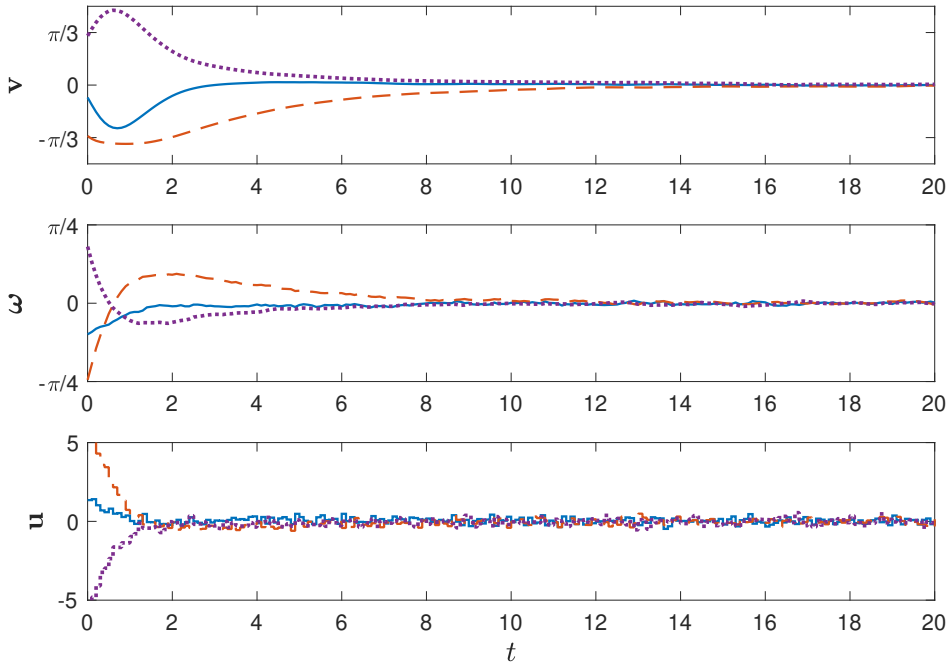


Figure 2.4: Sample closed loop trajectory of the rigid body system (2.48) with NN feedback control implemented with a zero-order hold and subject to measurement noise. Solid: ϕ, ω_1, and u_1. Dashed: θ, ω_2, and u_2. Dotted: ψ, ω_3, and u_3.

erately high-dimensional, practical setting by synthesizing optimal feedback controllers for an six-dimensional nonlinear rigid body. This example is chosen to facilitate comparison with the sparse grid characteristics method. We find that our approach significantly improves on the sparse grid approach in terms of accuracy and data efficiency. In Chapters 3 and 5 we will demonstrate scalability of the method on examples with up to $n = 64$ dimensions, far surpassing what can be achieved with any method based on (sparse) grids.

Unlike many other state of the art techniques, our method does not require finite difference approximations of the gradient, strict restrictions on the structure of the dynamics, or *a priori* knowledge of a stabilizing feedback law. Application to high-dimensional problems is enabled by NN approximation capacity, causality-

free data generation, and data-efficiency by means of physics-informed learning.

As noted in Section 1.2, the approach proposed by [56, 55] is similar to the core algorithm presented in this chapter. The primary difference is in data generation, which we discuss in Chapter 3. Other developments in the proposed framework, most notably the stability-enhancing *QRnet* architectures introduced in Chapter 5, set our work apart from [56, 55].

Chapter 3

Data Generation Strategies for Supervised Learning

To do any supervised learning we need data to learn from. This requires solving a large set of open loop OCPs. While solving open loop OCPs is easier than solving the full HJB equation, for any industry strength problem this can still be quite challenging and time consuming. Even though the computations are done offline and are parallelizable, we still need to be able to reliably generate sufficiently large data sets in a reasonable time frame. In Section 2.2 we gave an overview of the causality-free data generation process and in this chapter we discuss some practical details on *how* to solve the necessary open loop OCPs.

Figure 3.1 positions Chapter 3 within the greater computational framework. This chapter is organized as follows. First in Section 3.1 we describe the two kinds of computational methods for solving open loop OCPs and their associated advantages and disadvantages. These computational methods all require an initial

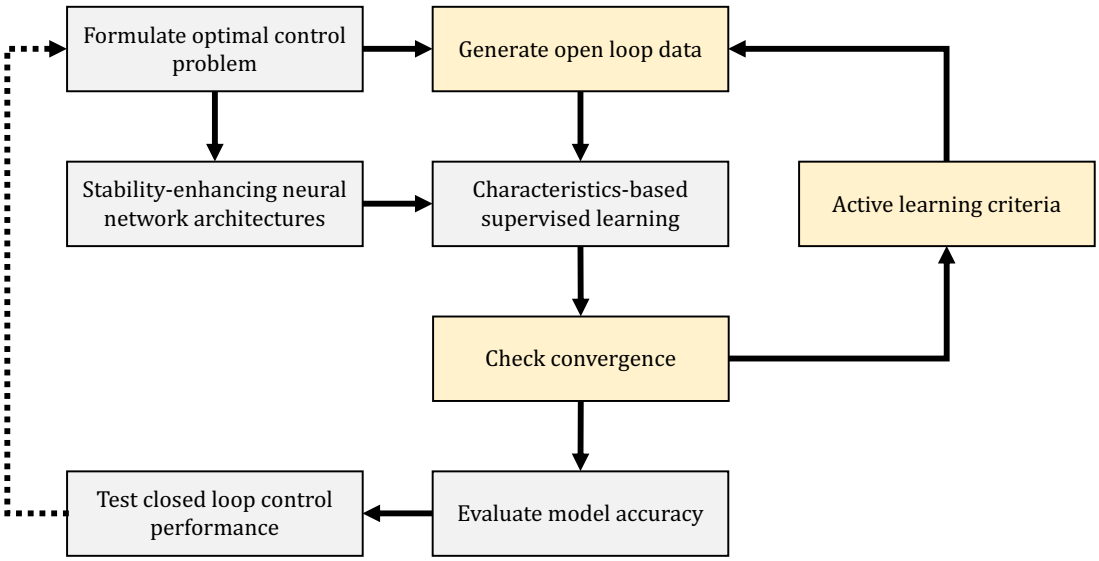


Figure 3.1: Summary of the of the steps and components of the proposed computational framework, highlighting the focus of Chapter 3.

guess for the optimal state trajectory $\mathbf{x}^*(t)$ and open loop optimal control $\mathbf{u}^*(t)$ or costate $\boldsymbol{\lambda}(t)$: without good guesses the OCP solver may fail to converge. To this end in Section 3.2 we present several *warm start* techniques to aid solver convergence, including time marching [66, 69, 67], LQR warm start, and NN warm start. In Section 3.3 we propose an *active learning* framework which includes criteria for selecting data set sizes and locations for new data points. Then in Sections 3.4 and 3.5 we illustrate the application of these ideas to the rigid body attitude control problem from Section 2.4 and stabilization of a Burgers'-like PDE from [62]. The second example shows scalability of the method up to $n = 30$ dimensions. We conclude the chapter with a brief discussion in Section 3.6.

The material presented in this chapter is drawn largely from [97, 98, 64, 100]. Related research on database generation may be found in e.g. [55, 110, 39].

3.1 Computational methods for open loop optimal control

Algorithms for solving open loop OCPs can be broadly classified as *indirect* or *direct* methods [14]. Indirect methods take the "optimize then discretize" approach, solving the OCP by way of the two-point BVP (1.10). Direct methods, on the other hand, take the "discretize then optimize" approach, transforming the OCP into a constrained finite dimensional optimization problem. We discuss indirect methods in Section 3.1.1. Here we also comment on how to solve infinite horizon OCPs (1.4) with indirect methods, and a method proposed by [56, 55] to generate additional data by perturbing a nominal solution. In Section 3.1.2 we cover direct methods. Both indirect and direct methods are causality-free, meaning that they can be easily parallelized so that multiple OCPs can be solved simultaneously.

There is a large literature on computational methods for open loop OCPs. This section is by no means a complete review; for more information we refer the reader to e.g. [14, 121, 70].

3.1.1 Indirect methods

Indirect methods are a broad class of methods which solve the two-point BVP (1.10). By PMP the solution of this BVP satisfies necessary conditions of optimality for the original OCP (1.2). Intuitively, indirect methods take advantage of the known problem structure and analytical work to derive the necessary conditions (1.10). That is, some of the difficulty of optimization is done prior to any numerical computation.

Unfortunately this advantage is a double-edged sword: to derive the BVP dy-

namics one needs to first solve (1.18) analytically to get the optimal control in terms of the costate, then differentiate the optimized Hamiltonian (1.9) with respect to each state variable. These calculations are extremely laborious, and for some OCPs (1.18) may not even have an explicit solution. In such cases propagating the costate dynamics requires additional numerical optimization, which may be prohibitively expensive and contribute to convergence difficulties [14, 70]. For these types of problems direct methods may be preferred.

Once the BVP dynamics have been derived, the primary challenge for indirect methods is that the state trajectory $\mathbf{x}^*(t)$ must be integrated forward in time while the costate $\boldsymbol{\lambda}(t)$ is integrated *backward* from the final condition, $\boldsymbol{\lambda}(t_f) = F_{\mathbf{x}}(\mathbf{x}(t_f))$. Both must be integrated simultaneously, begging the question - where to start?

A straightforward (i.e. naïve) approach is to start with a guess for the initial time costate, $\boldsymbol{\lambda}(0) \approx \widetilde{\boldsymbol{\lambda}}_1(0)$. With this in hand, both state $\mathbf{x}(t)$ and approximate costate $\widetilde{\boldsymbol{\lambda}}(t)$ can be integrated forward. Based on the discrepancy between the integrated values $\widetilde{\boldsymbol{\lambda}}(t_f)$ and the boundary condition $F_{\mathbf{x}}(\mathbf{x}(t_f))$, we adjust the guess $\widetilde{\boldsymbol{\lambda}}_1(0) \to \widetilde{\boldsymbol{\lambda}}_2(0)$ and repeat until the boundary condition is satisfied. This method is called *indirect shooting*. As one might imagine it is fraught with sensitivity problems [14, 119]: unless the initial guess $\widetilde{\boldsymbol{\lambda}}_1(0)$ is very close to the true $\boldsymbol{\lambda}(0)$ then convergence is unlikely.

Indirect collocation is a much more successful method which we use to generate data for the example OCPs in this dissertation[1]. Specifically, we use the *SciPy* [140] implementation of the two-point BVP solver introduced in [72]. This algo-

[1]We have varying degrees of success with the indirect method; some problems are harder than others. The UAV problem in Section 5.3 in particular is very difficult to solve with an indirect method. We achieve some success when we warm start the indirect solver with a direct method.

rithm is based on a three-stage Lobatto IIIa discretization, which is a fourth order accurate collocation formula. In more familiar terms, the right hand side of the BVP (1.10) is discretized with an implicit Runge-Kutta scheme and the trajectory is evaluated at a sequence of collocation points, $0 = t_0 < t_1 < \cdots < t_{N_t} = t_f$. The collocated BVP is then solved with a Newton-type method and more points are added iteratively until the residual meets a specified tolerance [72].

This algorithm is fast, highly accurate, and much less sensitive than indirect shooting. Still, we need to provide a good initial guess for the costate *trajectory* $\boldsymbol{\lambda}(\cdot)$, which in most cases cannot be derived directly from the problem physics. Furthermore, convergence is increasingly dependent on good initializations as we increase the length of the time interval or solve OCPs with initial conditions far from the goal state, $(\mathbf{x}_f, \mathbf{u}_f)$.

3.1.1.1 Solving infinite horizon boundary value problems

In Chapters 4 and 5 we will need to generate data for infinite horizon problems. Because indirect methods do not apply immediately to infinite horizon problems we will need to construct approximate solutions. To this end we recall that the infinite horizon PMP (1.13) is obtained with the limit $t_f \to \infty$ of a finite horizon BVP [114]. To reflect this, we solve the following BVP up to some large final time $t_{f,K} \in (0, \infty)$:

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathcal{H}^*_{\boldsymbol{\lambda}}(\mathbf{x}, \boldsymbol{\lambda}) = \mathbf{f}(\mathbf{x}, \mathbf{u}^*(\mathbf{x}; \boldsymbol{\lambda})), & \mathbf{x}(0) = \mathbf{x}_0, \\ \dot{\boldsymbol{\lambda}}(t) = -\mathcal{H}^*_{\mathbf{x}}(\mathbf{x}, \boldsymbol{\lambda}), & \boldsymbol{\lambda}(t_{f,K}) = \mathbf{0}. \end{cases} \tag{3.1}$$

We then check if the running cost $\mathcal{L}(\mathbf{x}^*(t_{f,K}), \mathbf{u}^*(t_{f,K}))$ is smaller than a desired tolerance. If not, we extend the time horizon $t_{f,K+1} = t_{f,K} + \Delta T$ for some chosen increment $\Delta T > 0$. Taking the previous solution as an initial guess for the BVP

solver, we re-solve (3.1) with the new time horizon, $t_{f,K+1}$. The process is repeated until the running cost is sufficiently small. Notice the apparent similarity to the time marching warm start method described in Section 3.2.1.

Once the running cost is small enough, it follows that the finite horizon solution approximates the solution of the infinite horizon problem as further integration should not change the cost significantly. This conclusion is reasonable as closed loop stability is a necessary condition for finiteness of the value function. Once the solution has converged, recall from Section 2.2.1 that in the infinite horizon case we can discard the time data since the value function, value gradient, and optimal control are all time-independent.

3.1.1.2 Data set generation by perturbations and propagation

[56, 55] propose an alternative approach to construct data sets for certain finite horizon BVPs based on perturbations from a nominal BVP solution. The basic idea is to apply random perturbations to the final state $\mathbf{x}^*(t_f; \mathbf{x}_0)$ and costate $\boldsymbol{\lambda}(t_f; \mathbf{x}_0)$ while still satisfying the final time boundary condition. Then the state and costate can both be propagated backwards by an ODE solver without solving another BVP.

Let us consider this concretely for the BVP (1.10). Suppose that $\mathbf{x}^*\left(t_f; \mathbf{x}_0^{(0)}\right)$ is the endpoint of the nominal trajectory, obtained by solving (1.10) with an indirect method. Then for each $i = 1, \ldots, N_{\text{train}}$ we randomly sample a perturbation $\Delta\mathbf{x}^{(i)} \in \mathbb{R}^n$ of appropriate size and set

$$\mathbf{x}^{(i)}(t_f) = \mathbf{x}^*\left(t_f; \mathbf{x}_0^{(0)}\right) + \Delta\mathbf{x}^{(i)}, \qquad \boldsymbol{\lambda}^{(i)}(t_f) = F_{\mathbf{x}}\left(\mathbf{x}^{(i)}(t_f)\right). \tag{3.2}$$

Integrating the BVP dynamics backwards in time yields an optimal trajectory

that, by uniqueness of ODE solutions, has a different initial condition $\mathbf{x}_0^{(i)} \neq \mathbf{x}_0^{(0)}$.

This algorithm can provide large amounts of additional data at substantially reduced computational burden [55]. There are, of course, some drawbacks. The first drawback is that the method does not work for all kinds of OCPs such as infinite horizon OCPs (1.4). Another drawback is that we have no control over where trajectories go and are unable to select initial conditions. Finally, backwards integration may still not be easy since costate dynamics tend to be highly sensitive and unstable, leading to trajectories which blow up or leave the computational domain. For these reasons we prefer NN warm start (see Section 3.2.3) for generating large data sets.

3.1.2 Direct methods

Unlike indirect methods, direct methods do not attempt to solve the two-point BVP (1.10). Instead, direct methods convert the OCP, which is originally an infinite dimensional (functional) optimization problem, into a large finite dimensional constrained optimization problem. This is done by discretizing the ODEs, the cost function, and any path constraints (not addressed in this work). The discretized problem is then solved with a nonlinear programming solver such as sequential quadratic programming (SQP) [74, 104] or interior point methods [146, 104]. An accessible introduction to direct methods is given by e.g. [70].

In contrast to indirect methods, direct methods do not require deriving the costate dynamics and can more easily handle complicated OCPs such as those with path constraints. This makes them far easier to use and more generally applicable. Direct methods also do not require any initial guess for the costate, taking instead an initial guess for the control $\mathbf{u}^*(t)$. These initial guesses are

easier to construct, and it is known that direct methods generally have a larger convergence basin than indirect methods [14]. On the other hand, direct methods yield less accurate solutions and can be slower than indirect methods [14]. Thus we prefer indirect methods whenever we can derive the BVP dynamics (1.10) and solutions converge reliably. In the context of supervised learning, [133, 82] use a Hermite-Simpson direct method to generate data for finite horizon OCPs.

In this dissertation, whenever we generate data by direct methods we specifically employ PS optimal control [121, 41]. PS methods employ a special collocation technique which exhibits "spectral convergence", meaning that the discretized OCP converges to the original OCP (1.2) extremely fast. This allows one to use far fewer grid points than with other algorithms, and hence solve a much smaller optimization problem. PS methods have the added benefit of the covector mapping theorem [118, 41, 121], which allows one to extract costate data from the solution of the discretized OCP.

For solving infinite horizon OCPs we use a *Legendre-Gauss-Radau* (LGR) PS method [120, 41]. LGR is suited for infinite horizon problems as it solves OCPs over the half open interval $\tau \in [-1, 1)$ via the monotonic transformation[2]

$$\tau = \frac{1+t}{1-t} \iff t = \frac{\tau - 1}{\tau + 1}.$$

Notice that $t = 0$ maps to $\tau = -1$ and $t = \infty$ to $\tau = 1$. Temporal collocation points in the LGR scheme are chosen so that the discretized problems exhibits spectral convergence to the origin OCP (1.4). With even a small number of collocation points we end up with $t_f \gg 1$ and a good approximation to the infinite horizon. In this dissertation we use the LGR PS algorithm to generate

[2]Other transformations are possible [46], but we have found these to be less effective for all problems we tested.

data for the UAV problem in Section 5.3. To the best of our knowledge, this is the first use of PS methods to generate data for supervised learning.

Remark 1. *While direct methods are typically less accurate than indirect methods, they will also for a much larger range of inital guesses. For OCPs where indirect methods do not reliably converge we can get the best of both worlds by first solving each open loop OCP with a direct method, then using the solution as an initial guess for an indirect method. This warm start approach hopefully alleviates convergence difficulties and allows us to obtain a refined, more accurate solution. Note that to do this we need an approximate costate, which can be obtained from PS methods via the covector mapping theorem [118, 41].*

3.2 Warm start strategies

Whether using an indirect or direct solver one needs to provide a initial guesses for the state $\mathbf{x}^*(t)$ and costate $\boldsymbol{\lambda}(t)$ for indirect methods or control $\mathbf{u}^*(t)$ for direct methods. A high quality initial guess will help the solver converge quickly, whereas a worse initial guess can often lead the solver (especially indirect methods) to diverge. The OCP becomes more difficult to solve as we increase the time horizon t_f, distance from the target equilibrium $(\mathbf{x}_f, \mathbf{u}_f)$, and problem complexity.

We can sometimes improve solver convergence by choosing an appropriate problem *scaling*, which is the ratio of magnitudes of state, control, and costates [14, 119]. Unfortunately, scaling is in art: finding the right scaling may not be easy and will not, by itself, making solving the OCP easy. In any case we are motivated to find *warm start* strategies to calculate good initial guesses.

If one has good knowledge of the problem structure then there may be bespoke techniques to come up with good initial guesses. But ideally we would like a set

of techniques which can be applied more generally. One possibility is the *time marching* trick [66, 69, 67] which we describe in Section 3.2.1. In time marching we solve the OCP over a short time horizon and gradually extend the solution to the desired final time. Alternatively we can often use LQR to get a viable initial guess; this *LQR warm start* is discussed in Section 3.2.2. Once some data is available we can train an NN feedback controller, which can then be used to help gather additional data more quickly and reliably; this *NN warm start* idea is introduced in Section 3.2.3. Data for each of the OCP examples in this dissertation are generated using one or more of these warm start techniques.

3.2.1 Time marching

Observing that solving OCPs over long time horizons without any initial guess is difficult, [66, 69, 67] use a continuation technique in which they sequentially extend the solution of an OCP from an initially short time interval to the desired final time t_f.

To illustrate the idea consider the finite horizon OCP (1.2) with initial condition $\mathbf{x}_0$ and final time t_f. We choose a sequence of intermediate times

$$0 < t_1 < t_2 < \cdots < t_K = t_f, \tag{3.3}$$

in which t_1 is small. For the short time interval $[0, t_1]$, we find empirically that the OCP solvers converge given most initial guesses near the initial state $\mathbf{x}_0$. After solving this first OCP we extend the resulting state and costate or control time series over the next time interval $[0, t_2]$. A number of approaches may work for extending the solution over time intervals. We use the straightforward approach of just copying the values of $\mathbf{x}(t_1)$, $\boldsymbol{\lambda}(t_1)$, and $\mathbf{u}(t_1)$. The extended trajectory is

used as the initial guess to solve the BVP over $t \in [0, t_2]$. We repeat this process until $t_K = t_f$, at which we obtain the full solution.

This approach works with indirect methods and fixed finite horizon OCPs with free endpoints (1.2). It is unlikely to work when using an indirect method to solve the infinite horizon BVP (1.13) since the boundary condition $\boldsymbol{\lambda}(t_f) = \mathbf{0}$ must be applied at each intermediate t_k, and there may not exist a feasible solution with $\mathbf{x}^*(0) = \mathbf{x}_0$. It can be used with direct methods for finite horizon OCPs (1.2) and infinite horizon OCPs (1.4).

By appropriately tuning the time sequence $\{t_k\}_{k=1}^K$, we can largely overcome the problem of sensitivity to initial guesses. Unfortunately this may require considerable effort and still not be reliable enough. Other methods like LQR warm start and NN warm start can also be less expensive.

Another related continuation procedure is *space marching*. Here we take the OCP solved for one (easier) initial condition $\mathbf{x}_0^{(0)}$ and repeatedly perturb the solution and resolve the OCP until we reach a desired initial condition, $\mathbf{x}_0^{(1)}$. Intermediate OCP solutions may kept or discarded as desired. This concept is closely related to the method used by [124] where data is generated by taking a random walk from one initial condition $\mathbf{x}_0^{(0)}$ and, taking the previous solution as an initial guess, solving the OCP at each step of the random walk.

3.2.2 LQR warm start

As an alternative to time marching, for systems that can be reliably stabilized by an LQR (1.27) (suboptimally) we might employ LQR warm start. For a given initial condition $\mathbf{x}_0$ we simply simulate the dynamics up to some large final time t_1, closing the loop with an LQR. If the system gets reasonably close to $\mathbf{x}_f$ then

we have a good initial guess for the state trajectory, $\mathbf{x}^*(t) \approx \mathbf{x}(t; \mathbf{x}_0)$ and control $\mathbf{u}^*(t) \approx \mathbf{u}^{\text{LQR}}(\mathbf{x}(t; \mathbf{x}_0))$ for $t \in [0, t_1]$. If desired, a guess for the costate can be obtained with the LQR approximation $\boldsymbol{\lambda}(t) \approx 2\mathbf{P}(\mathbf{x}(t; \mathbf{x}_0) - \mathbf{x}_f)$. While the costate guess is often far from perfect, we find that for many problems it is close enough to facilitate reliable convergence. With this initial guess in hand we can directly solve the OCP over $t \in [0, t_f]$, or combine the LQR initial guess with time marching if $t_1 < t_f$.

In the examples in this chapter we use time marching and NN warm start; LQR warm start is applied in Chapters 4 and 5.

3.2.3 Neural network warm start

Solving many OCPs with time marching and/or LQR warm start can become expensive and may not work well for all problems. Because of this, generating the large data sets necessary to train a NN can be difficult. In such cases we use time marching or LQR warm start only to generate a small initial data set, and adaptively add more points during training (see Section 3.3 below). Note that the newly generated data is not just for this single NN; it can be used for any subsequently trained NNs.

The key to doing this efficiently is simulating the system using the partially-trained NN to close the loop, as in LQR warm start. As training progresses we expect the NN to perform better and better, thus providing better initial guesses which facilitate faster and more reliable data generation. We refer to this strategy as *NN warm start*. Besides being more computationally efficient than time marching, this approach also requires no parameter tuning.

If the NN used for this purpose is a value function model or value gradi-

ent model then we can use it to predict $\boldsymbol{\lambda}(t) \approx \widehat{V}_{\mathbf{x}}\left(t, \mathbf{x}\left(t; \mathbf{x}_0\right); \boldsymbol{\theta}\right)$ or $\boldsymbol{\lambda}(t) \approx \widehat{\boldsymbol{\lambda}}\left(t, \mathbf{x}\left(t; \mathbf{x}_0\right); \boldsymbol{\theta}\right)$ along the closed loop trajectory. If the NN is reasonably accurate then this yields an approximate solution which is reasonably close to the optimal costate. Therefore these NNs enable fast solution with indirect methods. If only a u-NN is available then this is not immediately possible, though we can sometimes make do with using the LQR value gradient approximation.

3.3 Adaptive sampling and model refinement

Since generating each data point requires solving a potentially challenging OCP, it can be expensive to create large data sets which adequately cover the region of interest. This necessitates training on limited data and a method to generate new data in a smart and efficient way. In the proposed computational framework, effective training with small data sets is accomplished by physics-informed learning (Section 2.3.3) and guaranteed locally optimal NN architectures (Section 5.1). In this section we address the latter objective by proposing an algorithm for adaptive data generation and model refinement.

Compared to a standard deep learning problem, we are dealing with small NNs and data sets. Thus we expect second order methods like L-BFGS [85] to be superior for our purposes. Furthermore, we have the freedom to generate additional data throughout the learning process. Indeed, we find that with a small initial data set, which we denote by $\mathcal{D}^1_{\text{train}}$, training a low-fidelity model is very fast using L-BFGS. After this initial round, we want to increase the size of the data set so that it better captures the features of the value function and optimal control. We then continue training the model using this larger data set, $\mathcal{D}^2_{\text{train}}$. We continue this process until some convergence conditions are satisfied.

Our approach is similar to and inspired by a progressive batching method proposed in [19]. The primary difference is that the problem addressed in [19] is a standard supervised learning problem, where a massive data set is available from the start. This allows one to increase the sample size every few iterations, each time taking a completely different sample from the available data. In our problem, we start with only a small amount of data but we can generate more as we go. Because data generation is expensive we would like to generate only as much as is needed, thus motivating a progressive data generation scheme.

3.3.1 Convergence test and sample size selection

In this section we derive a convergence test and sample size selection scheme for the purpose of progressive data generation. To start, suppose that we train the NN in a series of rounds. In each round we use a numerical optimizer like L-BFGS to minimize a loss function $\ell(\cdot)$ given by (2.25), (2.26), or (2.27). Let r be the current training round, $\mathcal{D}^r_{\text{train}}$ be the available data for this training round, and N^r_{train} denote the number of data points in $\mathcal{D}^r_{\text{train}}$.

Throughout this section, to simplify the notation the derivations are made for time-independent (i.e. infinite horizon) problems. We assume that data $\mathbf{x}^{(i)}$ are independent and identically distributed (i.i.d.) uniformly in a domain of interest, $\mathbb{X} \subset \mathbb{R}^n$. Including time-dependence does not change the derivation, as long as we continue to assume data are i.i.d.[3].

Let us assume that at the end of the rth training round, the optimizer termi-

[3]In practice, even though initial conditions are i.i.d., points at future times lie along the optimal trajectories coming from these initial conditions and are thus spatially correlated. Active learning (see Section 3.3.2) also introduces sample dependence. This likely reduces sample variance compared to i.i.d. data, but we still find the numerical tests useful for providing sample size guidelines.

nates after satisfying the first order necessary condition for optimality,

$$\left\| \frac{\partial \ell}{\partial \boldsymbol{\theta}} (\boldsymbol{\theta}; \mathcal{D}^r_{\text{train}}) \right\| \ll 1. \tag{3.4}$$

For true first order optimality, we would like the gradient to be small when evaluated over the entire continuous domain $\mathbb{X}$. In other words, we want

$$\left\| \frac{\partial \ell}{\partial \boldsymbol{\theta}} (\boldsymbol{\theta}; \mathbb{X}) \right\| \ll 1, \tag{3.5}$$

where the Monte Carlo (MC) sums in the varioius loss terms (2.22–2.24) become integrals in the limit as the size of the data set approaches infinity.

If we follow standard machine learning practice, then to see if (3.5) holds we generate a validation data set $\mathcal{D}_{\text{val}}$, separate from the training and test data[4]. Then we check if, for example,

$$\left\| \frac{\partial \ell}{\partial \boldsymbol{\theta}} (\boldsymbol{\theta}; \mathcal{D}_{\text{val}}) \right\| < \varepsilon, \tag{3.6}$$

for some small parameter $\varepsilon > 0$. Standard machine learning convergence tests like (3.6) rely on the fact that $\frac{\partial \ell}{\partial \boldsymbol{\theta}} (\boldsymbol{\theta}; \mathcal{D}_{\text{val}}) \to \frac{\partial \ell}{\partial \boldsymbol{\theta}} (\boldsymbol{\theta}; \mathbb{X})$ in the limit as $N_{\text{val}} \to \infty$, where N_{val} is the number of validation data. But for many practical problems, it may be too expensive to generate enough validation data to make (3.6) meaningful. More importantly, such tests provides no clear guidance in selecting the sample size N^{r+1}_{train} should they not be satisfied.

In this work, we use test data to quantify model accuracy after training is complete (see Section 2.3.4). Indeed, the ability to empirically evaluate solutions

[4]Validation data is not used to optimize the NN, only to check that optimization has converged. It is also not used to evaluate final test accuracy since this would not be an unbiased score.

is a key benefit of the causality-free approach. For the purpose of determining convergence between training rounds, however, we propose a different statistically motivated test which provides information on choosing N_{train}^{r+1}. The idea is simple: since we already assume (3.4) holds, then to ensure that (3.5) is also satisfied, it suffices to check that the error in approximating (3.5) by (3.4) is relatively small.

To motivate this more rigorously, consider a finite sample set $\mathcal{D} \subset \mathbb{X}$ with fixed size N, and assume that the sample points $\mathbf{x}^{(i)} \in \mathcal{D}$ are i.i.d. By design, the sample gradient $\frac{\partial \ell}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}; \mathcal{D})$ is an unbiased estimator for the true gradient (evaluated over the entire continuous domain). That is,

$$\mathbb{E}_{\mathcal{D} \subset \mathbb{X}} \left\{ \frac{\partial \ell}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}; \mathcal{D}) \right\} = \frac{\partial \ell}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}; \mathbb{X}) \,. \tag{3.7}$$

where $\mathbb{E}_{\mathcal{D} \subset \mathbb{X}} \{\cdot\}$ denotes the population mean over all possible finite sample sets $\mathcal{D} \subset \mathbb{X}$ with fixed size N. Intuitively, (3.7) implies that if (3.4) holds, then on average we also have (3.5), as desired. But we must control the MSE or variance of the estimation (3.7). This is given by

$$\begin{aligned}
\text{MSE} \left\{ \frac{\partial \ell}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}; \mathcal{D}) \right\} :=& \mathbb{E}_{\mathcal{D} \subset \mathbb{X}} \left\{ \left\| \frac{\partial \ell}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}; \mathcal{D}) - \frac{\partial \ell}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}; \mathbb{X}) \right\|^2 \right\} && (3.8) \\
=& \mathbb{E}_{\mathcal{D} \subset \mathbb{X}} \left\{ \sum_{j=1}^{p} \left(\frac{\partial \ell}{\partial \theta_j}(\boldsymbol{\theta}; \mathcal{D}) - \frac{\partial \ell}{\partial \theta_j}(\boldsymbol{\theta}; \mathbb{X}) \right)^2 \right\} && (3.9) \\
=& \sum_{j=1}^{p} \mathbb{E}_{\mathcal{D} \subset \mathbb{X}} \left\{ \left(\frac{\partial \ell}{\partial \theta_j}(\boldsymbol{\theta}; \mathcal{D}) - \frac{\partial \ell}{\partial \theta_j}(\boldsymbol{\theta}; \mathbb{X}) \right)^2 \right\} && (3.10) \\
=& \sum_{j=1}^{p} \text{Var}_{\mathcal{D} \subset \mathbb{X}} \left\{ \frac{\partial \ell}{\partial \theta_j}(\boldsymbol{\theta}; \mathcal{D}) \right\}, && (3.11)
\end{aligned}$$

where p denotes the number of parameters $\boldsymbol{\theta}$ and we have used linearity of the

expectation. Then by construction of the loss function,

$$\mathrm{MSE}\left\{\frac{\partial \ell}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta};\mathcal{D})\right\} = \sum_{j=1}^{p} \mathrm{Var}_{\mathcal{D}\subset\mathbb{X}}\left\{\frac{1}{N}\sum_{i=1}^{N}\frac{\partial \ell}{\partial \theta_j}\left(\boldsymbol{\theta};\mathbf{x}^{(i)}\right)\right\}. \tag{3.12}$$

Using the simplifying assumption that $\mathbf{x}^{(i)}$ are i.i.d., this becomes

$$\mathrm{MSE}\left\{\frac{\partial \ell}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta};\mathcal{D})\right\} = \frac{1}{N^2}\sum_{j=1}^{p}\sum_{i=1}^{N}\mathrm{Var}_{\mathbf{x}\in\mathbb{X}}\left\{\frac{\partial \ell}{\partial \theta_j}(\boldsymbol{\theta};\mathbf{x})\right\} \tag{3.13}$$

$$= \frac{1}{N}\sum_{j=1}^{p}\mathrm{Var}_{\mathbf{x}\in\mathbb{X}}\left\{\frac{\partial \ell}{\partial \theta_j}(\boldsymbol{\theta};\mathbf{x})\right\}. \tag{3.14}$$

If the estimation error is small, then the sample mean is likely to be a good approximation of the true mean. Hence we expect that $\left\|\frac{\partial \ell}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta};\mathbb{X})\right\|$ will also be small as desired. To this end, we require that the *root* MSE not be too large compared to the expected gradient. Specifically, we check if

$$\sqrt{\mathrm{MSE}\left\{\frac{\partial \ell}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta};\mathcal{D})\right\}} \leq C\left\|\mathbb{E}_{\mathcal{D}\subset\mathbb{X}}\left\{\frac{\partial \ell}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta};\mathcal{D})\right\}\right\|_1, \tag{3.15}$$

where $C > 0$ is a scalar parameter. On the right hand side we use the ℓ^1 norm instead of the ℓ^2 as it is less sensitive to outliers in the loss gradient. We find that this makes the test less likely to suggest unreasonably large sample sizes.

In practice, evaluating of the true population variances on the left hand side of (3.15) is computationally intractable. But we can approximate these by the corresponding *sample* variances[5] taken over all data $\mathbf{x}^{(i)} \in \mathcal{D}^r_{\text{train}}$, which we denote

[5]Computing a large number of individual gradients can still be too costly, so we often evaluate sample variances over a smaller subset of the training data.

by $\mathrm{Var}_{\mathbf{x}^{(i)} \in \mathcal{D}^r_{\text{train}}} \{\cdot\}$:

$$\mathrm{MSE}\left\{\frac{\partial \ell}{\partial \boldsymbol{\theta}}\left(\boldsymbol{\theta}; \mathcal{D}\right)\right\} \approx \frac{1}{N^r_{\text{train}}} \sum_{j=1}^{p} \mathrm{Var}_{\mathbf{x}^{(i)} \in \mathcal{D}^r_{\text{train}}} \left\{\frac{\partial \ell}{\partial \theta_j}\left(\boldsymbol{\theta}; \mathbf{x}^{(i)}\right)\right\}. \tag{3.16}$$

Similarly, we approximate the expected gradient on the right hand side of (3.15) by the sample gradient and arrive at the following practical convergence criterion:

$$\sqrt{\sum_{j=1}^{p} \mathrm{Var}_{\mathbf{x}^{(i)} \in \mathcal{D}^r_{\text{train}}} \left\{\frac{\partial \ell}{\partial \theta_j}\left(\boldsymbol{\theta}; \mathbf{x}^{(i)}\right)\right\}} \leq C \left\| \frac{\partial \ell}{\partial \boldsymbol{\theta}}\left(\boldsymbol{\theta}; \mathcal{D}^r_{\text{train}}\right) \right\|_1 \sqrt{N^r_{\text{train}}}. \tag{3.17}$$

If the convergence test (3.17) is satisfied, then it is likely that the expected gradient $\left\| \frac{\partial \ell}{\partial \boldsymbol{\theta}}\left(\boldsymbol{\theta}; \mathbb{X}\right) \right\|$ is also small. In other words, we expect that the parameters $\boldsymbol{\theta}$ satisfy the first order optimality conditions evaluated over the entire domain, so we can stop optimization. Satisfaction of (3.17) does not imply that the trained model is good – merely that seeing more data would probably not improve it significantly. On the other hand, when the criterion is not met, then it guides us in selecting the next sample size N^{r+1}_{train}. Concretely, suppose that the ratio of the sample variance to the sample gradient doesn't change significantly by increasing the size of the data set, i.e.

$$\frac{\sqrt{\sum_{j=1}^{p} \mathrm{Var}_{\mathbf{x}^{(i)} \in \mathcal{D}^{r+1}_{\text{train}}} \left\{\frac{\partial \ell}{\partial \theta_j}\left(\boldsymbol{\theta}; \mathbf{x}^{(i)}\right)\right\}}}{\left\| \frac{\partial \ell}{\partial \boldsymbol{\theta}}\left(\boldsymbol{\theta}; \mathcal{D}^{r+1}_{\text{train}}\right) \right\|_1} \approx \frac{\sqrt{\sum_{j=1}^{p} \mathrm{Var}_{\mathbf{x}^{(i)} \in \mathcal{D}^{r}_{\text{train}}} \left\{\frac{\partial \ell}{\partial \theta_j}\left(\boldsymbol{\theta}; \mathbf{x}^{(i)}\right)\right\}}}{\left\| \frac{\partial \ell}{\partial \boldsymbol{\theta}}\left(\boldsymbol{\theta}; \mathcal{D}^{r}_{\text{train}}\right) \right\|_1}.$$

Then the appropriate choice of N^{r+1}_{train} to satisfy (3.17) after the next round is such that

$$M N^r_{\text{train}} \geq N^{r+1}_{\text{train}} \geq \frac{\sum_{j=1}^{p} \mathrm{Var}_{\mathbf{x}^{(i)} \in \mathcal{D}^r_{\text{train}}} \left\{\frac{\partial \ell}{\partial \theta_j}\left(\boldsymbol{\theta}; \mathbf{x}^{(i)}\right)\right\}}{\left(C \left\| \frac{\partial \ell}{\partial \boldsymbol{\theta}}\left(\boldsymbol{\theta}; \mathcal{D}^r_{\text{train}}\right) \right\|_1\right)^2}, \tag{3.18}$$

where $M > 1$ is a scalar parameter which prevents the data set size from growing

too quickly. In all examples we set $M = 2$.

The convergence test (3.17) and sample size selection scheme (3.18) derived above are close to that used in [19], except that we employ the ℓ^1 norm of the sample gradient in the denominator instead of the ℓ^2 norm. We prefer the ℓ^1 norm because it is less sensitive to outliers in the loss gradient. Intuitively, this improves robustness by making the test less likely to suggest unreasonably large sample sizes. We also contribute a different derivation, coming from the perspective of progressive data generation as opposed to sampling from a large pre-existing data set. Finally, like [19] our results are not specific to optimal control. They can be applied to many data-driven optimization problems where data is scarce but can be generated over time. Notably, these results facilitate the use of existing algorithms for second order and constrained optimization in such applications.

3.3.2 Active learning criteria

The sample size selection criterion (3.18) we propose indicates how many data are necessary to satisfy the convergence test (3.17), assuming a uniform sampling from the domain. In practice, since all the data we generate will be new, we can choose to generate new data where it is needed most. This is a form of *active learning* [39]. This condition for generating new data can be interpreted in many ways. In this work we propose concentrating samples where $\left\|\widehat{V}_{\mathbf{x}}(\cdot)\right\|$ is large. Here $\widehat{V}_{\mathbf{x}}(\cdot)$ can refer to the gradient of a V-NN or a λ-NN. Regions of the value function with large gradients tend to be steep or complicated, and thus may benefit from having more data to learn from. Furthermore, these regions typically correspond to places where the control effort is large and hence we would like controllers to be especially accurate there.

Specifically, for each initial condition we want to integrate, we can first randomly sample a set of N_c candidate initial conditions from $\mathbb{X}$. A quick pass through the NN yields the predicted gradient $\widehat{V}_{\mathbf{x}}\left(0, \mathbf{x}_0^{(i)}\right)$ at all candidate points, $i = 1, \ldots, N_c$. Here the predicted value gradient is used since the true value gradient is unknown. We choose the point(s) with the largest predicted gradient norms, $\left\|\widehat{V}_{\mathbf{x}}\left(0, \mathbf{x}_0^{(i)}\right)\right\|$, and use NN warm start to solve the corresponding open loop OCPs. This process is repeated for new sets of candidate initial conditions until we obtain the desired amount of data (each trajectory may contain hundreds of data points).

Note that if we do not have a model which can predict $\widehat{V}_{\mathbf{x}}(\cdot)$ then we could employ alternative criteria which places points in regions where the NN controller performs poorly in closed loop simulations. This strategy has not yet been tested and is left for future work. An alternative active learning strategy for optimal control data based on Gaussian process has been proposed by [39], and [110] generate optimal control databases using criteria which try to fill the space $\mathbb{X}$.

3.3.3 The full training algorithm

By combining the physics-informed training algorithm from Section 2.3.3, the convergence test and sample size selection criteria from Section 3.3.1, active learning criteria suggested in Section 3.3.2, and the NN warm start technique from Section 3.2.3, we arrive at the full training procedure summarized in Algorithm 1.

Algorithm 1 enables us to build up a rich data set and a high-fidelity NN model of the value function or value gradient. Moreover, the data set is not constrained to lie within a small neighborhood of some nominal trajectory. It can contain points from the entire domain $\mathbb{X}$, and we can concentrate more data near complicated

Algorithm 1 Adaptive sampling and model refinement

Generate $\mathcal{D}_{\text{train}}^1$ using time marching or LQR warm start
for $r = 1, 2, \ldots$ **do**
  Solve optimization problem (2.25), (2.26), or (2.27) to update NN parameters $\boldsymbol{\theta}$
  if (3.17) is satisfied **then**
    return optimized parameters $\boldsymbol{\theta}$
  else
    while (3.18) is not satisfied **do**
      Sample candidate initial conditions $\mathbf{x}_0^{(i)}$, $i = 1, \ldots, N_c$
      In parallel, predict $\left\| \widehat{V}_{\mathbf{x}} \left(0, \mathbf{x}_0^{(i)}\right) \right\|$, $i = 1, \ldots, N_c$
      Choose the initial conditions with largest predicted gradient norm and use NN warm start to solve the corresponding OCPs
      Add the resulting trajectories to $\mathcal{D}_{\text{train}}^{r+1}$
    end while
  end if
end for

features of the value function. As we progressively refine the NN model, we can adjust the loss weights $\mu_{\boldsymbol{\lambda}}$ and $\mu_{\mathbf{u}}$, as well as other hyperparameters such as the optimizer convergence tolerance and the number of terms in the L-BFGS Hessian approximation [85]. As the NN is already partially-trained, fewer iterations should be needed for convergence in each round so we can afford to make each iteration more expensive.

3.4 Example: Rigid body attitude control

Let us illustrate the application of Algorithm 1, the adaptive sampling and model refinement process, to the rigid body attitude control problem from Section 2.4.

We solve the following OCP (2.48):

$$\begin{cases} \underset{\mathbf{u}(\cdot)}{\text{minimize}} & \mathcal{J}\left[\mathbf{u}(\cdot)\right] = F\left(\mathbf{v}\left(t_f\right), \boldsymbol{\omega}\left(t_f\right)\right) + \displaystyle\int_0^{t_f} \mathcal{L}(\mathbf{v}, \boldsymbol{\omega}, \mathbf{u}) d\tau, \\ \text{subject to} & \dot{\mathbf{v}} = \mathbf{E}(\mathbf{v})\boldsymbol{\omega}, \\ & \dot{\boldsymbol{\omega}} = \mathbf{J}\left[-\boldsymbol{\omega} \times \mathcal{R}_{\mathbf{v}}(\mathbf{h}) + \mathbf{B}\mathbf{u}\right]. \end{cases}$$

Details and parameters are given in Section 2.4. To start, in Section 3.4.1 we evaluate the reliability and speed of solving OCPs when using time marching and NN warm start. We find that NN warm start is considerably more effective, and is thus the preferred method once an NN becomes available. Then in Section 3.4.2 we show how a model can be trained with Algorithm 1, and that the active learning criteria help make the database more useful for learning the value gradient.

3.4.1 Comparing time marching and NN warm start

In this section we investigate the convergence of the indirect OCP solver [72] combined with time marching and NN warm start. Results are given in Table 3.1 and Table 3.2, respectively. For these tests, we randomly sample 10^6 candidate initial conditions and pick $N_c = 1000$ points with the largest predicted gradient norm, $\left\|\widehat{V}_{\mathbf{x}}(\cdot)\right\|$, using an NN not tested in Table 3.2. Initial conditions with large gradient norm tend to be located in regions where the value function is steep and the control effort is large, and may thus be more difficult to solve. The set of initial conditions is fixed for all tests.

In the first row of Table 3.1, we attempt to solve the OCP with no time marching, i.e. over the entire time interval without constructing any initial guess. In this case, the proportion of convergent solutions is extremely small, obviating the need for good initial guesses. As shown in Table 3.1, we reliably obtain

K	% OCP convergence	mean integration time
1	0.3%	0.37 s
2	38.7%	0.44 s
3	76.2%	0.40 s
4	92.9%	0.45 s
8	98.4%	0.53 s

Table 3.1: Convergence of OCP solutions for (2.48) when using time marching, depending on the number of steps in the sequence $\{t_k\}_{k=1}^K$. The case $K = 1$ corresponds to a direct solution attempt over the whole time interval with no time marching. OCP integration time is measured only on successful attempts – failed solution attempts usually take much longer.

$\mu_{\boldsymbol{\lambda}}$	training time	gradient RMℓ^2	% OCP conv.	mean int. time
10^{-8}	7 s	1.2×10^{-1}	88.0%	0.50 s
10^{-4}	19 s	6.5×10^{-2}	98.6%	0.48 s
1	23 s	2.1×10^{-2}	99.7%	0.44 s

Table 3.2: Convergence of OCP solutions for (2.48) when using NN warm start with NNs of varying gradient prediction accuracy. All NNs are trained on the same data set with only $N_{\text{train}} = 64$ data. OCP integration time is measured only on successful attempts.

solutions for this problem when we use at least $K = 4$ time intervals. We note that the initial conditions are purposefully chosen to be difficult – if we simply take uniform samples from the domain $\mathbb{X}$ defined in (2.52), the proportion of convergent solutions increases significantly.

In Table 3.2, we present results using NN warm start. We train several NNs models of $V(t = 0, \mathbf{x})$ on a data set of only $N_{\text{train}} = 64$ points. Because the data set is so small, *each NN takes only seconds to finish training.* We also experiment with using different gradient loss weights $\mu_{\boldsymbol{\lambda}}$ for each NN (the control loss is fixed at $\mu_{\mathbf{u}} = 0$). This directly impacts the accuracy in predicting the costate, $\boldsymbol{\lambda}(t; \mathbf{v}_0, \boldsymbol{\omega}_0) \approx \widehat{V}_{\mathbf{x}}(\mathbf{v}_0, \boldsymbol{\omega}_0)$, which in turn is needed to synthesize optimal controls.

Even with these low-fidelity models, the rate of successful OCP convergence is just as high as when using $K = 4$ time intervals for time marching. The quality

of initial guesses improves with better costate prediction, and it is not difficult to exceed 99% convergence. For this problem, the speed of the two methods is about the same. However, when we consider higher-dimensional problems in Section 3.5.2, we find that NN warm start significantly improves both reliability and efficiency.

3.4.2 Training with adaptive data generation

Performing a thorough systematic study of the adaptive sampling and model refinement technique proposed in Section 3.3 is rather complicated. This is because results depend on various hyperparameter settings and random chance, since data points are chosen in a (partially) random way and the randomly-initialized NN training problem is highly non-convex. For this reason, in this section we show a just few conservative results which we feel illustrate the potential of the method.

We start from a data set with $N^1_{\text{train}} = 64$ points and set the gradient loss weight to $\mu_{\boldsymbol{\lambda}} = 10$ and the convergence parameter in (3.17) to $C = 0.25$. We apply Algorithm 1 to train an NN model of $V\,(t = 0, \mathbf{x})$: after each round, we check the convergence criterion (3.17) and increase the number of training data according to (3.18). For the active learning criteria we consider $N_c = 2$ candidate points. Each data set includes all previously generated data, and we generate additional data as needed using NN warm start. With these configurations the sample size selection criteria (3.18) specifies $N^2_{\text{train}} = 114$, $N^3_{\text{train}} = 201$, $N^4_{\text{train}} = 402$, and $N^5_{\text{train}} = 669$. The convergence test (3.17) is passed at the end of training round $r = 5$. Figure 3.2 shows the progress of the test error over the course of training.

The final value function RMAE is 2.0×10^{-4}. This is about 50% more accurate than the sparse grid method with 44,698 points but uses only about 1.5% as much

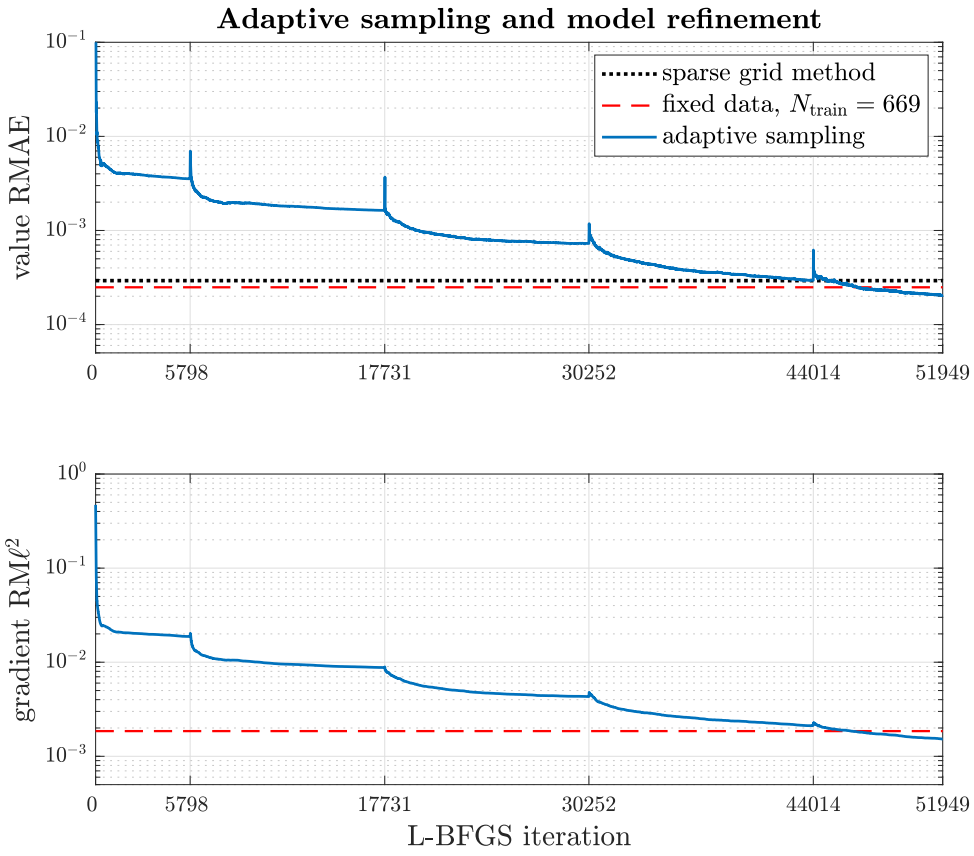


Figure 3.2: Progress of adaptive sampling and model refinement for the rigid body problem (2.48), compared to training on fixed data sets and the sparse grid characteristics method. Spikes in the error correspond to the start of new training rounds and expansion of the training data set.

data. The final gradient RMℓ^2 is 1.5×10^{-3}. As shown in Figure 3.2, the NN trained with Algorithm 1 is more accurate than an NN trained on a fixed data set of $N_{\text{train}} = 669$ samples. That is to say, the active learning method facilitates more acurate gradient predictions using fewer data. These results highlight the main advantages of the adaptive sampling and model refinement method: the ability to overcome an initial lack of data, efficiently generate a large data set, and improve gradient prediction accuracy which is needed for effective control. To fully realize the potential of the method, hyperparameters like $\mu_{\boldsymbol{\lambda}}$, $\mu_{\mathbf{u}}$, C, and

internal optimizer parameters need to be adjusted in each round. Development of algorithms to do this adaptively remains a topic for future research.

3.5 Example: Unstable Burgers'-like PDE

In this section, we test our method on high-dimensional nonlinear systems arising from a Chebyshev PS discretization of a one-dimensional forced Burgers'-like PDE. An infinite horizon version of this problem is studied in [62], in which the value function is approximated using polynomials. In [62] the discretized problem has $n = 12$ state dimensions; we solve it for $n = 10, 20$, and 30. We note that in [62], separability of the nonlinear dynamics is required to compute the high-dimensional integrals necessary in the Galerkin formulation. Our method does not require this restriction, although it does apply to this problem.

As in [62], let ξ denote the spatial variable on the domain $\xi \in [-1, 1]$, and let $X(t, \xi) : [0, t_f] \times [-1, 1] \to \mathbb{R}$ satisfy the following one-dimensional controlled PDE with Dirichlet boundary conditions:

$$\begin{cases} X_t = XX_\xi + \nu X_{\xi\xi} + \alpha X e^{\beta X} + I_\Omega(\xi)u, & t > 0,\ \xi \in (-1, 1), \\ X(t, -1) = X(t, 1) = 0, & t > 0, \\ X(0, \xi) = X_0, & \xi \in (-1, 1). \end{cases} \tag{3.19}$$

For notational convenience we have written $X = X(t, \xi)$, and as before we denote $X_t = \partial X/\partial t$, $X_\xi = \partial X/\partial \xi$, and $X_{\xi\xi} = \partial^2 X/\partial \xi^2$. The scalar-valued control $u = u(t, X)$ is actuated only on Ω, the support of the indicator function

$$I_\Omega(\xi) \coloneqq \begin{cases} 1, & \xi \in \Omega, \\ 0, & \xi \notin \Omega\ . \end{cases} \tag{3.20}$$

The PDE-constrained optimal control problem is

$$\begin{cases} \underset{u(\cdot)}{\text{minimize}} & \mathcal{J}\left[u(\cdot)\right] = \dfrac{W_1}{2}\left\|X\left(t_f\right)\right\|^2_{L^2_{(-1,1)}} + \displaystyle\int_0^{t_f} \mathcal{L}(X,u)dt, \\ \text{subject to} & X_t = XX_\xi + \nu X_{\xi\xi} + \alpha X e^{\beta X} + I_\Omega(\xi)u, \\ & X(t,-1) = X(t,1) = 0, \end{cases} \tag{3.21}$$

where the running cost is

$$\mathcal{L}(X,u) = \frac{1}{2}\left\|X(t)\right\|^2_{L^2_{(-1,1)}} + \frac{W_1}{2}u^2(\tau,X), \tag{3.22}$$

and we set

$$\Omega = (-0.5,-0.2), \quad \nu = 0.2, \quad \alpha = 1.5, \quad \beta = -0.1, \quad W_1 = 1, \quad W_2 = 0.1, \quad t_f = 8.$$

For reference the L^2 norm is defined as

$$\left\|X(t)\right\|^2_{L^2_{(-1,1)}} \coloneqq \int_{-1}^{1} X^2(t,\xi)d\xi. \tag{3.23}$$

In this problem, the goal of stabilizing $X(t,\xi)$ is made more challenging by the added nonlinear reaction term, $\alpha X e^{\beta X}$, which renders the origin unstable. This can be seen clearly in Figure 3.3a.

3.5.1 Discretizing the PDE-constrained problem

To solve (3.21) using our framework, we perform Chebyshev PS collocation to transform the PDE (3.19) into a system of ODEs. Following [136, 62], let

$$\xi_j = -\cos\left(\frac{j\pi}{n+1}\right), \qquad j = 0,1,\ldots n,n+1, \tag{3.24}$$

where n is the number of internal (non-boundary) Chebyshev collocation points. After accounting for boundary conditions, the discretized state is defined as

$$\mathbf{x}(t) \coloneqq \begin{pmatrix} X(t,\xi_1) & X(t,\xi_2) & \dots & X(t,\xi_n) \end{pmatrix}^T : [0,t_f] \to \mathbb{R}^n, \tag{3.25}$$

and the PDE (3.19) becomes a system of ODEs in n dimensions:

$$\dot{\mathbf{x}} = \mathbf{x} \odot \mathbf{D}\mathbf{x} + \nu \mathbf{D}^2 \mathbf{x} + \alpha \mathbf{x} \odot e^{\beta \mathbf{x}} + \mathbb{I}_\Omega u, \tag{3.26}$$

In the above, "$\odot$" denotes element-wise multiplication (Hadamard product), $\mathbb{I}_\Omega$ is the discretized indicator function, and $\mathbf{D}, \mathbf{D}^2 \in \mathbb{R}^{n \times n}$ are the internal parts of the first and second order Chebyshev differentiation matrices, which are obtained by deleting the first and last rows and columns of the full matrices [136]. This discretization automatically enforces the boundary conditions. Finally, since $X(t,\xi)$ is collocated at Chebyshev nodes, the inner product appearing in the cost function is conveniently approximated by Clenshaw-Curtis quadrature [136]:

$$\begin{cases} \|X(t)\|^2_{L^2_{(-1,1)}} = \displaystyle\int_{-1}^{1} X^2(t,\xi) d\xi \approx \|\mathbf{x}\|^2_{\mathbf{Q}}, \\ \|\mathbf{x}\|_{\mathbf{Q}} \coloneqq \sqrt{\mathbf{x}^T \mathbf{Q} \mathbf{x}}, \\ \mathbf{Q} = \operatorname{diag}\begin{pmatrix} w_1, & w_2, & \dots, & w_n \end{pmatrix}. \end{cases} \tag{3.27}$$

where $w_1, \dots, w_n$ are the internal Clenshaw-Curtis quadrature weights. Now the original OCP (3.21) can be reformulated as a quadratic cost ODE-constrained problem:

$$\begin{cases} \underset{u(\cdot)}{\text{minimize}} & \mathcal{J}_n\left[\mathbf{u}\left(\cdot\right)\right] = \dfrac{W_1}{2} \left\|\mathbf{x}\left(t_f\right)\right\|^2_{\mathbf{Q}} + \displaystyle\int_0^{t_f} \frac{1}{2} \left(\|\mathbf{x}\|^2_{\mathbf{Q}} + W_2 u^2\right) dt, \\ \text{subject to} & \dot{\mathbf{x}} = \mathbf{x} \odot \mathbf{D}\mathbf{x} + \nu \mathbf{D}^2 \mathbf{x} + \alpha \mathbf{x} \odot e^{\beta \mathbf{x}} + \mathbb{I}_\Omega u. \end{cases} \tag{3.28}$$

The state dimension n of the OCP (3.28) can be adjusted, presenting a good opportunity to test the scalability of our algorithms. For this problem, we learn the value function $V = V(t, \mathbf{x})$ with time-dependence, rather than just $V(0, \mathbf{x})$ as in Sections 2.4 and 3.4. Consequently, the resulting controls can be implemented as time-dependent controls or with a moving horizon, if desired. Following [62] we consider initial conditions drawn from the domain

$$\mathbb{X}_0 = \{\mathbf{x} \in \mathbb{R}^n | -2 \leq x_j \leq 2, j = 1, 2, \ldots, n\}. \tag{3.29}$$

3.5.2 Comparing time marching and NN warm start

In our experience, generating the initial training data set can be the most computationally demanding part of the process, especially as the problem dimension n increases. Consequently, for difficult high-dimensional problems it may be impractical to generate a large-enough data set from scratch. This obstacle can be substantially mitigated by using partially-trained/low-fidelity NNs to aid in further data generation. In this section, we briefly compare the reliability and speed of OCP convergence between time marching and NN warm start. These experiments demonstrate the importance of NN guesses for high-dimensional data generation.

For each of $n = 10$, 20, and 30, we randomly sample a set of $N_c = 1000$ candidate points from the domain $\mathbb{X}_0$ defined in (3.29). From these we choose 100 points with the largest predicted value gradient. The set of initial conditions is fixed for each n. Next we proceed as in Section 3.4.1, solving each OCP by time marching with different number of time marching iterations, K. We tune each time sequence to improve convergence as much as possible. Results are

summarized in Table 3.3.

We then solve the same OCPs directly over the whole time interval $t \in [0, 8]$ with NN warm start. These NNs are trained to approximate the value function on fixed data sets containing only 30 trajectories, but with different gradient loss weights $\mu_{\boldsymbol{\lambda}}$ resulting in varying costate prediction accuracy (we again keep $\mu_{\mathbf{u}} = 0$). We also limit the number of L-BFGS iterations so that each model is trained only for a short time. Results are given in Table 3.4. For both time marching and NN warm start experiments we use an indirect solver [72].

As before, we find that even NNs with relatively large costate prediction error enable consistently convergent BVP solutions. Time-marching also works once the sequence of time steps $\{t_k\}_{k=1}^K$ is properly tuned, but the speed of this method scales poorly with n. Now the advantage of utilizing NNs to aid in data generation becomes clear: when n is large, the average time needed for convergence when using NN warm start is drastically lower than that of the time marching trick. This approach also requires no tuning of the time marching sequence. Because low-fidelity NNs are quick to train, training such a NN and then using it to aid in data generation is the most efficient strategy for building larger data sets.

3.5.3 Learning high-dimensional value functions

Here we apply Algorithm 1 to learn approximate solutions to (3.28) in $n = 10$, 20, and 30 dimensions. We focus on demonstrating what is possible using our approach, rather than carrying out a detailed study of its effectiveness under different parameter tunings. In [62] an infinite horizon version of the problem is solved up to $n = 12$, but the accuracy of the solution is not readily verifiable. The ability to conveniently measure model accuracy for general high-dimensional

n	K	% OCP convergence	mean integration time
10	4	40%	0.7 s
	6	83%	0.8 s
	10	90%	1.3 s
20	4	46%	3.6 s
	5	86%	4.2 s
	6	99%	4.7 s
30	4	47%	11.3 s
	6	90%	14.6 s
	8	100%	19.1 s

Table 3.3: Convergence of OCP solutions for (3.28) when using time marching, depending on the problem dimension, n, and the number of steps in the sequence $\{t_k\}_{k=1}^K$. OCP integration time is measured only on successful attempts.

n	$\mu_{\boldsymbol{\lambda}}$	training time	gradient RMℓ^2	% OCP conv.	mean int. time
10	10^{-8}	20 s	5.0×10^{-2}	96%	0.8 s
	10^{-4}	31 s	2.4×10^{-2}	99%	0.8 s
	1	56 s	1.0×10^{-2}	88%	0.6 s
20	10^{-8}	29 s	7.4×10^{-2}	74%	2.9 s
	10^{-4}	47 s	2.0×10^{-2}	91%	2.5 s
	1	76 s	1.3×10^{-2}	98%	2.5 s
30	10^{-8}	38 s	5.7×10^{-2}	79%	7.1 s
	10^{-4}	125 s	1.4×10^{-2}	94%	6.9 s
	1	189 s	1.4×10^{-2}	96%	7.1 s

Table 3.4: Convergence of OCP solutions for (3.28) when using NN warm start with NNs of varying gradient prediction accuracy. All NNs are trained on data sets with 30 trajectories. OCP integration time is measured only on successful attempts.

problems with *no known analytical solution* is a key advantage of the causality-free framework.

For each discretized OCP, $n = 10$, 20, and 30, we apply the time marching strategy to build an initial training data set $\mathcal{D}^1_{\text{train}}$ from 30 uniformly sampled initial conditions, $\mathbf{x}_0^{(i)} \in \mathbb{X}_0$, $i = 1, \ldots, 30$. Recall that for each initial condition we get an optimal trajectory $\left\{\mathbf{x}^*\left(t_k^{(i)}; \mathbf{x}_0^{(i)}\right)\right\}_{k=1}^{N_t^{(i)}}$, evaluated at collocation points

n	num. trajectories	training time	value RMAE	gradient RMℓ^2
10	163	25 min	1.7×10^{-4}	1.4×10^{-3}
20	128	48 min	5.0×10^{-4}	2.2×10^{-3}
30	145	62 min	1.3×10^{-3}	4.2×10^{-3}

Table 3.5: Test accuracy of NNs for solving the collocated Burgers'-like OCP (3.28), depending on the state dimension n. Training time includes time spent generating additional data according to Algorithm 1.

$t_k^{(i)} \in [0, t_f]$ chosen by the solver. Typically this can be a few hundred per initial condition, depending on the state dimension n and the solver tolerances. Since these data sets can be get quite large, we often train on randomly selected subsets of the data. This can significantly improve training speed without sacrificing accuracy. When neeeded, we solve additional OCPs to expand the data set as per Algorithm 1. We use the same NN architecture as for the rigid body problem, with $L = 3$ hidden layers with $w = 64$ neurons each. We set $C = 0.3$, 1.3, and 1.8 for $n = 10$, 20, and 30, respectively; and set $N_c = 2$, $\mu_{\boldsymbol{\lambda}} = 10$, and $\mu_{\mathbf{u}} = 0$ in all cases.

In Table 3.5, we present test accuracy results for the trained NNs. We include the value RMAE and the gradient RMℓ^2. Accuracy is measured empirically on independently generated test data sets comprised of trajectories from 50 uniformly sampled initial conditions. We find that the trained NNs accurately predict both the value function and its gradient, even in $n = 30$ dimensions.

Table 3.5 also shows the total number of sample trajectories seen by the NN, including the initial data $\mathcal{D}^1_{\text{train}}$. It may seem surprising that we are able to reach similar levels of accuracy in higher dimensions with similar numbers of sample trajectories. This may happen because the BVP solver usually needs more collocation points for larger problems, thus producing more data per trajectory. Consequently, fewer trajectories are needed to fulfill the data set size recommendation

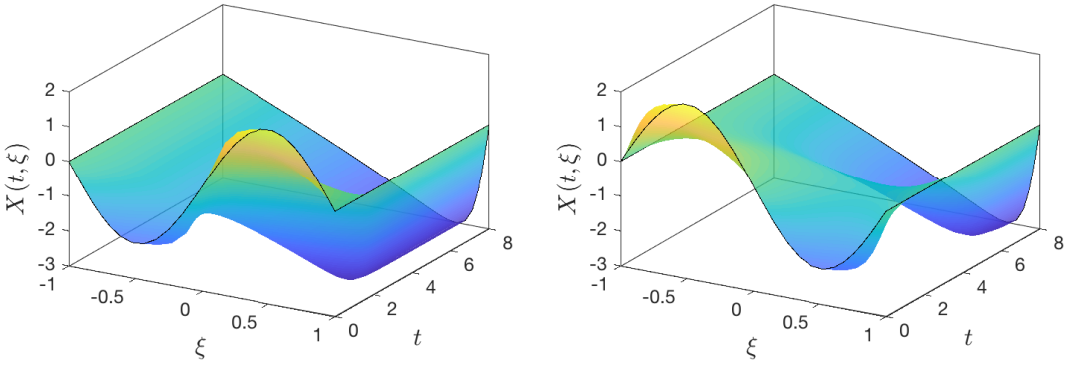
(3.18). Analogously, in Section 3.4 we used data only for $t = 0$, so we needed thousands of trajectories to fill the state space. This suggests that learning the time-dependent value function can be more efficient than learning $V(0, \mathbf{x})$ only. Note that, if preferred, the time-dependent controller can still be used with a moving horizon like in Section 2.4. Ultimately, however, infinite horizon problems prove to be the most data-efficient because all the data from all trajectories are valid.

Lastly, Table 3.5 shows the training time for each NN, including time spent testing convergence and generating additional trajectories on the fly, but not time spent generating the initial data. As seen in Section 3.5.2, generating data becomes the most expensive computation as n increases, but even so we find that computational effort scales reasonably with the problem dimension. Furthermore, it is possible to obtain a rough low-fidelity NN model in just minutes as shown in Table 3.4, which in turn allows for more efficient data generation. This demonstrates the viability of the proposed methodology for solving high-dimensional OCPs.

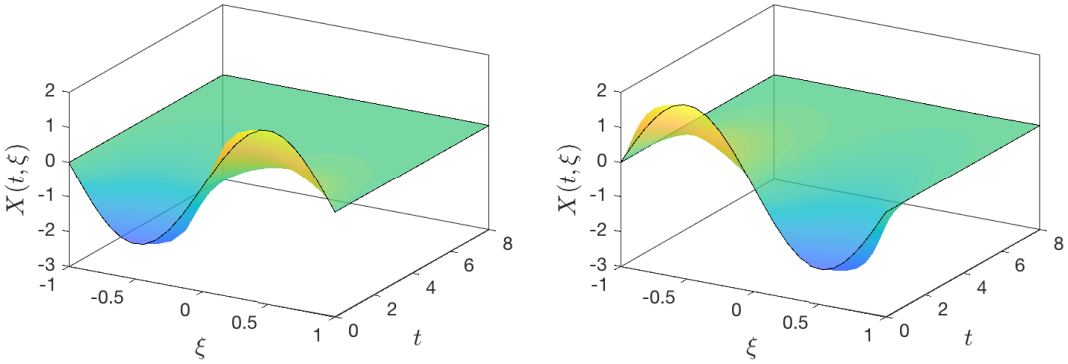
3.5.4 Closed loop simulations

We conclude the example with some closed loop simulations to demonstrate that the feedback control output by the trained NN not only stabilizes the high-dimensional system, but that it is close to the true optimal control.

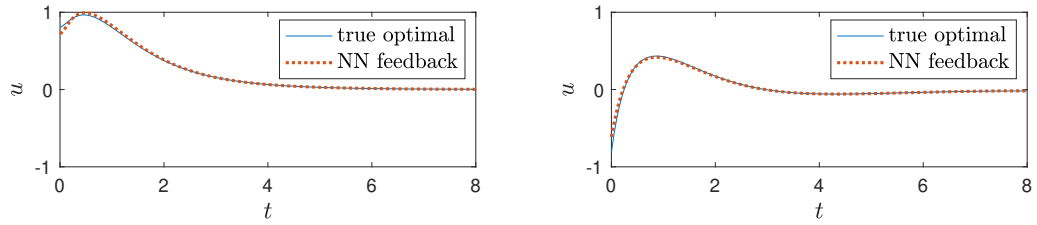
In Figure 3.3, we plot the uncontrolled and NN-controlled dynamics, starting from two different initial conditions, $X(0, \xi) = 2\sin(\pi\xi)$ and $X(0, \xi) = -2\sin(\pi\xi)$, where the dimension of the discretized system is $n = 30$. For both of these initial conditions (and almost all others tested), the NN controller successfully stabilizes



(a) Uncontrolled dynamics.



(b) NN-controlled dynamics.



(c) Comparison of true optimal (open loop BVP solution) and NN control profiles.

Figure 3.3: Simulations of the discretized Burgers'-like PDE (3.19) in $n = 30$ dimensions. Left column: $X(0,\xi) = 2\sin(\pi\xi)$. Right column: $X(0,\xi) = -2\sin(\pi\xi)$.

the open loop unstable origin. Further, as shown in Figure 3.3c, the NN-generated controls are very close to the true optimal controls. Finally, the speed of online control computation is not sensitive to the problem dimension: each evaluation still takes just milliseconds on both an NVIDIA RTX 2080Ti GPU and a 2019 MacBook Air (see Table 5.1).

3.6 Summary

As we showed in Sections 3.4.1 and 3.5.2, generating optimal control data can be surprisingly difficult without the proper tools. This is especially true for industry-strength problems, and because in a realistic control design setting one might frequently change the OCP itself (e.g. cost function weights) to achieve some desired performance criteria. Thus considering the importance and difficulty of generating data, we have devoted this chapter to discussing some practical considerations for this process.

In particular, we have two general choices in terms of open loop OCP solvers: indirect and direct methods. These come with their own tradeoffs: indirect methods are highly accurate and fast, but also difficult to use and highly sensitive; direct methods are easy to use and converge more reliably, but they tend to be less accurate and sometimes slower than indirect methods. Solutions obtained with direct methods can be fed to an indirect solver to increase their accuracy.

Both indirect and direct methods benefit from high-quality initial guesses. We have discussed several strategies for coming up with viable initial guesses. We recommend time marching and LQR warm start to construct small data sets from which a (low-fidelity) NN can be trained. Once an NN is available this can be used for NN warm start, which helps generate additional data faster and more reliably. This data becomes easier to generate as the NN is improved over time, producing better initial guesses. To complement this model refinement process we have proposed a convergence test which also acts to guide us in how much data to generate through. In addition, we give a simple active learning criteria to select where to place new points. This adaptive sampling and model refinement training process allows us to build rich data sets with points anywhere in a semi-global

domain. Thus the NN control is valid for large ranges of system states, rather than just in the neighborhood of some nominal trajectory. Furthermore, data can be generated specifically near complicated regions of the value function or where the required control effort is large.

Finally, through the attitude control and PDE stabilization examples we have illustrated the importance of warm start techniques, application of the proposed adaptive sampling and model refinement algorithm, as well as scalability of the method up to $n = 30$ dimensions.

While we have naturally focused on our own contributions, throughout the chapter we have referenced other promising approaches for warm start and active learning. Almost certainly there also exist others besides these. We make no claim that our approaches are superior, only that they appear viable. An important line of future work is to compare and possibly combine various methods to create a mature data generation toolbox.

Chapter 4

Stability of Neural Network Controlled Systems

Despite promising results shown in Chapters 2 and 3 and other works [124, 55, 6], much less work has been done to study and improve the stability and reliability of NN controllers. To see why this is needed, let us consider another Burgers'-type PDE problem (4.2). If we train a set of NN feedback controllers by the core method from Chapter 2, a surprisingly large fraction of these fail to stabilize the system despite having good test accuracy. Figure 4.1 shows a closed loop simulation with one such controller where the NN-controlled trajectory closely tracks the optimal (stable) trajectory before suddenly destabilizing and eventually settling at an undesired steady state.

This leads us to believe that optimal control approximation accuracy gives an incomplete indication of ultimate closed loop performance. NNs are widely regarded as "black boxes", and even when they have excellent test accuracy it is

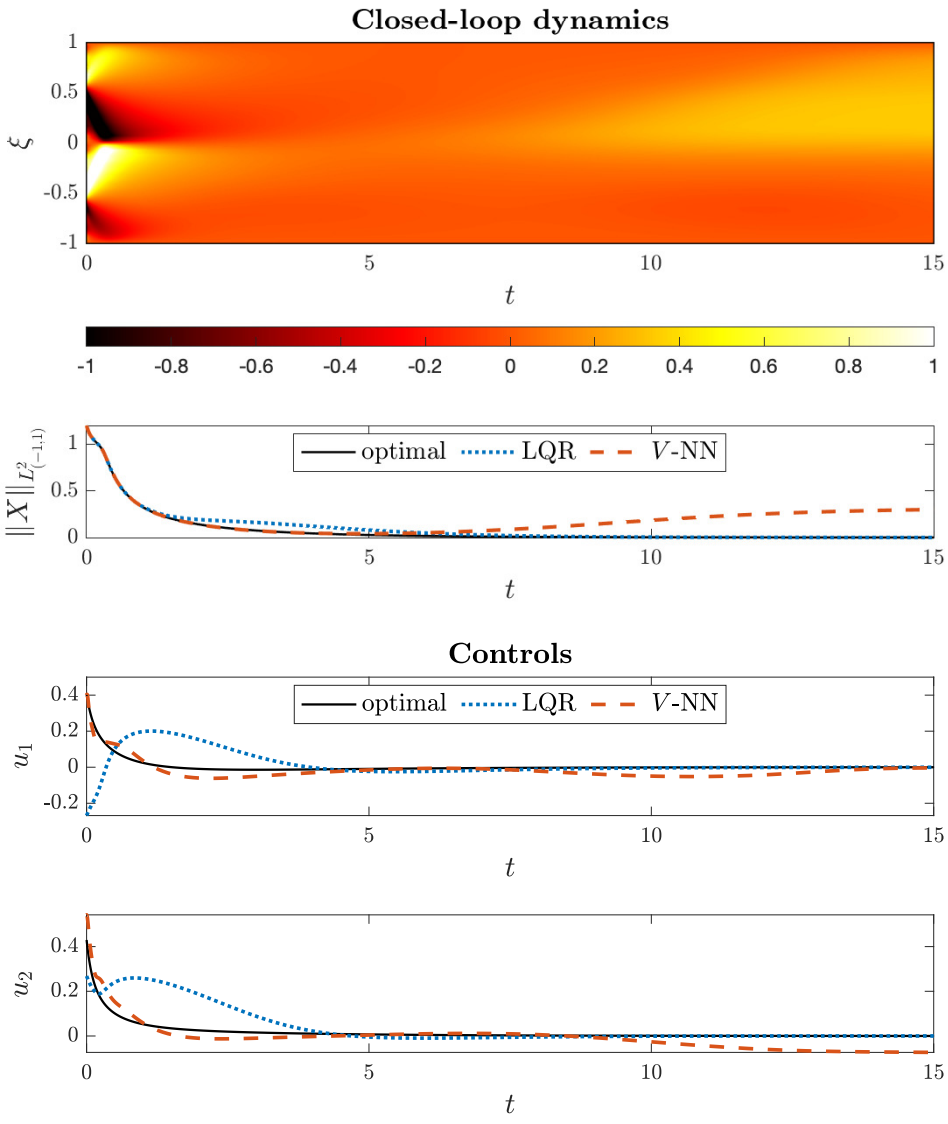


Figure 4.1: Closed loop simulation of the unstable Burgers' PDE (4.6) showing instability. Feedback control is based on a V-NN trained by the methods in Chapter 2. The top plot shows the state $X(t,\xi)$, where ξ is the spatial variable.

difficult to predict how they interact with complicated nonlinear dynamics. As we see in Figure 4.1 a well-trained NN can fail to even stabilize the system. Clearly this is an important problem that cannot be ignored if NN optimal feedback controllers are to be implemented in the real world.

This closed loop stability problem has also been recognized by [150]. Similarly,

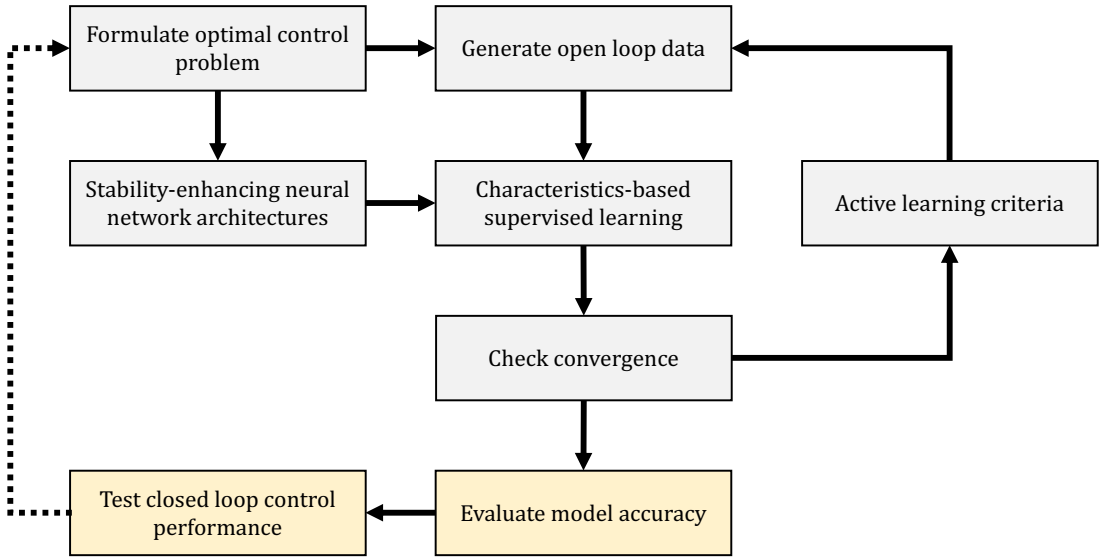


Figure 4.2: Summary of the of the steps and components of the proposed computational framework, highlighting the focus of Chapter 4.

[133] point out that test accuracy incompletely characterizes NN controller performance, suggesting some practical evaluations of optimality and stability. [57] study linear stability near a desired equilibrium, linear time delay stability, and stability around a nominal trajectory using high order Taylor maps.

Chapter 4 describes our contributions in studying this problem drawn from [98, 99, 100]. Figure 4.2 highlight the focus of this chapter within the greater computational framework. We organize the remainder of this chapter as follows. First in Section 4.1 we present a series of fundamental closed loop stability and optimality tests. We apply these tests to a discretized Burgers' PDE problem in $n = 64$ dimensions. Through these tests we show that controlling very high-dimensional systems is possible with the proposed framework, but also that instability in NN-controlled systems happens surprisingly often. Furthermore these results show that standard machine learning test accuracy is not enough to evaluate controller performance. This leads us to Section 4.2, in which we present

some preliminary ideas for theoretically analyzing this stability phenomenon. We obtain a qualitative result corroborating what we see in practice: that test accuracy has a *probabilistic* relationship with semi-global stability. We conclude the chapter with a brief summary discussion in Section 4.3.

Remark 2. *In this chapter in and in Chapter 5 we focus on system stability, which naturally leads us to consider infinite horizon OCPs* (1.4)*. All the examples given from this point forward are infinite horizon problems.*

4.1 Numerical stability and optimality analysis

We begin with numerical experiments to explore the relationship between test accuracy and closed loop stability and optimality. We present results for three different tests:

1. linear stability near $\mathbf{x}_f$ (Section 4.1.3);
2. MC nonlinear stability (Section 4.1.4);
3. MC optimality analysis (Section 4.1.5).

Such tests are of course familiar to the control community, but we believe it is worth emphasizing their importance since more rigorous and realistic testing is needed in order to start trusting NN controllers in real-world applications. We also note that these tests are just a starting point: further examples include stabilization time [133], time delay stability [57], and robustness to measurement noise, disturbances, and parameter variations.

As a testbed we consider a 64-dimensional infinite horizon OCP arising from Chebyshev PS discretization of a modified Burgers' equation with a destabilizing

reaction term. This problem is slightly different from the one studied in Section 3.5. The numerical results illustrate that the proposed methodology can synthesize optimal feedback controllers for very high-dimensional systems. On the other hand, these standard NNs are *not* consistently stable even when they have good approximation accuracy. To improve the reliability of the method we will need a specialized *QRnet* controller from Chapter 5.

4.1.1 Infinite horizon unstable Burgers' control

Let $X(t,\xi) : [0,\infty) \times [-1,1] \to \mathbb{R}$ satisfy the following one-dimensional controlled PDE with Dirichlet boundary conditions:

$$\begin{cases} X_t = -\frac{1}{2}(X^2)_\xi + \nu X_{\xi\xi} + \alpha(\xi) X e^{-\beta X} + \mathbf{b}^T(\xi)\mathbf{u}(t), & t > 0,\ \xi \in (-1,1), \\ X(t,-1) = X(t,1), & t > 0, \\ X(0,\xi) = X_0(\xi), & \xi \in (-1,1). \end{cases} \tag{4.1}$$

Here $\nu, \beta > 0$ are scalar parameters, $\alpha : (-1,1) \to \mathbb{R}$, and $\mathbf{b} : (-1,1) \to \mathbb{R}^m$. The control $\mathbf{u} : [0,\infty) \to \mathbb{R}^m$ is designed to stabilize the *open-loop unstable* origin by solving the PDE-constrained OCP

$$\begin{cases} \underset{\mathbf{u}(\cdot)}{\text{minimize}} & J\left[\mathbf{u}(\cdot)\right] = \displaystyle\int_0^\infty \left(\|X\|^2_{L^2_{(-1,1)}} + R\,\mathbf{u}^T\mathbf{u} \right) dt, \\ \text{subject to} & X_t = -\frac{1}{2}(X^2)_\xi + \nu X_{\xi\xi} + \alpha(\xi) X e^{-\beta X} + \mathbf{b}^T(\xi)\mathbf{u}(t), \\ & X(t,-1) = X(t,1). \end{cases} \tag{4.2}$$

The destabilizing reaction term is similar to the one included in the finite horizon Burgers'-like problem (3.21), which was adapted from an OCP studied in [62]. In

the present problem we define

$$\alpha(\xi) = \begin{cases} -\kappa\left(\xi + \frac{1}{5}\right)\left(\xi - \frac{1}{5}\right), & \xi \in \left[-\frac{1}{5}, \frac{1}{5}\right], \\ 0, & \xi \notin \left[-\frac{1}{5}, \frac{1}{5}\right]. \end{cases} \tag{4.3}$$

and consider the case with $m = 2$ actuators active on compact supports defined by

$$\mathbf{b}(\xi) = \begin{pmatrix} \begin{cases} -\kappa\left(\xi + \frac{4}{5}\right)\left(\xi + \frac{2}{5}\right), & \xi \in \left[-\frac{4}{5}, -\frac{2}{5}\right], \\ 0, & \xi \notin \left[-\frac{4}{5}, -\frac{2}{5}\right], \end{cases} \\ \begin{cases} -\kappa\left(\xi - \frac{2}{5}\right)\left(\xi - \frac{4}{5}\right), & \xi \in \left[\frac{2}{5}, \frac{4}{5}\right], \\ 0, & \xi \notin \left[\frac{2}{5}, \frac{4}{5}\right]. \end{cases} \end{pmatrix} \tag{4.4}$$

We set $\nu = 0.02$, $\beta = 0.1$, $\kappa = 25$, and $R = 0.5$. We consider initial conditions which are sums of sine functions with uniform random coefficients:

$$X_0(\xi) = \sum_{k=1}^{10} a_k \sin\left(k\pi\xi\right), \quad a_k \sim \mathcal{U}\left(-1/k, 1/k\right), \tag{4.5}$$

To solve (4.2) using our framework, we perform Chebyshev PS collocation as in Section 3.5.1 to get a quadratic cost ODE-constrained problem,

$$\begin{cases} \underset{\mathbf{u}(\cdot)}{\text{minimize}} & \mathcal{J}_n\left[\mathbf{u}(\cdot)\right] = \displaystyle\int_0^\infty \left(\|\mathbf{x}\|_{\mathbf{Q}}^2 + R\,\mathbf{u}^T\mathbf{u}\right) dt, \\ \text{subject to} & \dot{\mathbf{x}} = -\dfrac{1}{2}\mathbf{D}\left(\mathbf{x} \odot \mathbf{x}\right) + \nu\mathbf{D}^2\mathbf{x} + \boldsymbol{\alpha} \odot \mathbf{x} \odot e^{-\beta\mathbf{x}} + \mathbf{B}\mathbf{u}. \end{cases} \tag{4.6}$$

Here $\boldsymbol{\alpha} \in \mathbb{R}^n$ and $\mathbf{B} \in \mathbb{R}^{n\times m}$ are the collocated versions of $\alpha(\xi)$ and $\mathbf{b}^T(\xi)$; and recall from Section 3.5.1 that $\mathbf{D}$ is the Chebyshev differentiation matrix, $\|\mathbf{x}\|_{\mathbf{Q}} = \sqrt{\mathbf{x}^T\mathbf{Q}\mathbf{x}}$, and $\mathbf{Q}$ is a diagonal matrix of Clenshaw-Curtis quadrature weights. We choose $n = 64$ collocation points to accurately approximate the PDE dynamics. Of course this becomes a very large OCP. Nevertheless, we find

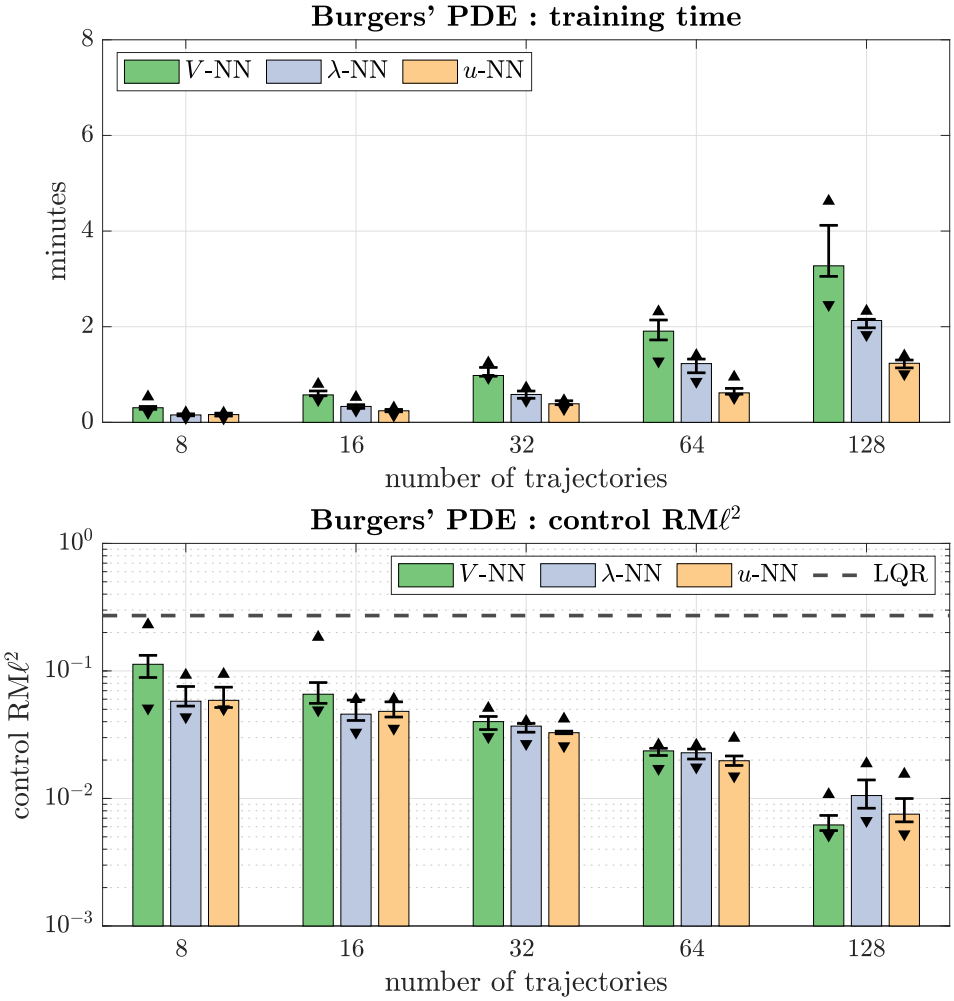


Figure 4.3: Training time and control prediction accuracy for the infinite horizon Burgers' OCP (4.6), depending on the amount of training data. Bar heights show the medians over ten trials, error bars show the 25th and 75th percentiles, and triangles are minimum and maximum values.

that our approach can handle the dimensionality – especially when we revisit this problem with *QRnet* architectures in Section 5.2.

4.1.2 Learning results

To generate the infinite horizon data we use an indirect method [72] with LQR warm start. We generate training data sets with different numbers of trajecto-

ries: $N_{\text{OCP, train}} = 8, 16, 32, 64$, and 128. Since generalization accuracy naturally improves as we provide more data, this yields NN controllers of varying quality. Because data generation depends on random sampling and NN training is a highly non-convex optimization problem, results can vary for different random seeds. To account for this, for each different data set size we conduct ten trials with different randomly generated training trajectories and NN weight initializations. We evaluate test metrics on an independent test data set with $N_{\text{test}} = 83327$ gathered along $N_{\text{OCP, test}} = 500$ trajectories.

We train V-NN, λ-NN, and u-NN models, all with $L = 5$ hidden layers with $w = 32$ neurons each and $\tanh(\cdot)$ activation functions. For V-NN and λ-NN we found that the value gradient loss term (2.23) did not improve results, so we set $\mu_V = 1/5$, $\mu_{\boldsymbol{\lambda}} = 0$, and $\mu_{\mathbf{u}} = 1$. Numerical optimization is carried out with L-BFGS [85] and all models are trained on an NVIDIA RTX 2080Ti GPU.

Figure 4.3 shows training time and control approximation accuracy statistics for each group of NNs. For this very high-dimensional problem we notice that V-NN takes the longest to train due to the need to compute gradients $\widehat{V}_{\mathbf{x}}(\cdot)$ at each training step. Each model type performed about equally well, on average, and all have better accuracy than LQR. As expected, generalization accuracy also improves with data set size. While these results appear promisng so far, we will see in the following sections that NNs with similar approximation accuracy can induce quite different closed loop behavior.

4.1.3 Local stability analysis

As a first step we assess the local stability of each NN-controlled system. Local stabilization is a bare minimum requirement of any feedback controller. Let $\bar{\mathbf{x}} \in$

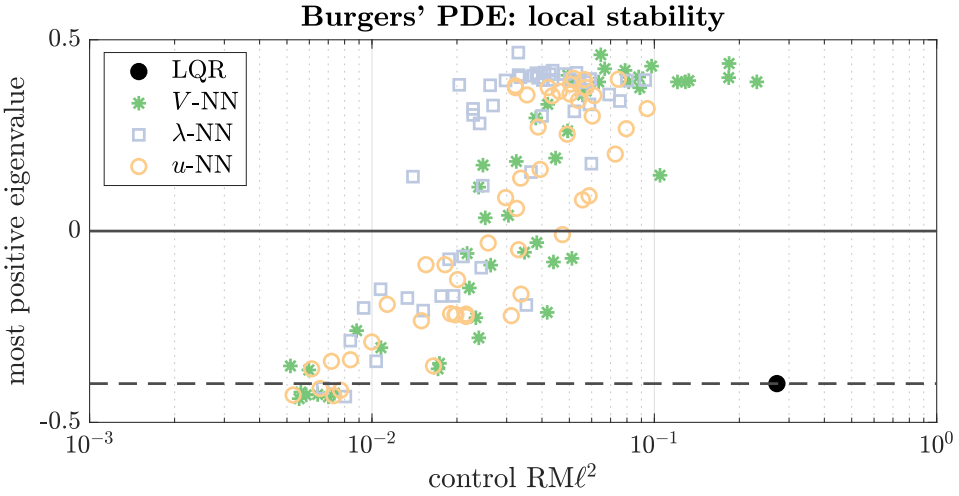


Figure 4.4: Real part of most positive closed loop Jacobian eigenvalue at an equilibrium $\bar{\mathbf{x}}$ near $\mathbf{x}_f$. Each marker represents a single model.

$\mathbb{R}^n$ be an equilibrium[1] of the closed loop system. The dynamics near $\bar{\mathbf{x}}$ can be approximated by $\dot{\mathbf{x}} \approx \mathbf{A}_{\text{cl}}(\mathbf{x} - \bar{\mathbf{x}})$, where

$$\mathbf{A}_{\text{cl}} \coloneqq \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\bar{\mathbf{x}},\widehat{\mathbf{u}}(\bar{\mathbf{x}})} + \left.\frac{\partial \mathbf{f}}{\partial \mathbf{u}}\right|_{\bar{\mathbf{x}},\widehat{\mathbf{u}}(\bar{\mathbf{x}})} \left.\frac{\partial \widehat{\mathbf{u}}}{\partial \mathbf{x}}\right|_{\bar{\mathbf{x}}} \tag{4.7}$$

is the closed loop Jacobian. Thus, after synthesizing a feedback controller we can easily check for local stability by seeing if $\mathbf{A}_{\text{cl}}$ is Hurwitz [71]. Furthermore as noted in [57], one benefit of using an NN controller with differentiable activation functions is that the closed loop dynamics are locally $\mathcal{C}^1$. This allows one to use tools from linear systems theory to characterize the local stability of $\bar{\mathbf{x}}$.

Figure 4.4 shows the real part of the most positive eigenvalue of $\mathbf{A}_{\text{cl}}$ for each NN. Overall we see a clear correlation between local stability and approximation accuracy. We should expect a trend like this because the optimal control – which we are trying to approximate – is stabilizing. Still, these standard NNs must be

[1] In general $\bar{\mathbf{x}} \neq \mathbf{x}_f$. An equilibrium $\bar{\mathbf{x}}$ near $\mathbf{x}_f$, if it exists, can be obtained with a root-finding algorithm.

trained to a high level of test accuracy before they are even LAS, which necessitates a large data set and long training time. Furthermore, until a high accuracy level is reached there is significant variability in local stability. In other words, unless we train the NN on a lot of data we have no confidence that it will meet a bare minimum LAS requirement.

When we revisit this problem in Section 5.2 we will find that the new *QRnet* architectures will guarantee LAS while still achieving the same approximation accuracy. These architectures also ensure that the equilibrium point $\bar{\mathbf{x}} = \mathbf{x}_f$ exactly.

4.1.4 Monte Carlo nonlinear stability analysis

Here and in Section 4.1.5 we empirically assess *nonlinear* semi-global stability and optimality by conducting MC closed loop simulations. Although MC tests do not guarantee semi-global stability with 100% certainty, they give us a probabilistic metric by which to evaluate controller performance. Nonlinear stability analysis for general, nonlinear, high-dimensional systems remains an open research problem, since analytical techniques base on Lyapunov analysis [71] are not applicable or are too conservative. Despite some recent progress in this area [117, 91, 47, 65, 151], MC simulation remains the most practical and straightforward approach.

For the Burgers' PDE problem we randomly select initial conditions $X_0^{(i)}$, $i = 1, \ldots, N_{\text{MC}} = 100$ according to (4.5). The collocated initial conditions are written as $\mathbf{x}_0^{(i)}$ and we rescale them such that

$$\left\|\mathbf{x}_0^{(i)}\right\|_{\mathbf{Q}} = 1.2 \approx \max_{\mathbf{x}^{(j)} \in \mathcal{D}_{\text{train}}} \left\|\mathbf{x}^{(j)}\right\|_{\mathbf{Q}}. \tag{4.8}$$

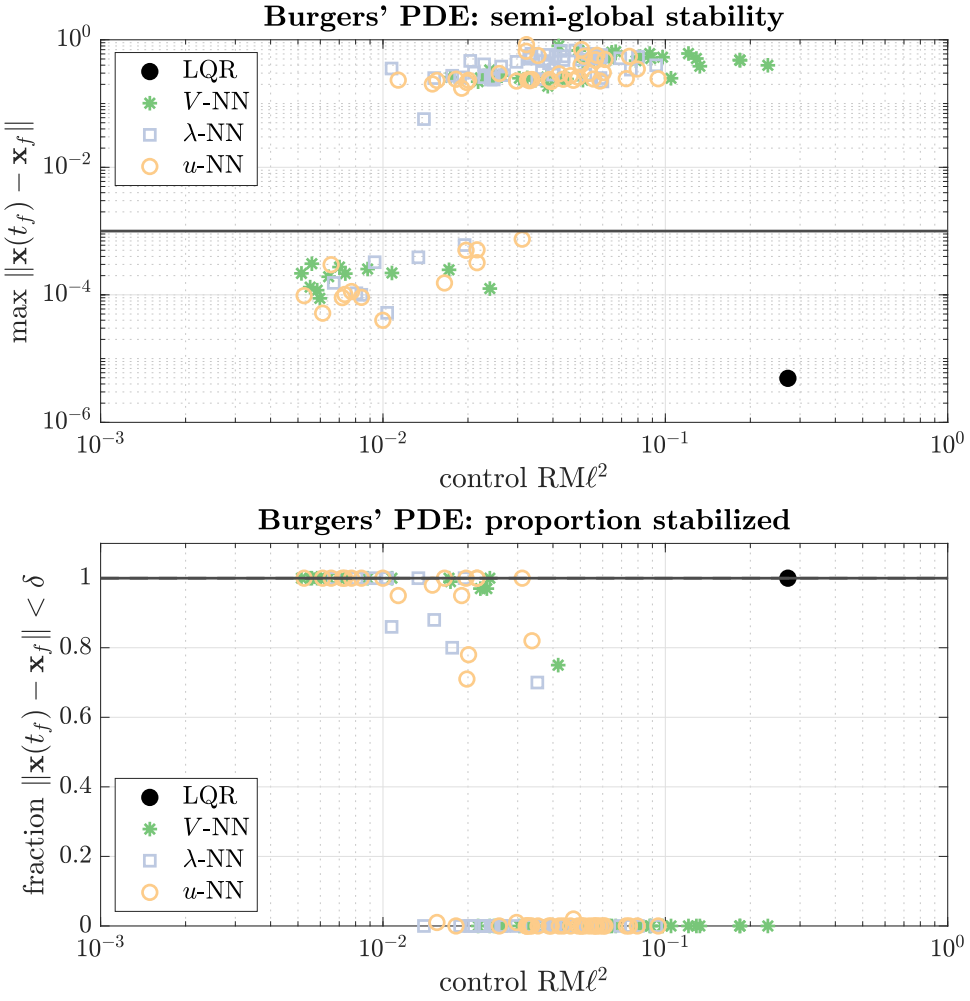


Figure 4.5: Worst-case norm of final state (top) and fraction of stabilized initial conditions (bottom) of $N_{\text{MC}} = 100$ simulations. Each marker represents a single model.

This places the MC initial conditions at the edge of the training domain where the NNs may be less accurate and the system harder to control.

For each initial condition we integrate the closed loop dynamics until the system reaches a steady state or we exceed a large final time. Given a specified tolerance $\delta > 0$ we say that a trajectory has been stabilized if $\left\|\mathbf{x}\left(t_f; \mathbf{x}_0^{(i)}\right) - \mathbf{x}_f\right\| < \delta$. If N_{MC} is large enough and the *worst-case failure*, $\max_{\mathbf{x}_0^{(i)}} \left\|\mathbf{x}\left(t_f; \mathbf{x}_0^{(i)}\right) - \mathbf{x}_f\right\|$, is

smaller than δ, then the closed loop system is likely to be semi-globally stable[2] Alternatively we could require that a controller stabilize a certain percentage of trajectories. In this dissertation we focus on the worst-case failure condition, $\max_{\mathbf{x}_0^{(i)}} \left\| \mathbf{x}\left(t_f; \mathbf{x}_0^{(i)}\right) - \mathbf{x}_f \right\| < \delta$.

For this problem we take $\delta = 10^{-3}$. Figure 4.5 shows the worst-case failures and fraction of successfully stabilized trajectories for each controller. We immediately see that only the most accurate NNs consistently stabilize a point close to the origin. Also notice that a number of NNs which are LAS as per Figure 4.4 fail to stabilize the *nonlinear* system.

4.1.5 Monte Carlo optimality analysis

In this work we are interested in both stability *and* optimality. Optimality of a given controller $\widehat{\mathbf{u}}(\cdot)$ can be quantified by the accumulated cost $\mathcal{J}\left[\widehat{\mathbf{u}}(\cdot); \mathbf{x}_0^{(i)}\right]$ compared to the optimal cost $V\left(\mathbf{x}_0^{(i)}\right)$, across all MC simulations $i = 1, \ldots, N_{\text{MC}}$. Concretely we evaluate

$$\%\text{suboptimality}\left(\widehat{\mathbf{u}}\left(\cdot\right); \mathbf{x}_0^{(i)}\right) = 100 \frac{\mathcal{J}\left[\widehat{\mathbf{u}}\left(\cdot\right); \mathbf{x}_0^{(i)}\right] - V\left(\mathbf{x}_0^{(i)}\right)}{V\left(\mathbf{x}_0^{(i)}\right)}. \tag{4.9}$$

Figure 4.6 shows the results of this analysis for the same set of MC simulations conducted in Section 4.1.4. Among the stabilizing NN controllers we see a correlation between higher test accuracy and better performance. All the stabilizing NN controllers follow this trend and perform better than LQR. This motivates us to pursue optimal control, which if successfully implemented also stabilizes the nonlinear system.

[2] for standard NN architectures we will always have at least a small perturbation from $\mathbf{x}_f$. Thus for these architectures $\mathbf{x}_f$ cannot be semi-globally stable, only ultimately bounded.

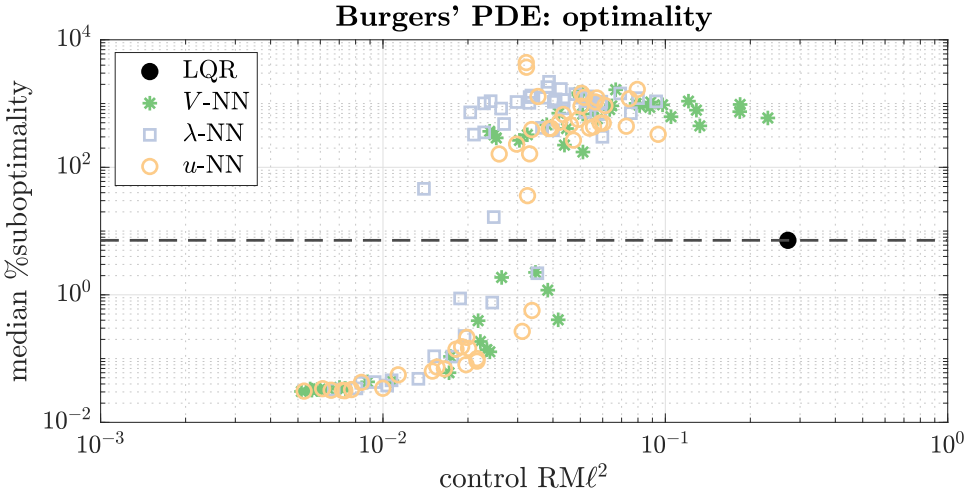


Figure 4.6: Median percent cost more than optimal cost over $N_{\text{MC}} = 100$ simulations. Each marker represents a single model.

4.1.6 Discussion

Let us conclude with another closed loop simulation of the initial condition shown in Figure 4.1, taking another V-NN trained on a different training data set also with $N_{\text{OCP, train}} = 64$ trajectories. Observing that the two NNs have similar control RMℓ^2 (0.0239 and 0.0173 for the unstable and stable cases, respectively), we should expect that they perform similarly well but this turns out not to be the case. The former is of course unstable, while the latter stabilizes the initial condition as shown in Figure 4.7. Furthermore, it is only 0.75% suboptimal for this initial condition. Compared to LQR which is 21.70% suboptimal, the second NN very closely approximates the optimal control. This stark performance difference is consistent among the MC trajectories but is not apparent from the test accuracy statistics. Thus while test accuracy does correlate with stability and performance, we should not rely on this as the primary metric to evaluate our models.

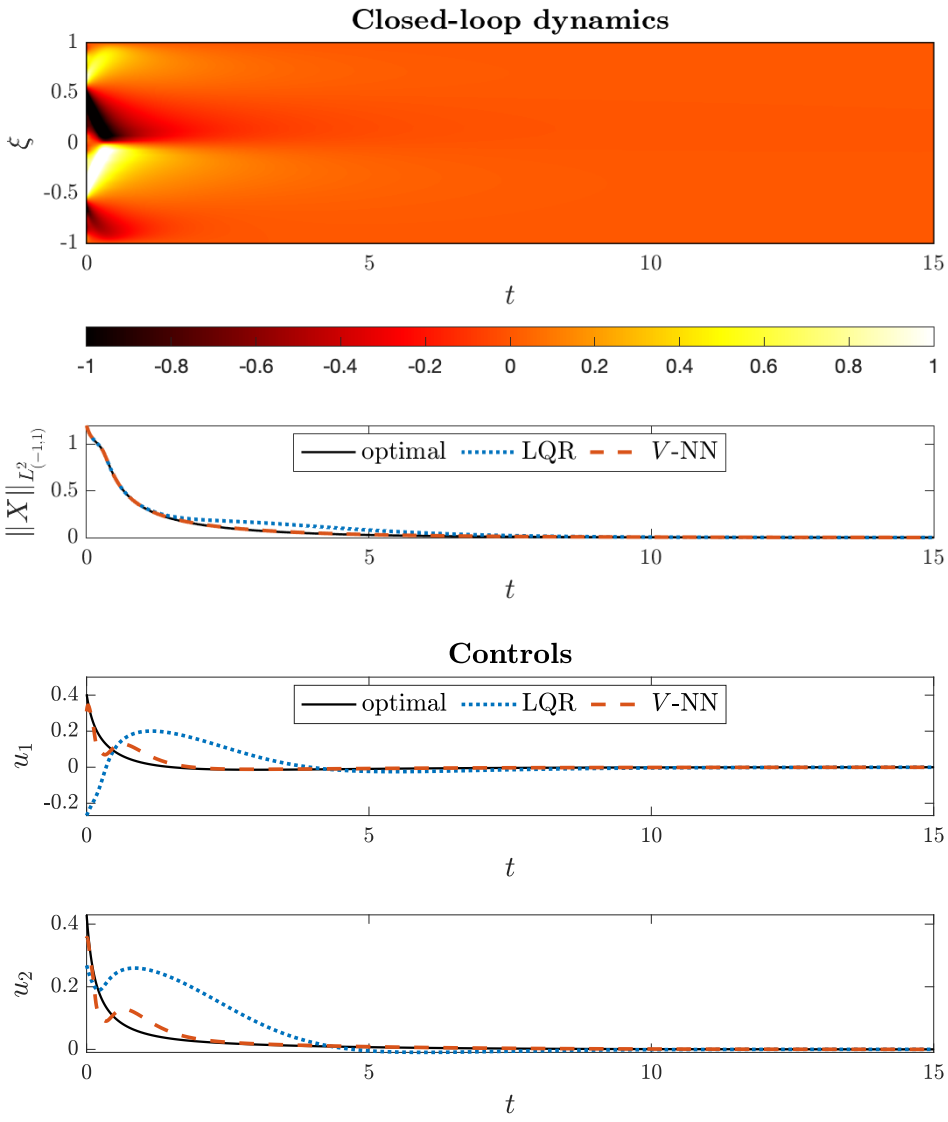


Figure 4.7: Closed loop simulation of the unstable Burgers' PDE (4.6). Feedback control is based on a V-NN trained by the methods in Chapter 2 on the same number of data as the controller shown in Figure 4.1.

4.2 Probabilistic stability analysis

In Section 4.1 we showed that NNs with high approximation accuracy could sometimes be unstable. This motivates us to try to futher understand stability properties of (deterministic) closed loop systems with control policies $\widehat{\mathbf{u}}(\cdot)$ that are

trained and tested on randomly-generated data. It is important to emphasize that $\widehat{\mathbf{u}}(\cdot)$, once trained, is completely deterministic, but its final performance is influenced by and measured by random sampling. This gives rise to an interesting situation where although the closed loop dynamics are deterministic, we can only speak about stability in a probabilistic sense. Our goal is to provide intuitive yet novel qualitative insights into the behavior of these systems.

4.2.1 Deterministic stability

We start with a standard stability result for perturbations to exponentially stable systems. Without loss of generality let the objective equilibrium state be $(\mathbf{x}_f = \mathbf{0}, \mathbf{u}_f = \mathbf{0})$. This simplifies notation and easily extends to arbitrary equilibrium states. Next let $r > 0$ and denote the open ball

$$\mathbb{B}_r := \{\mathbf{x} \in \mathbb{R}^n | \, \|\mathbf{x}\|_2 < r\} \subseteq \mathbb{R}^n. \tag{4.10}$$

We will make the following assumptions about the optimally-controlled system, $\dot{\mathbf{x}} = \mathbf{f}_{\text{cl}}^*(\mathbf{x}) := \mathbf{f}\left(\mathbf{x}, \mathbf{u}^*(\mathbf{x})\right)$:

Assumption 4. *The goal state $\mathbf{x}_f$ is an exponentially stable equilibrium of the optimally-controlled system.*

Assumption 5. *The Jacobian matrix $\frac{\partial \mathbf{f}_{cl}^*}{\partial \mathbf{x}}$ is Lipschitz throughout the closure $\overline{\mathbb{B}}_r$.*

We remark that these assumptions are not too restrictive in the present context. In particular, when we design the cost function (1.25) we generally choose this to satisfy Assumption 4 as exponential stability provides important robustness properties. Assumption 5 holds under Assumption 1 that the OCP admits a stabilizing solution, the standard assumption that the dynamics are locally $\mathcal{C}^1$

in $\mathbf{x}$ and $\mathbf{u}$, and when the optimal control $\mathbf{u}^*(\cdot)$ is $\mathcal{C}^1$ in $\mathbf{x}$. This last condition precludes OCPs with controls that saturate for some states in $\overline{\mathbb{B}}_r$, unless we shrink r. We expect that this condition is not strictly necessary and can be relaxed with an appropriate choice of Lyapunov function in Lemma 1.

The following lemma characterizes the stability of the closed loop dynamics if we replace the optimal feedback control $\mathbf{u}^*(\cdot)$ with some $\widehat{\mathbf{u}}(\cdot)$. Since there is no guarantee that the origin remains an equilibrium under the new control law, stability is stated in terms of *ultimate boundededness*, the property that trajectories starting close to the origin stay close to the origin [71].

Lemma 1 (Ultimate boundedness of perturbed systems)**.** *Suppose that Assumptions 4 and 5 are satisfied. Let $\widehat{\mathbf{u}} : \mathbb{R}^n \to \mathbb{U}$ be a continuous feedback control policy and assume that its true maximum control approximation error,*

$$\delta \coloneqq \max_{\mathbf{x} \in \overline{\mathbb{B}}_r} \left\| \widehat{\mathbf{u}}(\mathbf{x}) - \mathbf{u}^*(\mathbf{x}) \right\|_2 , \tag{4.11}$$

is sufficiently small. Then trajectories of the closed loop system starting sufficiently close to the origin are ultimately bounded with ultimate bound proportional to δ.

Proof. Assumptions 4 and 5 satisfy the conditions of [71, Theorem 4.14] which establishes the existence of a $\mathcal{C}^1$ Lyapunov function $W : \mathbb{B}_r \to [0, \infty)$ for the optimally-controlled system $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}^*(\mathbf{x}))$. This $W(\cdot)$ is of quadratic type,

meaning that there exist constants $k_1, k_2, k_3, k_4 > 0$ such that[3]

$$\begin{cases} k_1\|\mathbf{x}\|_2^2 \leq W(\mathbf{x}) \leq k_2\|\mathbf{x}\|_2^2, \\ \left[\frac{\partial W}{\partial \mathbf{x}}(\mathbf{x})\right] \mathbf{f}\left(\mathbf{x}, \mathbf{u}^*(\mathbf{x})\right) \leq -k_3\|\mathbf{x}\|_2^2, \\ \left\|\frac{\partial W}{\partial \mathbf{x}}(\mathbf{x})\right\|_2 \leq k_4\|\mathbf{x}\|_2. \end{cases} \tag{4.12}$$

Now we rewrite the closed loop dynamics as a perturbation of the optimally controlled system, $\dot{\mathbf{x}} = \mathbf{f}\left(\mathbf{x}, \mathbf{u}^*(\mathbf{x})\right)$:

$$\dot{\mathbf{x}} = \mathbf{f}\left(\mathbf{x}, \widehat{\mathbf{u}}(\mathbf{x})\right) = \mathbf{f}\left(\mathbf{x}, \mathbf{u}^*(\mathbf{x})\right) + \left[\mathbf{f}\left(\mathbf{x}, \widehat{\mathbf{u}}(\mathbf{x})\right) - \mathbf{f}\left(\mathbf{x}, \mathbf{u}^*(\mathbf{x})\right)\right]. \tag{4.13}$$

Then for all $\mathbf{x} \in \overline{\mathbb{B}}_r$ the perturbation satisfies

$$\left\|\mathbf{f}\left(\mathbf{x}, \widehat{\mathbf{u}}(\mathbf{x})\right) - \mathbf{f}\left(\mathbf{x}, \mathbf{u}^*(\mathbf{x})\right)\right\|_2 \leq L_{\mathbf{u}}\delta, \tag{4.14}$$

where $L_{\mathbf{u}}$ is the Lipschitz constant of $\mathbf{f}(\cdot)$ with respect to $\mathbf{u}$. Hence if there exists $\theta \in (0, 1)$ such that

$$\delta < \delta^+ := \frac{\theta k_3}{L_{\mathbf{u}} k_4}\sqrt{\frac{k_1}{k_2}}r, \tag{4.15}$$

then [71, Lemma 9.2] guarantees that for all trajectories of the perturbed system (4.13) with $\|\mathbf{x}_0\|_2 < r\sqrt{k_1/k_2}$, there exists some finite time $T = T\left(\mathbf{x}_0\right)$ such that

$$\begin{cases} \|\mathbf{x}(t; \mathbf{x}_0)\|_2 \leq K\|\mathbf{x}_0\|_2 \exp\left(-\alpha t\right), & 0 \leq t < T, \\ \|\mathbf{x}(t; \mathbf{x}_0)\|_2 \leq B, & T \leq t, \end{cases} \tag{4.16}$$

[3] If $\mathbf{Q}$ is positive definite then we can take the quadratic Lyapunov function for the optimal system to be $W(\mathbf{x}) = V^{\mathrm{LQR}}(\mathbf{x})$, and the constants $k_1, \ldots, k_4$ can be computed based on the eigenvalues of $\mathbf{P}$ and $\mathbf{Q}$.

with

$$K = \sqrt{\frac{k_2}{k_1}}, \quad \alpha = \frac{(1-\theta)k_3}{2k_2}, \quad B = \frac{L_{\mathbf{u}}k_4}{\theta k_3}\sqrt{\frac{k_2}{k_1}}\delta. \tag{4.17}$$

□

4.2.2 Maximum error estimation

Since we do not have access to the true optimal control $\mathbf{u}^*(\cdot)$, we cannot compute the true maximum error δ needed to apply Lemma 1. Hence in practice we need to estimate this using test data. Let $\delta_N = \max \ell^2$ be the maximum error (2.35) for a set of test points $\mathbf{x}^{(i)} \in \overline{\mathbb{B}}_r$, $i = 1, \ldots, N$. Since we only have δ_N and not δ, this leads us to two important questions:

1. How can we accurately approximate $\delta_N \approx \delta$ using a reasonably-sized test data set?

2. With what level of confidence can we rely on δ_N to characterize the stability of the closed loop system?

The simplest approach to estimating δ_N is with independent uniform samples. But as we will see in Lemma 2 and (4.22), this is not sample-efficient in high dimensions. Since generating each test point requires solving an OCP (1.4), it can become prohibitively costly to generate sufficient data for testing in this way. Global optimization may offer a more tractable approach, and many methods have been developed for this purpose. Since there are too many algorithms to review here, we refer the reader to e.g. [86] for a summary, and note that any method chosen for this application should work without gradient information and not require too many function evaluations. Clearly this is a difficult problem, though one upside is that each OCP which we solve can generate an entire trajectory of

test points which can subsequently be used for training new models.

To work towards an answer to the second question, we start with Lemma 2 which considers the problem of estimating the maximum value of a continuous function on a compact domain using independent samples. It is easy to see how this specializes to computing maximum test errors, and in Section 4.2.3 we apply this result to stability analysis. Estimates of the gap between δ_N and δ for correlated test points[4] are beyond the scope of this work. For this lemma we require the following definition.

Definition 1 (Convergence in probability)**.** *A sequence $\{Z_k\}$ of random variables converges in probability to the random variable Z if for all $\epsilon > 0$,*

$$\lim_{k\to\infty} \Pr\left(|Z_k - Z| > \epsilon\right) = 0. \tag{4.18}$$

More explicitly, for any $\epsilon, P > 0$ there exists $K \geq 0$ such that for all $k \geq K$ we have $\Pr\left(|Z_k - Z| > \epsilon\right) < P$.

Lemma 2 (Maximum estimation)**.** *Let $\mathbb{D} \subset \mathbb{R}^n$ be a compact set with non-zero volume and let $g : \mathbb{D} \to [0, \infty)$ be a continuous, non-negative function with maximum value $\delta \coloneqq \max_{\mathbf{x}\in\mathbb{D}} g(\mathbf{x})$. Suppose that $\mathbf{x}^{(i)}, i = 1, \ldots, N$, are independently sampled according to a probability distribution $\mu(\mathbf{x})$ supported everywhere on $\mathbb{D}$, and let $\delta_N \coloneqq \max_{i\in\{1,\ldots,N\}} g\left(\mathbf{x}^{(i)}\right)$. Then for any $\epsilon > 0$, we have*

$$\Pr\left(\delta > \delta_N + \epsilon\right) = \left[\Pr\left(\delta - g(\mathbf{x}) > \epsilon\right)\right]^N \tag{4.19}$$

with $\Pr\left(\delta - g(\mathbf{x}) > \epsilon\right) < 1$, and thus $\{\delta_N\}_{N=1}^{\infty} \to \delta$ in probability.

[4]Such correlations can be induced because entire trajectories are included in the data and by the global optimization algorithm.

Proof. For any $N, \epsilon > 0$ we compute

$$\begin{aligned}\Pr\left(\delta > \delta_N + \epsilon\right) &= \Pr\left(\delta - \max_{i\in\{1,\ldots,N\}} g\left(\mathbf{x}^{(i)}\right) > \epsilon\right) \\ &= \Pr\left(\delta - g\left(\mathbf{x}^{(i)}\right) > \epsilon, i = 1, \ldots, N\right) \\ &= \left[\Pr\left(\delta - g(\mathbf{x}) > \epsilon\right)\right]^N,\end{aligned}$$

where we have used the fact that $\mathbf{x}^{(i)}$ are independent. Let

$$F(\delta, \epsilon) \coloneqq \Pr\left(\delta - g(\mathbf{x}) > \epsilon\right). \tag{4.20}$$

We claim that $F(\delta, \epsilon) < 1$ for all $\epsilon > 0$. To see this, first since $g(\mathbf{x}) \geq 0$ we immediately have $F(\delta, \epsilon) = 0$ for all $\epsilon \geq \delta$. Next if $0 < \epsilon < \delta$ let $\mathbf{z} \in \mathbb{D}$ be any point which achieves the true maximum, i.e. $g(\mathbf{z}) = \delta$. By continuity of $g(\cdot)$, we know that there must exist some $d > 0$ such that for all $\mathbf{x}$ in the neighborhood $\|\mathbf{z} - \mathbf{x}\|_2 < d$, we have

$$|g(\mathbf{z}) - g(\mathbf{x})| = |\delta - g(\mathbf{x})| = \delta - g(\mathbf{x}) \leq \epsilon.$$

We also know that for any $d > 0$, the intersection

$$\mathbb{D} \cap \{\mathbf{x} \in \mathbb{R}^n \,|\, \|\mathbf{z} - \mathbf{x}\|_2 < d\} = \{\mathbf{x} \in \mathbb{D} \,|\, \|\mathbf{z} - \mathbf{x}\|_2 < d\}$$

is non-empty and open. Then because $\mu(\mathbf{x})$ is supported everywhere on $\mathbb{D}$, it follows that the set $\{\mathbf{x} \in \mathbb{D} \,|\, \delta - g(\mathbf{x}) \leq \epsilon\}$ must have non-zero probability mass and hence $F(\delta, \epsilon) < 1$. Therefore

$$\lim_{N\to\infty} \Pr\left(|\delta - \delta_N| > \epsilon\right) = \lim_{N\to\infty} \Pr\left(\delta > \delta_N + \epsilon\right) = \lim_{N\to\infty} \left[F(\delta, \epsilon)\right]^N = 0. \tag{4.21}$$

Furthermore, $\delta \geq \delta_N$ implies $|\delta - \delta_N| = \delta - \delta_N$ and so the sequence $\{\delta_N\}_{N=1}^{\infty}$ converges in probability to δ. □

Eq. (4.21) appears promising since $[F(\delta, \epsilon)]^N$ decrease exponentially in N, but it is worth pointing out that $F(\delta, \epsilon)$ depends on the volume of the sample domain, how smooth $g(\cdot)$ is, and of course the method for sampling $\mathbf{x}^{(i)}$. For example, suppose that $\mathbf{x}^{(i)}$ are sampled uniformly from $\mathbb{D}$ so that

$$F(\delta, \epsilon) = \frac{\int_{\{\mathbf{x} \in \mathbb{D} | \delta - g(\mathbf{x}) > \epsilon\}} d\mathbf{x}}{\int_{\mathbb{D}} d\mathbf{x}}. \tag{4.22}$$

By inspection we can see that $F(\delta, \epsilon)$ is smaller if $g(\mathbf{x})$ is flatter, and conversely $F(\delta, \epsilon) \to 1$ as the domain grows (which happens if we increase the dimension). While $[F(\delta, \epsilon)]^N$ does decreases exponentially in N once we fix ϵ and the test domain $\mathbb{D}$, because of the strong dependence on the problem dimension in practice we should prefer optimization-based strategies over random sampling.

4.2.3 Probabilistic stability based on test accuracy

Now we are ready to apply Lemma 2 to find the probability that the our error estimate δ_N is close enough to δ such that the feedback controller $\widehat{\mathbf{u}}(\cdot)$ meets the requirements for Lemma 1, and hence sufficient conditions for ultimate boundedness.

Proposition 1 (Probability of ultimate boundedness)**.** *Suppose that Assumptions 4 and 5 are satisfied and let $\widehat{\mathbf{u}} : \mathbb{R}^n \to \mathbb{U}$ be a continuous feedback control. Consider test points $\mathbf{x}^{(i)}, i = 1, \ldots, N$, independently sampled according to some probability distribution supported everywhere on $\overline{\mathbb{B}}_r$. Let δ bet the true unknown control error (4.11), δ_N be the error estimate (2.35), and δ^+ be the conservative*

allowable error (4.15). *If* $\delta_N < \delta^+$, *then the probability that the error estimate is accurate enough to determine if the closed loop system satisfies sufficient conditions for ultimate boundedness is given by*

$$P = 1 - \left[F\left(\delta, \epsilon_N\right)\right]^N > 0, \tag{4.23}$$

where $\epsilon_N \coloneqq \delta^+ - \delta_N$ *and*

$$F\left(\delta, \epsilon_N\right) = \Pr\left(\delta - \|\widehat{\mathbf{u}}(\mathbf{x}) - \mathbf{u}^*(\mathbf{x})\|_2 > \epsilon_N\right) < 1. \tag{4.24}$$

Proof. Noting that Assumption 5 implies $\mathbf{u}^*(\cdot)$ is continuous in $\mathbf{x}$, Lemma 2 yields

$$P\left(\delta, \delta_N, N\right) \coloneqq \Pr\left(\delta < \delta_N + \epsilon_N\right) = 1 - \Pr\left(\delta \geq \delta_N + \epsilon_N\right) = 1 - \left[F\left(\delta, \epsilon_N\right)\right]^N.$$

Though δ is fixed, (4.23) tells us the probability that δ_N is close *enough* to δ so that we can expect (4.15) holds, which by Lemma 1 implies the trajectories of closed loop system are ultimately bounded. □

While stability is a deterministic property of the system (i.e. the system either is stable or it isn't), we can loosely think of Proposition 1 as a (conservative) probabilistic stability condition based on test error. In general, it may be difficult to apply this result quantitatively because it requires knowledge about the Lyapunov function $W(\cdot)$ and the true maximum error δ. Nevertheless, we believe that Proposition 1 begins to qualitatively explain the phenomenon seen in Section 4.1, where NNs with similar test accuracy can sometimes produce stable systems and other times yield trajectories like the one shown in Figure 4.1.

4.3 Summary

In this chapter we have seen that the core algorithm introduced in Chapter 2 can be used to synthesize nearly optimal feedback controllers for nonlinear systems with $n = 64$ dimensions. This is a nontrivial achievement. But we have also seen that the success can be somewhat unpredictable.

In supervised learning test accuracy statistics are typically used to quantify a model's quality. But for our application the ultimate goal is really semi-global stability and optimality of the closed loop dynamics. By the numerical experiments in Section 4.1 we have clearly shown that test accuracy correlates with stability and performance, but cannot always predict how the NN will behave when implemented in the nonlinear system. Our theoretical analysis in Section 4.2 qualitatively supports our observation that test accuracy is *probabilistically* related to nonlinear stability.

This uncertainty would render our proposed approach impractical since we would not know if a controller is even stabilizing until after extensive testing. Hence these results motivate our work in Chapter 5 where we explore NN architectures that guarantee at least LAS.

Chapter 5

QRnet: Stability-enhancing Neural Network Architectures for Optimal Feedback Design

---

In Chapters 2 and 3 we saw that characteristics-based supervised learning can be an effective tool for designing optimal feedback controllers for high-dimensional nonlinear dynamical systems. In Chapter 4 we saw that it was possible to apply the core framework to learn controllers for systems with even $n = 64$ dimensions. However, we also saw that some NNs with high test accuracy can fail to even locally stabilize the dynamics. Overall, the behavior of NN-controlled systems is still not well understood. Studying this problem in further detail – with both numerical experiments and theoretical analysis – promises to be a rich area for future research. But we can also address this challenge from a practical perspective by building stability-enhancing properties into the computational framework itself.

In Chapter 5 we develop a series of NN architectures which can mitigate some of these challenges. Several of these architectures guarantee, at a minimum, LAS of the system. This is accomplished by exactly recovering the LQR gain at the origin by construction. Incorporating a local LQR in this fashion serves as another way to leverage existing results from control theory, thereby enhancing our data-driven computational algorithm. We also prove a universal approximation theorem for NNs with this structure, showing that they can approximate the *nonlinear* optimal feedback law up to arbitrary accuracy. Consequently the proposed architectures can provide semi-global stability and optimality. We call this group of architectures *QRnet*. To state the obvious, *QRnet* combines "Quadratic Regulator" and "network". There is no "L" because the model is nonlinear. Prefixes to *QRnet* will indicate variations on this theme.

Guaranteeing semi-global stability is exceedingly difficult to do by construction. Recall that some SA methods come with such guarantees, but these depend on control architectures which are not suitable for high-dimensional problems, require assuming a special system structure, or need to be initialized with a stabilizing control policy. Our supervised learning approach does not have these restrictions, and we have already seen it is capable of synthesizing nonlinear feedback controllers with near-optimal performance. Moreover, we saw in Section 4.1 that stability of the nonlinear system was closely connected with linear stability, and hence we hypothesize that building in linear stability will translate to improved nonlinear stability. Ultimately this is a key step in making our proposed control design framework more reliable.

Figure 5.1 highlights the stability-enhancing architectures introduced in Chapter 5 within the greater computational framework. This chapter is organized as follows. In Section 5.1 we describe the novel *QRnet* architectures and present the-

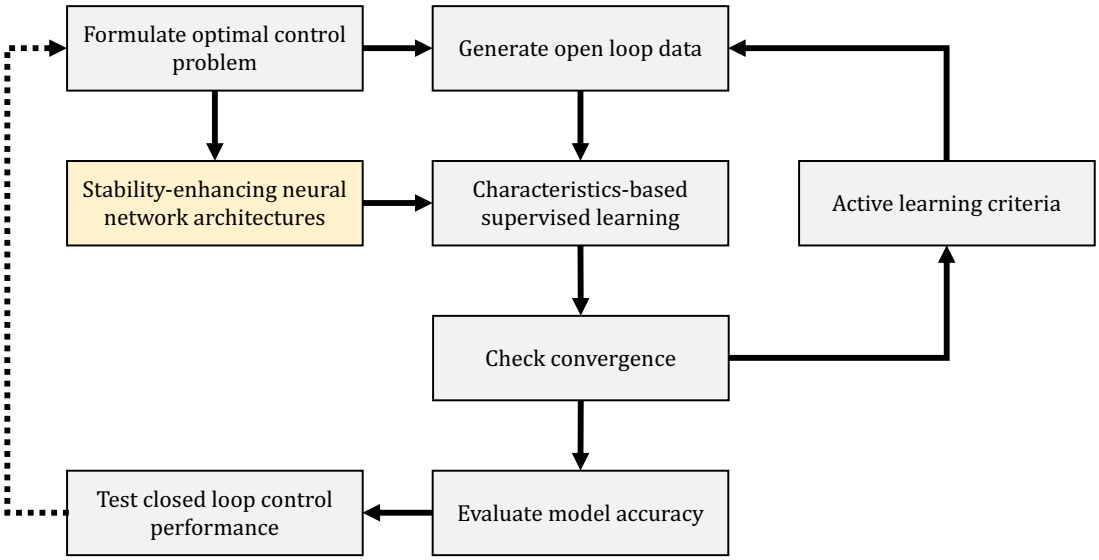


Figure 5.1: Summary of the of the steps and components of the proposed computational framework, highlighting the focus of Chapter 5.

ory underpinning their local stability properties and their ability to approximate the full nonlinear optimal feedback policy. In Section 5.2 we will revisit the Burgers' PDE example from Section 4.1. We demonstrate that the new architectures are always at least LAS. They are also quite consistently nearly optimal, indicating that we are not making tradeoffs in terms of optimality. Then in Section 5.3 we apply the proposed control design methodology to learn optimal feedback controllers for a 6DoF UAV with nonlinear dynamics and aerodynamics, showing that the computational framework can be applied to practical problems. A brief summary is given in Section 5.4.

The material presented in this chapter is drawn from [98, 99, 100]. The proofs of Proposition 3 and Theorem 2 have not been published elsewhere as they consist of straightforward calculations. They are included in this dissertation for completeness.

5.1 Architectures for optimal feedback design

Our goal is to learn a feedback policy which approximates the optimal control, i.e. $\widehat{\mathbf{u}}(\mathbf{x}) \approx \mathbf{u}^*(\mathbf{x})$. So far we have clearly demonstrated the potential of deep learning as a means of overcoming the curse of dimensionality in optimal control. But NNs are notoriously "black boxes" and their behavior – especially when implemented in the closed loop system – is complex and hard to predict. Notably, even if we can train a highly accurate NN, it can still fail to stabilize the system. Thus there is a clear need for designing NN feedback controllers with *built-in* stability properties.

To this end we explore a number of NN architectures which smoothly combine LQR controller with NNs. The LQR terms are good approximations of the optimal control near $\mathbf{x}_f$, and improve or in some cases guarantee LAS. Meanwhile, the NNs are intended to capture nonlinearities and thereby learn the nonlinear optimal feedback over a large domain.

The first of these architectures, V-*QRnet*, is introduced in Section 5.1.1. It is intended to be self-explanatory from the title that V-*QRnet* is an approximation of the value function analogous to V-NN. Next in Section 5.1.3 we introduce λ-*QRnet* and u-*QRnet*, which are slight modifications of λ-NN and u-NN, respectively. We will find that all of these first three architectures improve local stability in practice. In addition, λ-*QRnet* and u-*QRnet* ensure that $\mathbf{x}_f$ is a closed loop equilibrium.

Still, none of these architectures *guarantee* LAS, which motivates us to pursue alternative designs. In Section 5.1.4 we introduce λ_{Jac}-*QRnet* and u_{Jac}-*QRnet*. These are modifications of λ-*QRnet* and u-*QRnet* which exactly recover the LQR control locally at $\mathbf{x}_f$. Then in Section 5.1.5 we propose introduce λ_{mat}-*QRnet* and u_{mat}-*QRnet* which also exactly recover the LQR control at $\mathbf{x}_f$ but using matrix-

	LAS guarantee	number of NN parameters (p)	control evaluation time
V-NN*,† (2.1)	-	$\sim 2w + Lw^2$	0.8 [ms]
V-*QRnet**,† (5.1)	-	$\sim 2w + Lw^2$	1.2 [ms]
V-*QRnet* (alt.)*,†,‡ (5.3)	-	$\sim 2w + Lw^2$	$\sim$ 1.2 [ms]
V_{Hess}-*QRnet**,†,‡ (5.4)	✓	$\sim 2w + Lw^2$	$\sim$ 1.2 [ms]
λ-NN† (2.3)	-	$\sim 2wn + Lw^2$	0.4 [ms]
λ-*QRnet*† (5.5)	-	$\sim 2wn + Lw^2$	0.5 [ms]
λ_{Jac}-*QRnet*† (5.7)	✓	$\sim 2wn + Lw^2$	0.5 [ms]
λ_{mat}-*QRnet*† (5.9)	✓	$\sim wn^2 + wn + Lw^2$	0.7 [ms]
u-NN (2.17)	-	$\sim wm + wn + Lw^2$	0.4 [ms]
u-*QRnet* (5.6)	-	$\sim wm + wn + Lw^2$	0.5 [ms]
u_{Jac}-*QRnet* (5.8)	✓	$\sim wm + wn + Lw^2$	0.5 [ms]
u_{mat}-*QRnet* (5.10)	✓	$\sim wmn + wn + Lw^2$	0.6 [ms]

Table 5.1: Summary of NN control architectures in this dissertation. L denotes the number of layers and w is their width. Control evaluation times are for NN controllers for the Burgers' OCP (4.6) run on a 2019 MacBook Air. Notes: *Need to take gradients with respect to $\mathbf{x}$ at evaluation time. †Practical implementation requires solving (1.18) for $\mathbf{u}^*(\cdot)$ in terms of $\mathbf{x}$ and $\boldsymbol{\lambda}$. ‡Not yet implemented; control evaluation time estimated.

valued NNs. In Section 5.1.6 we prove that these "Jacobian" and "matrix" *QRnet*s all guarantee LAS of $\mathbf{x}_f$, and finally in Section 5.1.7 we prove that they retain the approximation capacity to approximate the full nonlinear optimal feedback control on semi-global domains.

V QRnet is trained by solving (2.25) exactly as for V-NN. λ-*QRnet*, λ_{Jac}-*QRnet*, and λ_{mat}-*QRnet* are trained by solving (2.26) exactly as for λ-NN. Similarly, u-*QRnet*, u_{Jac}-*QRnet*, and u_{mat}-*QRnet* are trained by solving (2.27) exactly as for u-NN. A summary of the NN control architectures discussed in this dissertation is given in Table 5.1.

Remark 3. *It should also be noted that the proposed* QRnet *architectures do not have to trained using supervised learning. In principle they can also be implemented with self-supervised learning, or more generally whenever we can compute*

an LQR or other LAS control law.

5.1.1 *V-QRnet*

As a first attempt we seek a value function model $\widehat{V}(\cdot)$ which combines (1.26) for the linearized OCP with an NN to learn the higher order terms. This results in *V-QRnet* (originally just called *QRnet* [98]):

$$\widehat{V}(\mathbf{x};\boldsymbol{\theta}) = \frac{1}{\gamma}\log\left[1+\gamma V^{\mathrm{LQR}}(\mathbf{x})\right] + \mathcal{N}(\mathbf{x};\boldsymbol{\theta}), \tag{5.1}$$

where $\gamma > 0$ is a trainable parameter, $V^{\mathrm{LQR}}(\mathbf{x}) = (\mathbf{x}-\mathbf{x}_f)^T\mathbf{P}(\mathbf{x}-\mathbf{x}_f)$, and $\mathcal{N}:\mathbb{R}^n\times\mathbb{R}^p\to\mathbb{R}$ is an NN. Since (5.1) is just the sum of a bounded, smooth function and an NN, it is easy to see that it still has universal approximation capacity (Theorem 1).

Intuitively, LQR provides a good approximation near the equilibrium $\mathbf{x}_f$. There $V^{\mathrm{LQR}}(\mathbf{x})$ is small and hence $\gamma^{-1}\log\left[1+\gamma V^{\mathrm{LQR}}(\mathbf{x})\right] \approx V^{\mathrm{LQR}}(\mathbf{x})$ for all $\gamma\in(0,\infty)$. Further away from $\mathbf{x}_f$, we have $\gamma^{-1}\log\left[1+\gamma V^{\mathrm{LQR}}(\mathbf{x})\right] \ll V^{\mathrm{LQR}}(\mathbf{x})$, thereby increasing the relative importance of the corrective NN. The parameter γ governs the radius in which this term approximates $V^{\mathrm{LQR}}(\mathbf{x})$; in particular

$$\lim_{\gamma\to 0}\gamma^{-1}\log\left[1+\gamma V^{\mathrm{LQR}}(\mathbf{x})\right] = V^{\mathrm{LQR}}(\mathbf{x}). \tag{5.2}$$

Notice that the model structure (5.1) is similar to a series expansion, except that it explicitly reduces the impact of lower order terms away from the linearization point.

5.1.2 Alternative *V-QRnet* architectures

Although *V-QRnet* (5.1) empirically improves stability properties, it does not guarantee that the goal state $\mathbf{x}_f$ will be stable, let alone an equilibrium. To handle these challenges we propose the following modification:

$$\widehat{V}(\mathbf{x};\boldsymbol{\theta}) = \frac{1}{\gamma}\log\left[1+\gamma V^{\mathrm{LQR}}(\mathbf{x})\right] + \mathcal{N}(\mathbf{x};\boldsymbol{\theta}) - \mathcal{N}(\mathbf{x}_f;\boldsymbol{\theta}) - \left[\frac{\partial \mathcal{N}}{\partial \mathbf{x}}(\mathbf{x}_f;\boldsymbol{\theta})\right](\mathbf{x}-\mathbf{x}_f). \tag{5.3}$$

Straightforward computations (see Proposition 2) show that an NN controller $\widehat{\mathbf{u}}(\mathbf{x}) = \mathbf{u}^*\left(\mathbf{x}; \widehat{V}_{\mathbf{x}}(\mathbf{x};\boldsymbol{\theta})\right)$ using the gradient $\widehat{V}_{\mathbf{x}}(\cdot)$ of (5.3) automatically makes $\mathbf{x}_f$ an equilibrium. This modified *V-QRnet* architecture is similar to the "Jacobian" *QRnets* which we introduce in Section 5.1.4, and universal approximation for locally $\mathcal{C}^2$ functions can be proved along the lines of Theorem 4.

However, the Hessian $\frac{\partial^2 \widehat{V}}{\partial \mathbf{x}^2}(\mathbf{x}_f;\boldsymbol{\theta})$ of (5.3) will not in general be exactly equal to $2\mathbf{P}$. Consequently the linearized NN control will not recover the LQR gain. If the perturbation is large enough then there is a chance that the system will not be LAS. To guarantee LAS by construction we can take (5.3) a step further and specify

$$\begin{aligned}\widehat{V}(\mathbf{x};\boldsymbol{\theta}) =& \frac{1}{\gamma}\log\left(1+\gamma(\mathbf{x}-\mathbf{x}_f)^T\left[\mathbf{P}-\frac{1}{2}\cdot\frac{\partial^2\mathcal{N}}{\partial\mathbf{x}^2}(\mathbf{x}_f;\boldsymbol{\theta})\right](\mathbf{x}-\mathbf{x}_f)\right)\\ &+\mathcal{N}(\mathbf{x};\boldsymbol{\theta}) - \mathcal{N}(\mathbf{x}_f;\boldsymbol{\theta}) - \left[\frac{\partial \mathcal{N}}{\partial \mathbf{x}}(\mathbf{x}_f;\boldsymbol{\theta})\right](\mathbf{x}-\mathbf{x}_f).\end{aligned} \tag{5.4}$$

Now (5.4) exactly recovers the LQR value function $V^{\mathrm{LQR}}(\cdot)$ around $\mathbf{x}_f$. As a result it recovers the true value function up to second order there, and exactly recovers the LQR control at $\mathbf{x}_f$ thus guaranteeing LAS (see Proposition 3). Universal approximation for locally $\mathcal{C}^2$ functions can again be proved along the lines of

Theorem 4. We call this alternative value function model V_{Hess}-*QRnet* based on the construction with the NN Hessian.

At the time of writing we have *not yet implemented and tested* the alternative V-*QRnet* architectures (5.3) and (5.4). We conjecture that training them, in particular (5.4), will be expensive due to the NN gradient and Hessian terms. Computing the approximate value gradient, $\widehat{V}_{\mathbf{x}}(\cdot)$, will require yet another gradient. Once the models are trained, however, they should be just as fast as the standard V-*QRnet* because the NN gradient and Hessian, $\frac{\partial \mathcal{N}}{\partial \mathbf{x}}(\mathbf{x}_f;\boldsymbol{\theta})$ and $\frac{\partial^2 \mathcal{N}}{\partial \mathbf{x}^2}(\mathbf{x}_f;\boldsymbol{\theta})$, can be stored in memory.

5.1.3 λ-*QRnet* and u-*QRnet*

It turns out that we can more easily force $\mathbf{x}_f$ to be a (stable) equilibrium if we directly approximate the value gradient or optimal control instead of the value function. This motivated the development of λ-*QRnet* and u-*QRnet*, which are straightforward linear combinations of LQR with NNs.

We start with λ-*QRnet*, which approximates the value gradient $V_{\mathbf{x}}(\cdot)$. It can be implemented when we can solve (1.18) for an explicit formula the optimal feedback control in terms of the state and value gradient, as is the case for many problems of interest (see Section 2.3.5). λ-*QRnet* is specified as

$$\widehat{\boldsymbol{\lambda}}(\mathbf{x};\boldsymbol{\theta}) = 2\mathbf{P}(\mathbf{x}-\mathbf{x}_f) + \boldsymbol{\mathcal{N}}(\mathbf{x};\boldsymbol{\theta}) - \boldsymbol{\mathcal{N}}(\mathbf{x}_f;\boldsymbol{\theta}). \tag{5.5}$$

Here $\boldsymbol{\mathcal{N}} : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^n$ is an NN with $\mathcal{C}^1$ activation functions and parameters $\boldsymbol{\theta} \in \mathbb{R}^p$, and the linear component $2\mathbf{P}(\mathbf{x}-\mathbf{x}_f)$ is gradient of the LQR value function.

Alternatively, we can directly approximate the optimal control with *u-QRnet*:

$$\widehat{\mathbf{u}}(\mathbf{x};\boldsymbol{\theta}) = \sigma\left[\text{sat}\left(\mathbf{u}^{\text{LQR}}(\mathbf{x})\right) + \boldsymbol{\mathcal{N}}(\mathbf{x};\boldsymbol{\theta}) - \boldsymbol{\mathcal{N}}(\mathbf{x}_f;\boldsymbol{\theta})\right], \tag{5.6}$$

where now $\boldsymbol{\mathcal{N}} : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^m$. Recall that $\text{sat}(\cdot)$ is the hard saturation function (2.16), and $\sigma : \mathbb{R}^m \to \mathbb{R}^m$ is the smooth saturation function defined by (2.18–2.19). Recall also that this choice of saturation function preserves the unsaturated control behavior near $\mathbf{u}_f$.

It is easy to show (see Proposition 2) that these architectures make the goal state $\mathbf{x}_f$ an equilibrium. In contrast this is *not* true for an arbitrary NN controller. Including the LQR term also promotes local stability because LQR's large gain and phase margins tend to be robust to perturbations from the NN Jacobian. Still, these models do not exactly recover LQR and so LAS cannot be assured by construction alone.

5.1.4 "Jacobian" *QRnet* architectures

Now we describe λ_{Jac}-*QRnet* and u_{Jac}-*QRnet*. These are similar to λ-*QRnet* and *u-QRnet*, except that we subtract the Jacobian of the NN components. As we show in Section 5.1.6 this ensures that the controllers exactly recover LQR at $\mathbf{x}_f$, thus guaranteeing LAS. Furthermore, in Section 5.1.7 we prove that these architectures retain the nonlinear function approximation capacity of standard feedforward NNs, thus allowing them to approximate the full nonlinear value gradient and optimal control.

First we have λ_{Jac}-*QRnet*:

$$\widehat{\boldsymbol{\lambda}}(\mathbf{x};\boldsymbol{\theta}) = \left[2\mathbf{P} - \frac{\partial \boldsymbol{\mathcal{N}}}{\partial \mathbf{x}}(\mathbf{x}_f;\boldsymbol{\theta})\right](\mathbf{x} - \mathbf{x}_f) + \boldsymbol{\mathcal{N}}(\mathbf{x};\boldsymbol{\theta}) - \boldsymbol{\mathcal{N}}(\mathbf{x}_f;\boldsymbol{\theta}). \tag{5.7}$$

u_{Jac}-*QRnet* has an analogous structure:

$$\widehat{\mathbf{u}}(\mathbf{x};\boldsymbol{\theta}) = \sigma\left[\text{sat}\left(\mathbf{u}^{\text{LQR}}(\mathbf{x})\right) - \left[\frac{\partial \boldsymbol{\mathcal{N}}}{\partial \mathbf{x}}(\mathbf{x}_f;\boldsymbol{\theta})\right](\mathbf{x}-\mathbf{x}_f) + \boldsymbol{\mathcal{N}}(\mathbf{x};\boldsymbol{\theta}) - \boldsymbol{\mathcal{N}}(\mathbf{x}_f;\boldsymbol{\theta})\right]. \tag{5.8}$$

These models are slower to train than λ-*QRnet* (5.5) and u-*QRnet* (5.6) since the Jacobian $\frac{\partial \boldsymbol{\mathcal{N}}}{\partial \mathbf{x}}(\mathbf{x}_f;\boldsymbol{\theta})$ must be evaluated during each forward pass. *After the NN has been trained, however, we can store the Jacobian matrix in memory so that it does not have to be recomputed online.* Therefore online control evaluation is just as fast as λ-*QRnet* and u-*QRnet*.

5.1.5 "Matrix" *QRnet* architectures

In this section we describe λ_{mat}-*QRnet* and u_{mat}-*QRnet*. These alternatives to the "Jacobian"-style architectures employ *matrix-valued* NNs. Thus they avoid the costly Jacobian computations in exchange for having to optimize more NN parameters. These "matrix" *QRnet*s enjoy the same stability and approximation properties as the "Jacobian" *QRnet*s.

First consider λ_{mat}-*QRnet*:

$$\widehat{\boldsymbol{\lambda}}(\mathbf{x};\boldsymbol{\theta}) = \left[2\mathbf{P} + \boldsymbol{\mathcal{N}}(\mathbf{x};\boldsymbol{\theta}) - \boldsymbol{\mathcal{N}}(\mathbf{x}_f;\boldsymbol{\theta})\right](\mathbf{x}-\mathbf{x}_f). \tag{5.9}$$

Notice that in this case $\boldsymbol{\mathcal{N}} : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^{n\times n}$ is matrix-valued. Next we have u_{mat}-*QRnet*:

$$\widehat{\mathbf{u}}(\mathbf{x};\boldsymbol{\theta}) = \sigma\left[\text{sat}\left(\mathbf{u}^{\text{LQR}}(\mathbf{x})\right) - \left[\boldsymbol{\mathcal{N}}(\mathbf{x};\boldsymbol{\theta}) - \boldsymbol{\mathcal{N}}(\mathbf{x}_f;\boldsymbol{\theta})\right](\mathbf{x}-\mathbf{x}_f)\right], \tag{5.10}$$

where now $\boldsymbol{\mathcal{N}} : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^{m\times n}$.

A drawback of λ_{mat}-*QRnet* (5.9) is that the number of NN parameters scales with $O\left(Lw^2 + wn^2\right)$, where L is the number of layers and w is their width. For high dimensional OCPs, this can make (5.9) challenging to train as well as deploy on small processors. Meanwhile, the number of NN parameters in (5.10) scales with $O\left(Lw^2 + wmn\right)$. Since we typically have $m \ll n$, u_{mat}-*QRnet* can be much smaller and hence much faster to train than λ_{mat}-*QRnet*.

Compared to the corresponding "Jacobian" QRnet architectures, λ_{mat}-*QRnet* and u_{mat}-*QRnet* are slightly slower for online computation since the NNs are larger. In terms of performance we have not found a consistent advantage of one type; their relative learning ability appears to be problem-dependent.

5.1.6 Local asymptotic stability guarantees

Having described the various *QRnet* architectures let us study their local stability properties. First we will verify that all the new architectures – excepte V-*QRnet* – automatically make the goal state $\mathbf{x}_f$ an equilibrium of the NN-controlled system. This property, stated formally in Proposition 2 below, is a consequence of subtracting the NN contribution $\boldsymbol{\mathcal{N}}\left(\mathbf{x}_f;\boldsymbol{\theta}\right)$, and has also been suggested by [78]. In practice we find that the large gain and phase margins of LQR [145, 130, 44] are mostly robust to small perturbations from the NN Jacobian, $\frac{\partial \boldsymbol{\mathcal{N}}}{\partial \mathbf{x}}\left(\mathbf{x}_f;\boldsymbol{\theta}\right)$, so in most cases we recover LAS.

The "Jacobian" and "matrix" architectures take this a step further and guarantee at least LAS at $\mathbf{x}_f$. Specifically, if we linearize the feedback control $\widehat{\mathbf{u}}(\cdot)$ at $\mathbf{x}_f$ then we recover the LQR control gain (1.27). This holds even when the models are poorly trained. This property is desirable because LQR locally asymptotically stabilizes $\mathbf{x}_f$, and hence the proposed controllers provide LAS by construction.

LQR is also locally optimal and therefore we start with first order optimality by construction. This property is stated formally in Proposition 3 below.

Proposition 2 (Closed loop equilibria)**.** *Suppose* $\widehat{\mathbf{u}}(\cdot)$ *is a feedback policy specified by* (2.5) *with* (5.3), (5.5), (5.7), *or* (5.9)*; or by* (5.6), (5.8), *or* (5.10)*. Then* $\mathbf{x}_f$ *is an equilibrium of the NN-controlled system,* $\dot{\mathbf{x}} = \mathbf{f}\left(\mathbf{x}, \widehat{\mathbf{u}}(\mathbf{x})\right)$.

Proof. First we consider NNs which approximate the value function or value gradient. Evaluating (5.5), (5.7), or (5.9) at $\mathbf{x} = \mathbf{x}_f$ gives $\widehat{\boldsymbol{\lambda}}\left(\mathbf{x}_f\right) = \mathbf{0} = V_{\mathbf{x}}\left(\mathbf{x}_f\right)$ which implies

$$\widehat{\mathbf{u}}\left(\mathbf{x}_f\right) = \mathbf{u}^*\left(\mathbf{x}; \widehat{\boldsymbol{\lambda}}\left(\mathbf{x}_f\right)\right) = \mathbf{u}^*\left(\mathbf{x}; V_{\mathbf{x}}\left(\mathbf{x}_f\right)\right) = \mathbf{u}_f.$$

Similarly for (5.3) some quick computations show $\widehat{V}_{\mathbf{x}}\left(\mathbf{x}_f\right) = \mathbf{0}$ so again $\widehat{\mathbf{u}}\left(\mathbf{x}_f\right) = \mathbf{u}_f$. Thus for these architectures we conclude $\dot{\mathbf{x}} = \mathbf{f}\left(\mathbf{x}_f, \widehat{\mathbf{u}}\left(\mathbf{x}_f\right)\right) = \mathbf{f}\left(\mathbf{x}_f, \mathbf{u}_f\right) = \mathbf{0}$.

Next we consider NNs which approximate the optimal control. Evaluating (5.6), (5.8), or (5.10) at $\mathbf{x} = \mathbf{x}_f$ gives $\widehat{\mathbf{u}}\left(\mathbf{x}_f\right) = \sigma\left[\text{sat}\left(\mathbf{u}^{\text{LQR}}\left(\mathbf{x}_f\right)\right)\right] = \mathbf{u}_f$. Consequently $\dot{\mathbf{x}} = \mathbf{f}\left(\mathbf{x}_f, \widehat{\mathbf{u}}\left(\mathbf{x}_f\right)\right) = \mathbf{f}\left(\mathbf{x}_f, \mathbf{u}_f\right) = \mathbf{0}$, as desired. □

To prove Proposition 3 we will need the following lemma which can be found in e.g. [88].

Lemma 3 (Linearization of the optimal control [88])**.** *Suppose that* $\mathbf{u}^*(\mathbf{x}) = \mathbf{u}^*\left(\mathbf{x}; V_{\mathbf{x}}(\mathbf{x})\right)$ *is the optimal feedback control which solves* (1.4)*. Then sufficiently near* $\mathbf{x}_f$, $\mathbf{u}^*(\cdot)$ *is* $\mathcal{C}^1$ *and is given by*

$$\mathbf{u}^*(\mathbf{x}; \boldsymbol{\lambda}) = \mathbf{u}_f - \frac{1}{2}\mathbf{R}^{-1}\mathbf{B}^T\boldsymbol{\lambda} + \mathbf{h}\left(\mathbf{x}; \boldsymbol{\lambda}\right), \tag{5.11}$$

where $\mathbf{h} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^m$ *contains higher order terms which vanish at* $\mathbf{x}_f$*. Consequently*

$$\frac{\partial \mathbf{u}^*}{\partial \mathbf{x}}\left(\mathbf{x}_f\right) = \mathbf{0}, \qquad \frac{\partial \mathbf{u}^*}{\partial \boldsymbol{\lambda}}\left(\mathbf{x}_f\right) = -\frac{1}{2}\mathbf{R}^{-1}\mathbf{B}^T. \tag{5.12}$$

Furthermore, taking $\boldsymbol{\lambda} = V_{\mathbf{x}}(\mathbf{x})$ *so* $\mathbf{u}^*(\mathbf{x}) = \mathbf{u}^*\left(\mathbf{x}; V_{\mathbf{x}}(\mathbf{x})\right)$ *we get*

$$\frac{\partial \mathbf{u}^*}{\partial \mathbf{x}}(\mathbf{x}_f) = -\mathbf{K} = -\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P}. \tag{5.13}$$

We are now ready to demonstrate LAS. Here we drop the notational dependence of the NNs $\boldsymbol{\mathcal{N}}(\cdot)$ on parameters $\boldsymbol{\theta}$, since these do not affect the local stability properties.

Proposition 3 (LAS guarantees)**.** *Suppose* $\widehat{\mathbf{u}}(\cdot)$ *is a feedback policy specified by* (2.5) *with eq: value-hess-QRnet architecture,* (5.7)*, or* (5.9)*; or by* (5.8) *or* (5.10)*. Then*

$$\frac{\partial \widehat{\mathbf{u}}}{\partial \mathbf{x}}(\mathbf{x}_f) = -\mathbf{K} \tag{5.14}$$

and $\mathbf{x}_f$ *is an LAS equilibrium of the NN-controlled system,* $\dot{\mathbf{x}} = \mathbf{f}\left(\mathbf{x}, \widehat{\mathbf{u}}(\mathbf{x})\right)$.

Proof. We know from Proposition 2 that $\mathbf{x}_f$ is a closed loop equilibrium for each controller. Let us then verify that $\frac{\partial \widehat{\mathbf{u}}}{\partial \mathbf{x}}(\mathbf{x}_f) = -\mathbf{K}$ for each architecture. If this holds then LAS follows immediately from LAS of the LQR-controlled system.

We start with V_{Hess}*-QRnet.* The Jacobian $\partial\widehat{\mathbf{u}}/\partial\mathbf{x}$ as defined by (2.2) with (5.4) is given by

$$\frac{\partial \widehat{\mathbf{u}}}{\partial \mathbf{x}} = \frac{\partial \mathbf{u}^*}{\partial \mathbf{x}} + \frac{\partial \mathbf{u}^*}{\partial \boldsymbol{\lambda}}\frac{\partial^2 \widehat{V}}{\partial \mathbf{x}^2}. \tag{5.15}$$

We compute

$$\frac{\partial \widehat{V}}{\partial \mathbf{x}}(\mathbf{x}) = \frac{\left[2\mathbf{P} - \frac{\partial^2 \mathcal{N}}{\partial \mathbf{x}^2}(\mathbf{x}_f)\right](\mathbf{x} - \mathbf{x}_f)}{1 + \gamma\,(\mathbf{x} - \mathbf{x}_f)^T\left[\mathbf{P} - \frac{1}{2}\cdot\frac{\partial^2 \mathcal{N}}{\partial \mathbf{x}^2}(\mathbf{x}_f)\right](\mathbf{x} - \mathbf{x}_f)} + \frac{\partial \mathcal{N}}{\partial \mathbf{x}}(\mathbf{x}) - \frac{\partial \mathcal{N}}{\partial \mathbf{x}}(\mathbf{x}_f) \tag{5.16}$$

and, applying the quotient rule,

$$\begin{aligned}\frac{\partial^2 \widehat{V}}{\partial \mathbf{x}^2}(\mathbf{x}) =& \frac{\left[1+\gamma(\mathbf{x}-\mathbf{x}_f)^T\left[\mathbf{P}-\frac{1}{2}\cdot\frac{\partial^2\mathcal{N}}{\partial\mathbf{x}^2}(\mathbf{x}_f)\right](\mathbf{x}-\mathbf{x}_f)\right]\left[2\mathbf{P}-\frac{\partial^2\mathcal{N}}{\partial\mathbf{x}^2}(\mathbf{x}_f)\right]}{\left[1+\gamma(\mathbf{x}-\mathbf{x}_f)^T\left[\mathbf{P}-\frac{1}{2}\cdot\frac{\partial^2\mathcal{N}}{\partial\mathbf{x}^2}(\mathbf{x}_f)\right](\mathbf{x}-\mathbf{x}_f)\right]^2} \\ & - \frac{\gamma\left[\left[2\mathbf{P}-\frac{\partial^2\mathcal{N}}{\partial\mathbf{x}^2}(\mathbf{x}_f)\right](\mathbf{x}-\mathbf{x}_f)\right]\left[\left[2\mathbf{P}-\frac{\partial^2\mathcal{N}}{\partial\mathbf{x}^2}(\mathbf{x}_f)\right](\mathbf{x}-\mathbf{x}_f)\right]^T}{\left[1+\gamma(\mathbf{x}-\mathbf{x}_f)^T\left[\mathbf{P}-\frac{1}{2}\cdot\frac{\partial^2\mathcal{N}}{\partial\mathbf{x}^2}(\mathbf{x}_f)\right](\mathbf{x}-\mathbf{x}_f)\right]^2} \\ & + \frac{\partial^2\mathcal{N}}{\partial\mathbf{x}^2}(\mathbf{x})\,. \end{aligned} \tag{5.17}$$

At $\mathbf{x}_f$ this becomes just

$$\frac{\partial^2 \widehat{V}}{\partial \mathbf{x}^2}(\mathbf{x}_f) = \frac{(1+\gamma\cdot 0)\left[2\mathbf{P}-\frac{\partial^2\mathcal{N}}{\partial\mathbf{x}^2}(\mathbf{x}_f)\right]-\gamma\mathbf{0}_{n\times n}}{(1+\gamma\cdot 0)^2} + \frac{\partial^2\mathcal{N}}{\partial\mathbf{x}^2}(\mathbf{x}_f) = 2\mathbf{P}.$$

Then evaluating the linearized control (5.15) at $\mathbf{x}_f$ and applying Lemma 3 yields $\frac{\partial\widehat{\mathbf{u}}}{\partial\mathbf{x}}(\mathbf{x}_f) = \mathbf{0} - \frac{1}{2}\mathbf{R}^{-1}\mathbf{B}^T[2\mathbf{P}] = -\mathbf{K}$.

Next for λ_{Jac}-*QRnet*, the Jacobian $\partial\widehat{\mathbf{u}}/\partial\mathbf{x}$ as defined by (2.5) with (5.7) is given by

$$\frac{\partial\widehat{\mathbf{u}}}{\partial\mathbf{x}} = \frac{\partial\mathbf{u}^*}{\partial\mathbf{x}} + \frac{\partial\mathbf{u}^*}{\partial\boldsymbol{\lambda}}\left[2\mathbf{P} - \frac{\partial\boldsymbol{\mathcal{N}}}{\partial\mathbf{x}}(\mathbf{x}_f) + \frac{\partial\boldsymbol{\mathcal{N}}}{\partial\mathbf{x}}(\mathbf{x})\right]. \tag{5.18}$$

Evaluating the above linearization at $\mathbf{x} = \mathbf{x}_f$ and applying Lemma 3 yields

$$\begin{aligned}\frac{\partial\widehat{\mathbf{u}}}{\partial\mathbf{x}}(\mathbf{x}_f) =& \mathbf{0} - \frac{1}{2}\mathbf{R}^{-1}\mathbf{B}^T\left[2\mathbf{P} - \frac{\partial\boldsymbol{\mathcal{N}}}{\partial\mathbf{x}}(\mathbf{x}_f) + \frac{\partial\boldsymbol{\mathcal{N}}}{\partial\mathbf{x}}(\mathbf{x}_f)\right] \\ =& -\frac{1}{2}\mathbf{R}^{-1}\mathbf{B}^T[2\mathbf{P}] \\ =& -\mathbf{K}.\end{aligned}$$

Next consider λ_{mat}-*QRnet*, for which the Jacobian is given by

$$\frac{\partial \widehat{\mathbf{u}}}{\partial \mathbf{x}} = \frac{\partial \mathbf{u}^*}{\partial \mathbf{x}} + \frac{\partial \mathbf{u}^*}{\partial \boldsymbol{\lambda}} \left[2\mathbf{P} + \boldsymbol{\mathcal{N}}(\mathbf{x}) - \boldsymbol{\mathcal{N}}(\mathbf{x}_f) + \left[\frac{\partial \boldsymbol{\mathcal{N}}}{\partial \mathbf{x}}(\mathbf{x}) \right] (\mathbf{x} - \mathbf{x}_f) \right], \tag{5.19}$$

where $\left[\frac{\partial \boldsymbol{\mathcal{N}}}{\partial \mathbf{x}}(\mathbf{x})\right](\mathbf{x} - \mathbf{x}_f) : \mathbb{R}^n \to \mathbb{R}^{n \times n}$ is a tensor product. Evaluating the above linearization at $\mathbf{x} = \mathbf{x}_f$ and applying Lemma 3 yields

$$\frac{\partial \widehat{\mathbf{u}}}{\partial \mathbf{x}}(\mathbf{x}_f) = \mathbf{0} - \frac{1}{2}\mathbf{R}^{-1}\mathbf{B}^T [2\mathbf{P}] = -\mathbf{K}.$$

Next we compute the Jacobian of u_{Jac}-*QRnet*. For $\mathbf{x}$ sufficiently near $\mathbf{x}_f$, $\mathbf{u}^{\text{LQR}}(\mathbf{x})$ is unsaturated and thus

$$\begin{aligned} \frac{\partial \widehat{\mathbf{u}}}{\partial \mathbf{x}} = &\operatorname{diag}\left[\frac{\mathbf{c}_1 \mathbf{c}_2 (\mathbf{u}_{\max} - \mathbf{u}_{\min}) e^{-\mathbf{c}_2(\mathbf{u} - \mathbf{u}_f)}}{\left(1 + \mathbf{c}_1 e^{-\mathbf{c}_2(\mathbf{u} - \mathbf{u}_f)}\right)^2} \right] (\mathbf{u} = \widehat{\mathbf{u}}(\mathbf{x})) \\ &\times \left[-\mathbf{K} - \frac{\partial \boldsymbol{\mathcal{N}}}{\partial \mathbf{x}}(\mathbf{x}_f) + \frac{\partial \boldsymbol{\mathcal{N}}}{\partial \mathbf{x}}(\mathbf{x}) \right]. \end{aligned} \tag{5.20}$$

Note that the multiplication and division of vectors inside the diagonal of the first matrix are to be understood as performed element-wise. Evaluating the linearization at $\mathbf{x} = \mathbf{x}_f$ we get

$$\begin{aligned} \frac{\partial \widehat{\mathbf{u}}}{\partial \mathbf{x}}(\mathbf{x}_f) = &\operatorname{diag}\left[\frac{\mathbf{c}_1 \mathbf{c}_2 (\mathbf{u}_{\max} - \mathbf{u}_{\min}) e^{-\mathbf{c}_2(\mathbf{u} - \mathbf{u}_f)}}{\left(\mathbf{1} + \mathbf{c}_1 e^{-\mathbf{c}_2(\mathbf{u} - \mathbf{u}_f)}\right)^2} \right] (\mathbf{u} = \mathbf{u}_f) \\ &\times \left[-\mathbf{K} - \frac{\partial \boldsymbol{\mathcal{N}}}{\partial \mathbf{x}}(\mathbf{x}_f) + \frac{\partial \boldsymbol{\mathcal{N}}}{\partial \mathbf{x}}(\mathbf{x}_f) \right] \\ = &\operatorname{diag}\left[\frac{\mathbf{c}_1 \mathbf{c}_2 (\mathbf{u}_{\max} - \mathbf{u}_{\min}) e^{\mathbf{0}}}{(\mathbf{1} + \mathbf{c}_1 e^{\mathbf{0}})^2} \right] [-\mathbf{K}]. \end{aligned}$$

Here we see that

$$\frac{\mathbf{c}_1\mathbf{c}_2\left(\mathbf{u}_{\max}-\mathbf{u}_{\min}\right)e^{\mathbf{0}}}{\left(\mathbf{1}+\mathbf{c}_1e^{\mathbf{0}}\right)^2}=\frac{\frac{\mathbf{u}_{\max}-\mathbf{u}_f}{\mathbf{u}_f-\mathbf{u}_{\min}}\frac{\mathbf{u}_{\max}-\mathbf{u}_{\min}}{\left(\mathbf{u}_{\max}-\mathbf{u}_f\right)\left(\mathbf{u}_f-\mathbf{u}_{\min}\right)}\left(\mathbf{u}_{\max}-\mathbf{u}_{\min}\right)}{\left(\mathbf{1}+\frac{\mathbf{u}_{\max}-\mathbf{u}_f}{\mathbf{u}_f-\mathbf{u}_{\min}}\right)^2}=\mathbf{1}.$$

Therefore

$$\frac{\partial\widehat{\mathbf{u}}}{\partial\mathbf{x}}\left(\mathbf{x}_f\right)=\mathbf{I}\left[-\mathbf{K}\right]=-\mathbf{K}.$$

Lastly the Jacobian of u_{mat}-*QRnet* is given by

$$\begin{aligned}\frac{\partial\widehat{\mathbf{u}}}{\partial\mathbf{x}}=&\operatorname{diag}\left[\frac{\mathbf{c}_1\mathbf{c}_2\left(\mathbf{u}_{\max}-\mathbf{u}_{\min}\right)e^{-\mathbf{c}_2\left(\mathbf{u}-\mathbf{u}_f\right)}}{\left(\mathbf{1}+\mathbf{c}_1e^{-\mathbf{c}_2\left(\mathbf{u}-\mathbf{u}_f\right)}\right)^2}\right]\left(\mathbf{u}=\widehat{\mathbf{u}}\left(\mathbf{x}\right)\right)\\&\times\left[-\mathbf{K}-\boldsymbol{\mathcal{N}}\left(\mathbf{x}\right)+\boldsymbol{\mathcal{N}}\left(\mathbf{x}_f\right)+\left[\frac{\partial\boldsymbol{\mathcal{N}}}{\partial\mathbf{x}}\left(\mathbf{x}\right)\right]\left(\mathbf{x}-\mathbf{x}_f\right)\right],\end{aligned}\tag{5.21}$$

where $\left[\frac{\partial\boldsymbol{\mathcal{N}}}{\partial\mathbf{x}}\left(\mathbf{x}\right)\right]\left(\mathbf{x}-\mathbf{x}_f\right):\mathbb{R}^n\to\mathbb{R}^{d\times n}$ is a tensor product. Evaluating the above linearization at $\mathbf{x}=\mathbf{x}_f$ we get

$$\begin{aligned}&\frac{\partial\widehat{\mathbf{u}}}{\partial\mathbf{x}}\left(\mathbf{x}_f\right)\\=&\operatorname{diag}\left[\frac{\mathbf{c}_1\mathbf{c}_2\left(\mathbf{u}_{\max}-\mathbf{u}_{\min}\right)e^{-\mathbf{c}_2\left(\mathbf{u}-\mathbf{u}_f\right)}}{\left(\mathbf{1}+\mathbf{c}_1e^{-\mathbf{c}_2\left(\mathbf{u}-\mathbf{u}_f\right)}\right)^2}\right]\left(\mathbf{u}=\mathbf{u}_f\right)\\&\times\left[-\mathbf{K}-\boldsymbol{\mathcal{N}}\left(\mathbf{x}_f\right)-\boldsymbol{\mathcal{N}}\left(\mathbf{x}_f\right)+\left[\frac{\partial\boldsymbol{\mathcal{N}}}{\partial\mathbf{x}}\left(\mathbf{x}_f\right)\right]\left(\mathbf{x}_f-\mathbf{x}_f\right)\right]\\=&\mathbf{I}\left[-\mathbf{K}\right]\\=&-\mathbf{K}.\end{aligned}$$

□

5.1.7 Universal approximation capacity

Local stability is a critical but bare minimum requirement. To achieve the ultimate goal of semi-global stability and optimality through training, the NN architectures must be able to approximate $\mathbf{u}^*(\cdot)$ with sufficient accuracy. Note that because $V_{\mathbf{x}}(\cdot)$ and $\mathbf{u}^*(\cdot)$ are not analytic in general, we cannot directly use the Taylor series-like structures of (5.5–5.10) to show this is possible. But for OCPs like (1.2) we expect $V_{\mathbf{x}}(\cdot)$ and $\mathbf{u}^*(\cdot)$ to be everywhere continuous and locally $\mathcal{C}^1$. In this case Theorems 2, 4 and 5 developed below will show that NNs of the form (5.5–5.10) are universal approximators for such functions.

If $V_{\mathbf{x}}(\cdot)$ and $\mathbf{u}^*(\cdot)$ are semi-globally $\mathcal{C}^1$, then existing NN universal approximation theorems such as Theorem 1 guarantee the existence of dense NNs with $\mathcal{C}^1$ activation functions which can approximate these functions. However, we cannot directly apply such results because the functions of interest are only *locally* $\mathcal{C}^1$ and the NN architectures used in this work are not standard.

Throughout this section let $\mathbb{X} \subset \mathbb{R}^n$ be compact with an open subset containing $\mathbf{x}_f$, and without loss of generality let $\mathbf{x}_f = \mathbf{0} \in \mathbb{X}$ and $\mathbf{u}_f = \mathbf{0}$. Let $\mathcal{C}(\mathbb{X})$ and $\mathcal{C}\left(\mathbb{X} \to \mathbb{R}^d\right)$ be the algebras[1] of continuous functions on $\mathbb{X}$ taking values in $\mathbb{R}$ and $\mathbb{R}^d$, respectively, where d is the dimension of the target variable (i.e. $d = m$ or $d = n$). In the statement of Theorems 2, 4 and 5 we consider target functions $\mathbf{f} \in \mathcal{C}\left(\mathbb{X} \to \mathbb{R}^d\right)$ which could represent the value gradient or control (i.e. $\mathbf{f}(\mathbf{x}) = V_{\mathbf{x}}(\mathbf{x})$ or $\mathbf{f}(\mathbf{x}) = \mathbf{u}^*(\mathbf{x})$).

Remark 4. *The proofs in this section do not deal with saturation constraints* (2.18) *which may be applied to the control models,* u-QRnet*,* u_{Jac}-QRnet*, and* u_{mat}-

[1] We call a set of functions $\mathcal{A}$ an algebra if it is closed under (element-wise) addition, multiplication, and scalar multiplication. A subalgebra of $\mathcal{A}$ is a subset of $\mathcal{A}$ which is also an algebra.

QRnet. *In practice we do not see a problem with learning in control-constrained problems, and the smooth saturation function* (2.18) *is very nearly identity for most controls* $\mathbf{u} \in \mathbb{U}$. *Furthermore,* (2.18) *is invertible on the open interior of* $\mathbb{U}$, *so we should be able to handle the constrained case by applying this inverse transformation to the target function,* $\mathbf{u}^*(\mathbf{x})$.

5.1.7.1 λ-*QRnet* and u-*QRnet*

Let us first verify that the basic λ-*QRnet* and u-*QRnet* architectures recover universal approximation. This may be relatively obvious but we provide a proof for completeness.

Theorem 2 (*QRnet* approximation)**.** *Suppose* $\mathbf{f} \in \mathcal{C}\left(\mathbb{X} \to \mathbb{R}^d\right)$, $\mathbf{f}(\mathbf{0}) = 0$, *and* $\mathbf{f}(\cdot)$ *is* $\mathcal{C}^1$ *in a neighborhood of* $\mathbf{0}$. *Then for all* $\varepsilon > 0$, *there exists a feedforward NN with* $\mathcal{C}^1$ *bounded, non-constant activation functions,* $\boldsymbol{\mathcal{N}} \in \mathcal{C}^1\left(\mathbb{X} \to \mathbb{R}^d\right)$, *such that for all* $\mathbf{x} \in \mathbb{X}$,

$$\left\| \mathbf{f}(\mathbf{x}) - \left(\left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{0}) \right] \mathbf{x} + \boldsymbol{\mathcal{N}}(\mathbf{x}) - \boldsymbol{\mathcal{N}}(\mathbf{0}) \right) \right\|_1 < \varepsilon. \tag{5.22}$$

Proof. Let $\mathbf{g}(\mathbf{x}) = \mathbf{f}(\mathbf{x}) - \left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{0})\right] \mathbf{x}$. Now by Theorem 1 there exists an NN, $\boldsymbol{\mathcal{N}} \in \mathcal{C}^1\left(\mathbb{X} \to \mathbb{R}^d\right)$, which satisfies

$$\max_{\mathbf{x} \in \mathbb{X}} \|\mathbf{g}(\mathbf{x}) - \boldsymbol{\mathcal{N}}(\mathbf{x})\|_1 < \frac{\varepsilon}{2}.$$

This holds for all $\mathbf{x} \in \mathbb{X}$ including $\mathbf{x} = \mathbf{0}$, and so using the observation that $\mathbf{g}(\mathbf{0}) = \mathbf{0}$, we must have

$$\|\boldsymbol{\mathcal{N}}(\mathbf{0})\|_1 = \|\mathbf{0} - \boldsymbol{\mathcal{N}}(\mathbf{0})\|_1 = \|\mathbf{g}(\mathbf{0}) - \boldsymbol{\mathcal{N}}(\mathbf{0})\|_1 < \frac{\varepsilon}{2},$$

Consequently, for all $\mathbf{x} \in \mathbb{X}$ we get

$$\|\mathbf{g}(\mathbf{x}) - \boldsymbol{\mathcal{N}}(\mathbf{x}) - \boldsymbol{\mathcal{N}}(\mathbf{0})\|_1 \leq \|\mathbf{g}(\mathbf{x}) - \boldsymbol{\mathcal{N}}(\mathbf{x})\|_1 + \|\boldsymbol{\mathcal{N}}(\mathbf{0})\|_1 < \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon.$$

This completes the proof. □

5.1.7.2 "Jacobian" and "matrix" *QRnet*s

To prove Theorems 4 and 5 for the "Jacobian" and "matrix" architectures, we will first specialize the Stone-Weierstrass approximation theorem [132] to locally $\mathcal{C}^1$ functions, and then apply Theorem 1. Let us first review the classic Stone-Weierstrass theorem for context.

Theorem 3 (Stone-Weierstrass [132])**.** *Suppose that $\mathcal{A}$ is a subalgebra of $\mathcal{C}(\mathbb{X})$ which separates points*[2] *and does not vanish*[3] *anywhere in $\mathbb{X}$. Then for all $f \in \mathcal{C}(\mathbb{X})$ and all $\varepsilon > 0$ there exists $g \in \mathcal{A}$ satisfying $\max_{\mathbf{x}\in\mathbb{X}} |f(\mathbf{x}) - g(\mathbf{x})| < \varepsilon$.*

For the present problem we consider a special case of Theorem 3 where we want to approximate a locally $\mathcal{C}^1$ functions and their Jacobian at origin.

Corollary 1 (Stone-Weierstrass for locally $\mathcal{C}^1$ functions)**.** *Suppose $\mathbf{f} \in \mathcal{C}\left(\mathbb{X} \to \mathbb{R}^d\right)$, $f(\mathbf{0}) = \mathbf{0}$, and $\mathbf{f}(\cdot)$ is $\mathcal{C}^1$ in a neighborhood of $\mathbf{0}$. Then for all $\varepsilon > 0$, there exists a function $\mathbf{g} \in \mathcal{C}^1\left(\mathbb{X} \to \mathbb{R}^d\right)$ satisfying $\mathbf{g}(\mathbf{0}) = \mathbf{0}$, $\frac{\partial \mathbf{g}}{\partial \mathbf{x}}(\mathbf{0}) = \mathbf{0}$, and*

$$\max_{\mathbf{x}\in\mathbb{X}} \left\| \mathbf{f}(\mathbf{x}) - \left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{0})\right] \mathbf{x} - \mathbf{g}(\mathbf{x}) \right\|_1 < \varepsilon. \tag{5.23}$$

Proof. First consider the set of functions $\mathcal{A} \subset \mathcal{C}^1(\mathbb{X})$ which have $\frac{\partial g}{\partial \mathbf{x}}(\mathbf{0}) = \mathbf{0}$. We claim that $\mathcal{A}$ is an algebra which vanishes nowhere and separates points.

[2] An algebra $\mathcal{A}$ separates points if for all $\mathbf{x}, \mathbf{y} \in \mathbb{X}$, $\mathbf{x} \neq \mathbf{y}$, there exists $g \in \mathcal{A}$ such that $g(\mathbf{x}) \neq g(\mathbf{y})$.

[3] A set of functions $\mathcal{A}$ vanishes at $\mathbf{x}_1 \in \mathbb{X}$ if $\mathbf{f}(\mathbf{x}_1) = \mathbf{0}$ for all $\mathbf{f} \in \mathcal{A}$.

It is easy to verify that $\mathcal{A}$ is closed under addition, multiplication, and scalar multiplication; hence $\mathcal{A}$ is an algebra. $\mathcal{A}$ also contains the constant functions, so it vanishes nowhere. Lastly, to see that $\mathcal{A}$ separates points, note that for any $\mathbf{x} \neq \mathbf{y}$ without loss of generality $x_1 \neq y_1$. Then take $g(\mathbf{x}) = x_1^3 \neq y_1^3 = g(\mathbf{y})$, since x^3 is one-to-one.

Now write $\mathbf{f}(\mathbf{x}) = \begin{pmatrix} f_1(\mathbf{x}), & \dots, & f_d(\mathbf{x}) \end{pmatrix}^T$. For each $i = 1, \dots, d$, by Theorem 3 we can find a function $h_i \in \mathcal{A}$ which satisfies

$$\max_{\mathbf{x} \in \mathbb{X}} \left| f_i(\mathbf{x}) - \left[\frac{\partial f_i}{\partial \mathbf{x}}(\mathbf{0}) \right] \mathbf{x} - h_i(\mathbf{x}) \right| < \frac{\varepsilon}{2d}.$$

Since $f_i(\mathbf{0}) = 0$ by assumption, this implies

$$|h_i(\mathbf{0})| = \left| f_i(\mathbf{0}) - \left[\frac{\partial f_i}{\partial \mathbf{x}}(\mathbf{0}) \right] \mathbf{0} - h_i(\mathbf{0}) \right| < \frac{\varepsilon}{2d}.$$

Defining $g_i(\mathbf{x}) \coloneqq h_i(\mathbf{x}) - h_i(\mathbf{0})$ we get $g_i(\mathbf{0}) = 0$ and

$$\begin{aligned} &\max_{\mathbf{x} \in \mathbb{X}} \left| f_i(\mathbf{x}) - \left[\frac{\partial f_i}{\partial \mathbf{x}}(\mathbf{0}) \right] \mathbf{x} - g_i(\mathbf{x}) \right| \\ \leq &\max_{\mathbf{x} \in \mathbb{X}} \left| f_i(\mathbf{x}) - \left[\frac{\partial f_i}{\partial \mathbf{x}}(\mathbf{0}) \right] \mathbf{x} - h_i(\mathbf{x}) \right| + |h_i(\mathbf{0})| \\ < &\frac{\varepsilon}{2d} + \frac{\varepsilon}{2d} = \frac{\varepsilon}{d}. \end{aligned}$$

Because A is an algebra we also have $g_i \in \mathcal{A}$ and hence $\frac{\partial g_i}{\partial \mathbf{x}}(\mathbf{0}) = \mathbf{0}$. Thus setting $\mathbf{g}(\mathbf{x}) = \begin{pmatrix} g_1(\mathbf{x}), & \dots, & g_d(\mathbf{x}) \end{pmatrix}^T$ yields the desired function. □

We are now ready to state the first main result concerning the approximation capacity of the "Jacobian" *QRnet* architectures introduced in Section 5.1.4. As mentioned previously, since we expect the value gradient and optimal control for the OCP (1.4) to be continuous and locally $\mathcal{C}^1$, this supports the use of λ_{Jac}-*QRnet*

and u_{Jac}-*QRnet* as function approximators in this context.

Theorem 4 (Jacobian *QRnet* approximation)**.** *Suppose* $\mathbf{f} \in \mathcal{C}\left(\mathbb{X} \to \mathbb{R}^d\right)$, $\mathbf{f}(\mathbf{0}) = 0$, *and* $\mathbf{f}(\cdot)$ *is* $\mathcal{C}^1$ *in a neighborhood of* $\mathbf{0}$. *Then for all* $\varepsilon > 0$, *there exists a feedforward NN with* $\mathcal{C}^1$ *bounded, non-constant activation functions,* $\boldsymbol{\mathcal{N}} \in \mathcal{C}^1\left(\mathbb{X} \to \mathbb{R}^d\right)$, *such that for all* $\mathbf{x} \in \mathbb{X}$,

$$\left\| \mathbf{f}(\mathbf{x}) - \left(\left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{0}) - \frac{\partial \boldsymbol{\mathcal{N}}}{\partial \mathbf{x}}(\mathbf{0}) \right] \mathbf{x} + \boldsymbol{\mathcal{N}}(\mathbf{x}) - \boldsymbol{\mathcal{N}}(\mathbf{0}) \right) \right\|_1 < \varepsilon. \tag{5.24}$$

Proof. Since $\mathbb{X}$ is bounded, there is some $B > 0$ for which $\max_{\mathbf{x} \in \mathbb{X}} \|\mathbf{x}\|_1 \leq B$. For any $\varepsilon > 0$, define $\varepsilon^* \coloneqq \min\{\varepsilon, \varepsilon/B\}$. From Corollary 1 we can find some $\mathbf{g} \in \mathcal{C}^1\left(\mathbb{X} \to \mathbb{R}^d\right)$ satisfying $\mathbf{g}(\mathbf{0}) = \mathbf{0}$, $\frac{\partial \mathbf{g}}{\partial \mathbf{x}}(\mathbf{0}) = \mathbf{0}$, and

$$\max_{\mathbf{x} \in \mathbb{X}} \left\| \mathbf{f}(\mathbf{x}) - \left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{0}) \right] \mathbf{x} - \mathbf{g}(\mathbf{x}) \right\|_1 < \frac{\varepsilon^*}{2}.$$

By Theorem 1 there exists an NN, $\boldsymbol{\mathcal{N}} \in \mathcal{C}^1\left(\mathbb{X} \to \mathbb{R}^d\right)$, which approximates $\mathbf{g}(\cdot)$ and its derivative to arbitrary accuracy, say

$$\max \left\{ \begin{array}{l} \max_{\mathbf{x} \in \mathbb{X}} \|\mathbf{g}(\mathbf{x}) - \boldsymbol{\mathcal{N}}(\mathbf{x})\|_1, \\ \max_{\mathbf{x} \in \mathbb{X}} \left\| \frac{\partial \mathbf{g}}{\partial \mathbf{x}}(\mathbf{x}) - \frac{\partial \boldsymbol{\mathcal{N}}}{\partial \mathbf{x}}(\mathbf{x}) \right\|_{1,1} \end{array} \right\} < \frac{\varepsilon^*}{6}.$$

Here we define the matrix norm $\|\mathbf{A}\|_{1,1} \coloneqq \|\text{vec}(\mathbf{A})\|_1$ for a matrix $\mathbf{A} \in \mathbb{R}^{d \times n}$ and its vectorization, $\text{vec}(\mathbf{A}) \in \mathbb{R}^{mn}$. Notice that

$$\|\boldsymbol{\mathcal{N}}(\mathbf{0})\|_1 = \|\mathbf{0} - \boldsymbol{\mathcal{N}}(\mathbf{0})\|_1 = \|\mathbf{g}(\mathbf{0}) - \boldsymbol{\mathcal{N}}(\mathbf{0})\|_1 < \frac{\varepsilon^*}{6}.$$

Consequently, for all $\mathbf{x} \in \mathbb{X}$ we get

$$\|\mathbf{g}(\mathbf{x}) - \boldsymbol{\mathcal{N}}(\mathbf{x}) - \boldsymbol{\mathcal{N}}(\mathbf{0})\|_1 \leq \|\mathbf{g}(\mathbf{x}) - \boldsymbol{\mathcal{N}}(\mathbf{x})\|_1 + \|\boldsymbol{\mathcal{N}}(\mathbf{0})\|_1 < \frac{\varepsilon^*}{3}.$$

Similarly,

$$\left\|\frac{\partial \mathbf{g}}{\partial \mathbf{x}}(\mathbf{0}) - \frac{\partial \boldsymbol{\mathcal{N}}}{\partial \mathbf{x}}(\mathbf{0})\right\|_{1,1} = \left\|\frac{\partial \boldsymbol{\mathcal{N}}}{\partial \mathbf{x}}(\mathbf{0})\right\|_{1,1} < \frac{\varepsilon^*}{6},$$

which implies

$$\left\|\left[\frac{\partial \boldsymbol{\mathcal{N}}}{\partial \mathbf{x}}(\mathbf{0})\right]\mathbf{x}\right\|_1 \leq \left\|\frac{\partial \boldsymbol{\mathcal{N}}}{\partial \mathbf{x}}(\mathbf{0})\right\|_{1,1} \|\mathbf{x}\|_1 < \frac{\varepsilon^*}{6} B,$$

for all $\mathbf{x} \in \mathbb{X}$. Putting this all together we obtain

$$\begin{aligned}
&\left\|\mathbf{f}(\mathbf{x}) - \left(\left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{0}) - \frac{\partial \boldsymbol{\mathcal{N}}}{\partial \mathbf{x}}(\mathbf{0})\right]\mathbf{x} + \boldsymbol{\mathcal{N}}(\mathbf{x}) - \boldsymbol{\mathcal{N}}(\mathbf{0})\right)\right\|_1 \\
\leq &\left\|\mathbf{f}(\mathbf{x}) - \left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{0})\right]\mathbf{x} - \mathbf{g}(\mathbf{x})\right\|_1 + \|\mathbf{g}(\mathbf{x}) - \boldsymbol{\mathcal{N}}(\mathbf{x}) - \boldsymbol{\mathcal{N}}(\mathbf{0})\|_1 + \left\|\left[\frac{\partial \boldsymbol{\mathcal{N}}}{\partial \mathbf{x}}(\mathbf{0})\right]\mathbf{x}\right\|_1 \\
< &\frac{\varepsilon^*}{2} + \frac{\varepsilon^*}{3} + \frac{\varepsilon^*}{6} B \\
\leq &\frac{\varepsilon}{2} + \frac{\varepsilon}{3} + \frac{\varepsilon}{6} \\
= &\varepsilon,
\end{aligned}$$

for all $\mathbf{x} \in \mathbb{X}$, as desired. □

An analogous approximation theorem can be obtained for the "matrix" *QRnet* architectures introduced in Section 5.1.5, λ_{mat}-*QRnet* and u_{mat}-*QRnet*.

Theorem 5 (Matrix *QRnet* approximation)**.** *Suppose* $\mathbf{f} \in \mathcal{C}\left(\mathbb{X} \to \mathbb{R}^d\right)$, $\mathbf{f}(\mathbf{0}) = \mathbf{0}$, *and* $\mathbf{f}(\cdot)$ *is* $\mathcal{C}^1$ *in a neighborhood of* $\mathbf{0}$*. Then for all* $\varepsilon > 0$*, there exists a feedforward NN with* $\mathcal{C}^1$ *bounded, non-constant activation functions,* $\boldsymbol{\mathcal{N}} \in \mathcal{C}^1\left(\mathbb{X} \to \mathbb{R}^{d \times n}\right)$*, such that for all* $\mathbf{x} \in \mathbb{X}$*,*

$$\left\|\mathbf{f}(\mathbf{x}) - \left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{0}) + \boldsymbol{\mathcal{N}}(\mathbf{x}) - \boldsymbol{\mathcal{N}}(\mathbf{0})\right]\mathbf{x}\right\| < \varepsilon. \tag{5.25}$$

Proof. From Corollary 1 we can find some $\mathbf{g} \in \mathcal{C}^1\left(\mathbb{X} \to \mathbb{R}^d\right)$ satisfying $\mathbf{g}(\mathbf{0}) = \mathbf{0}$,

$\frac{\partial \mathbf{g}}{\partial \mathbf{x}}(\mathbf{0}) = \mathbf{0}$, and

$$\max_{\mathbf{x}\in\mathbb{X}} \left\| \mathbf{f}(\mathbf{x}) - \left[\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{0})\right] \mathbf{x} - \mathbf{g}(\mathbf{x}) \right\|_1 < \frac{\varepsilon}{2}. \tag{5.26}$$

Applying [71, Exercise 3.23], for all $\mathbf{x} \in \mathbb{X}$ we can decompose $\mathbf{g}(\mathbf{x}) = [\mathbf{h}(\mathbf{x})]\,\mathbf{x}$, where $\mathbf{h} \in \mathcal{C}\left(\mathbb{X} \to \mathbb{R}^{d\times n}\right)$ is given by $\mathbf{h}(\mathbf{x}) = \int_0^1 \frac{\partial \mathbf{g}}{\partial \mathbf{x}}(s\mathbf{x})ds$. Further, since $\frac{\partial \mathbf{g}}{\partial \mathbf{x}}(\mathbf{0}) = \mathbf{0}$ we have $\mathbf{h}(\mathbf{0}) = \int_0^1 \frac{\partial \mathbf{g}}{\partial \mathbf{x}}(\mathbf{0})ds = \mathbf{0}$.

Since $\mathbb{X}$ is bounded, there is some $B > 0$ for which $\max_{\mathbf{x}\in\mathbb{X}} \|\mathbf{x}\|_1 \leq B$. Now given $\varepsilon > 0$, by Theorem 1 there exists an NN, $\boldsymbol{\mathcal{N}} \in \mathcal{C}^1\left(\mathbb{X} \to \mathbb{R}^{d\times n}\right)$, with $\max_{\mathbf{x}\in\mathbb{X}} \|\mathbf{h}(\mathbf{x}) - \boldsymbol{\mathcal{N}}(\mathbf{x})\|_{1,1} < \varepsilon/(4B)$. In particular,

$$\|\boldsymbol{\mathcal{N}}(\mathbf{0})\|_{1,1} = \|\mathbf{0} - \boldsymbol{\mathcal{N}}(\mathbf{0})\|_{1,1} = \|\mathbf{h}(\mathbf{0}) - \boldsymbol{\mathcal{N}}(\mathbf{0})\|_{1,1} < \frac{\varepsilon}{4B}.$$

Therefore, for all $\mathbf{x} \in \mathbb{X}$ we get

$$\begin{aligned} \|\mathbf{g}(\mathbf{x}) - [\boldsymbol{\mathcal{N}}(\mathbf{x}) - \boldsymbol{\mathcal{N}}(\mathbf{0})]\,\mathbf{x}\|_1 &= \|[\mathbf{h}(\mathbf{x}) - \boldsymbol{\mathcal{N}}(\mathbf{x}) + \boldsymbol{\mathcal{N}}(\mathbf{0})]\,\mathbf{x}\|_1 \\ &\leq \left(\|\mathbf{h}(\mathbf{x}) - \boldsymbol{\mathcal{N}}(\mathbf{x})\|_{1,1} + \|\boldsymbol{\mathcal{N}}(\mathbf{0})\|_{1,1}\right) \|\mathbf{x}\|_1 \\ &< \left(\frac{\varepsilon}{4B} + \frac{\varepsilon}{4B}\right) B \\ &= \frac{\varepsilon}{2}. \end{aligned} \tag{5.27}$$

Applying the triangle inequality to (5.26) and (5.27) finishes the proof. □

5.2 Example: Revisiting the Burgers' PDE

Here we revisit the unstable Burgers' PDE example from Section 4.1. The discretized OCP (4.6) in $n = 64$ dimensions is restated here for convenience:

$$\begin{cases} \underset{\mathbf{u}(\cdot)}{\text{minimize}} & \mathcal{J}_n\left[\mathbf{u}(\cdot)\right] = \displaystyle\int_0^\infty \left(\|\mathbf{x}\|_{\mathbf{Q}}^2 + R\,\mathbf{u}^T\mathbf{u}\right) dt, \\ \text{subject to} & \dot{\mathbf{x}} = -\dfrac{1}{2}\mathbf{D}\left(\mathbf{x}\odot\mathbf{x}\right) + \nu\mathbf{D}^2\mathbf{x} + \boldsymbol{\alpha}\odot\mathbf{x}\odot e^{-\beta\mathbf{x}} + \mathbf{B}\mathbf{u}. \end{cases}$$

Previously we saw that with enough training data we could get good results with the standard NN architectures, V-NN, λ-NN, and u-NN. But these results were inconsistent, and it was hard to predict if a controller would be even locally stabilizing. In this section we will apply the novel *QRnet* architectures to the same problem with the same learning conditions. What we find is that the new models do just as well at *approximation*, but even the lowest-fidelity *QRnet*s are locally and even semi-globally stabilizing. This added robustness means that their performance improves along with their approximation accuracy, instead of being contingent on stability.

5.2.1 Learning results and local stability verification

Let us first see how the models perform in terms of approximation accuracy. Since there are so many architectures it becomes incomprehensible to view all the architectures at once, so in this section we divide plots into two groups: the first contains value function and value gradient models, and other contains optimal control models.

Figure 5.2 shows training times for each of the NN types. We immediately observe that optimal control models are the fastest to train, while λ_{Jac}-*QRnet*

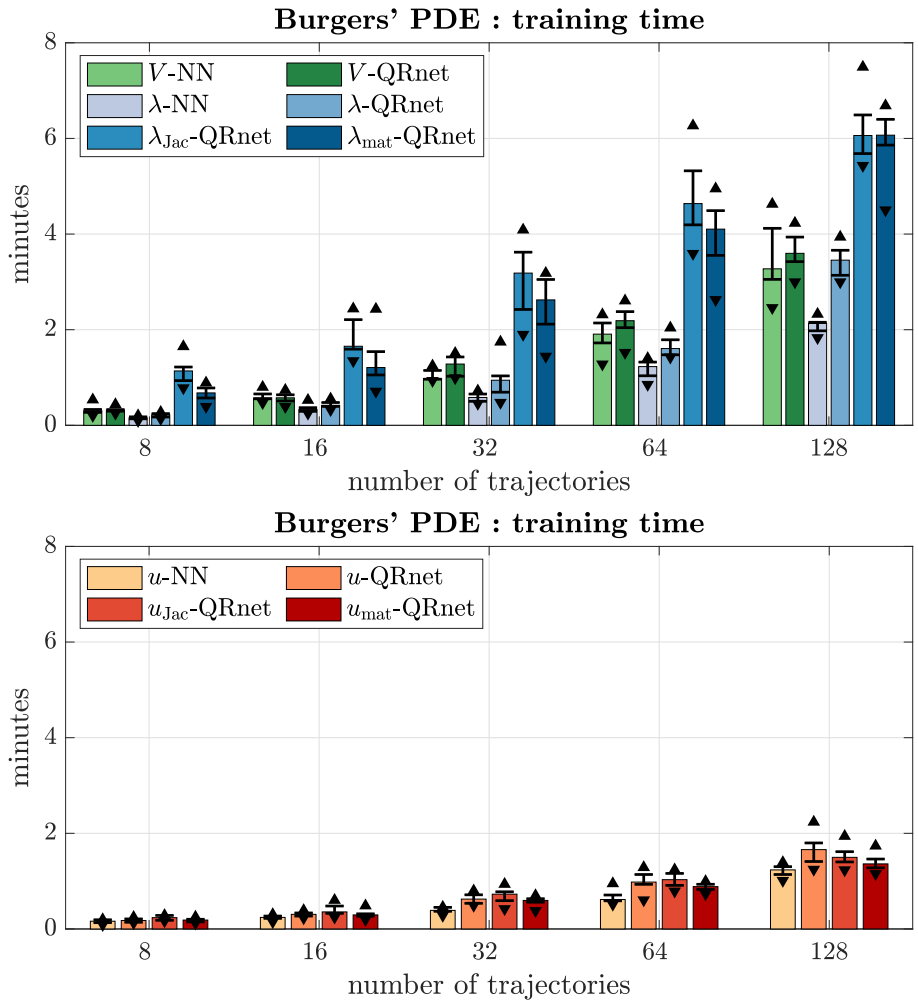


Figure 5.2: Training time for the infinite horizon Burgers' OCP (4.6), depending on the amount of training data. Top: value function and value gradient models. Bottom: optimal control models.

and λ_{mat}-*QRnet* take the longest. λ_{Jac}-*QRnet* takes a long time because of the high-dimensional Jacobians while λ_{mat}-*QRnet* takes long simply because of the large number of NN parameters. Note that because of L-BFGS's stopping criteria these results may differ depending on how difficult each model is to train for a specific problem. NN in-the-loop control evaluation times reported in Table 5.1 confirm that the new architectures are very nearly as fast to evaluate as standard NNs.

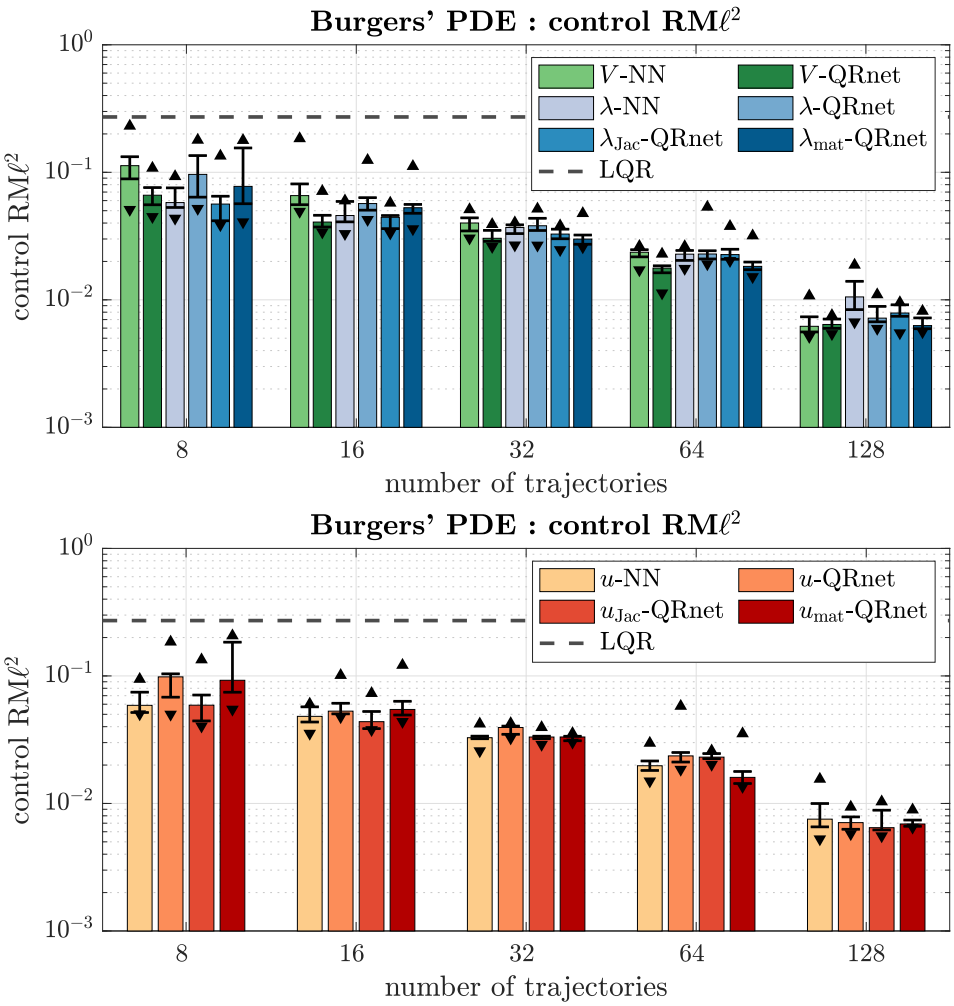


Figure 5.3: Average control prediction error for the infinite horizon Burgers' OCP (4.6), depending on the amount of training data. Top: value function and value gradient models. Bottom: optimal control models.

Figure 5.3 shows that the new architectures have similar test accuracy statistics to the standard NNs, confirming that they can learn complicated nonlinear functions as suggested by Theorems 2, 4 and 5. For this problem there is no clear performance distinction between the "Jacobian" and "matrix" architectures or between value gradient and control models.

Next let us verify that the new architectures are LAS. For all of these models but *V-QRnet*, the equilibrium is exactly $\bar{\mathbf{x}} = \mathbf{x}_f = \mathbf{0}$ as expected. As in Sec-

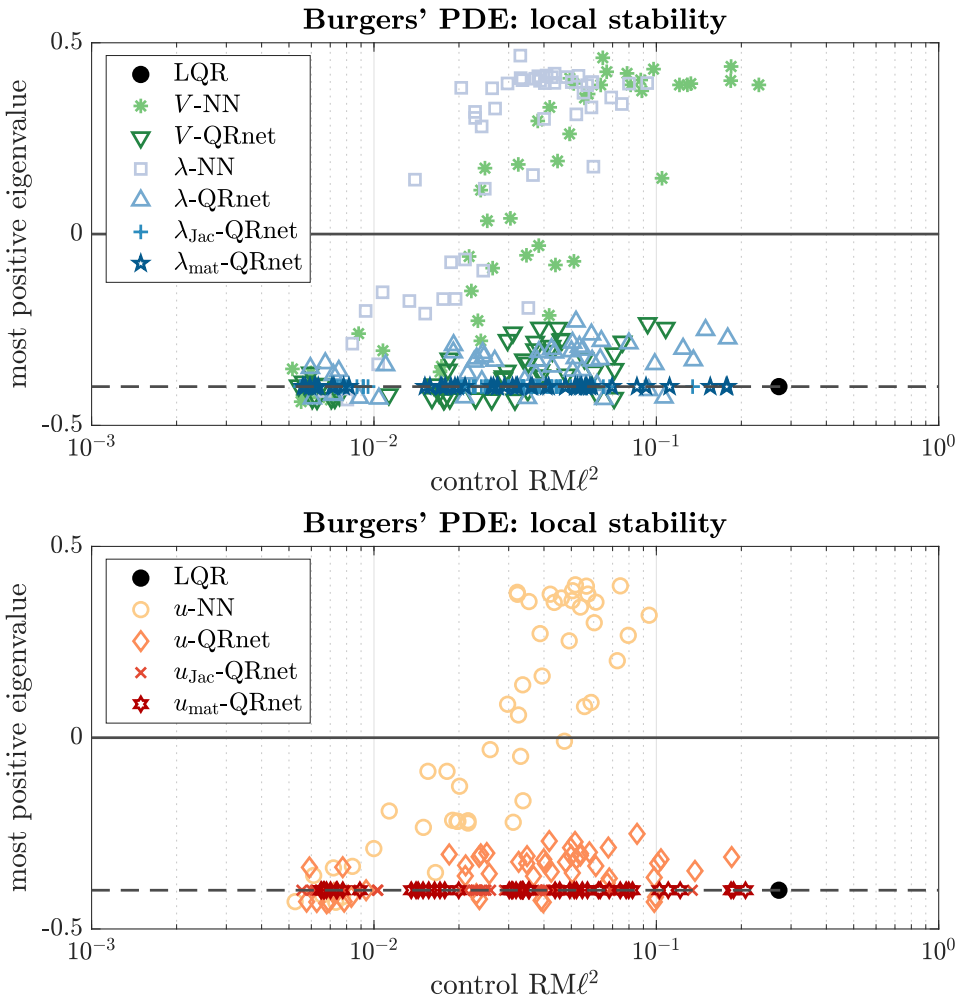


Figure 5.4: Real part of most positive closed loop Jacobian eigenvalue at an equilibrium $\bar{\mathbf{x}}$ near $\mathbf{x}_f$. Top: value function and value gradient models. Bottom: optimal control models.

tion 4.1.3 we compute the eigenvalues of the closed loop Jacobian, $\mathbf{A}_{\text{cl}}$, for each NN. Figure 5.4 shows the real part of the most positive eigenvalues for each NN. While the standard NNs must be trained to a high level of test accuracy before they are even LAS, all of the *QRnet* controllers are LAS even when trained on small data sets. Recall that Proposition 3 guarantees this for "Jacobian" and "matrix" *QRnet*s, and indeed the largest Jacobian eigenvalues are precisely those of LQR.

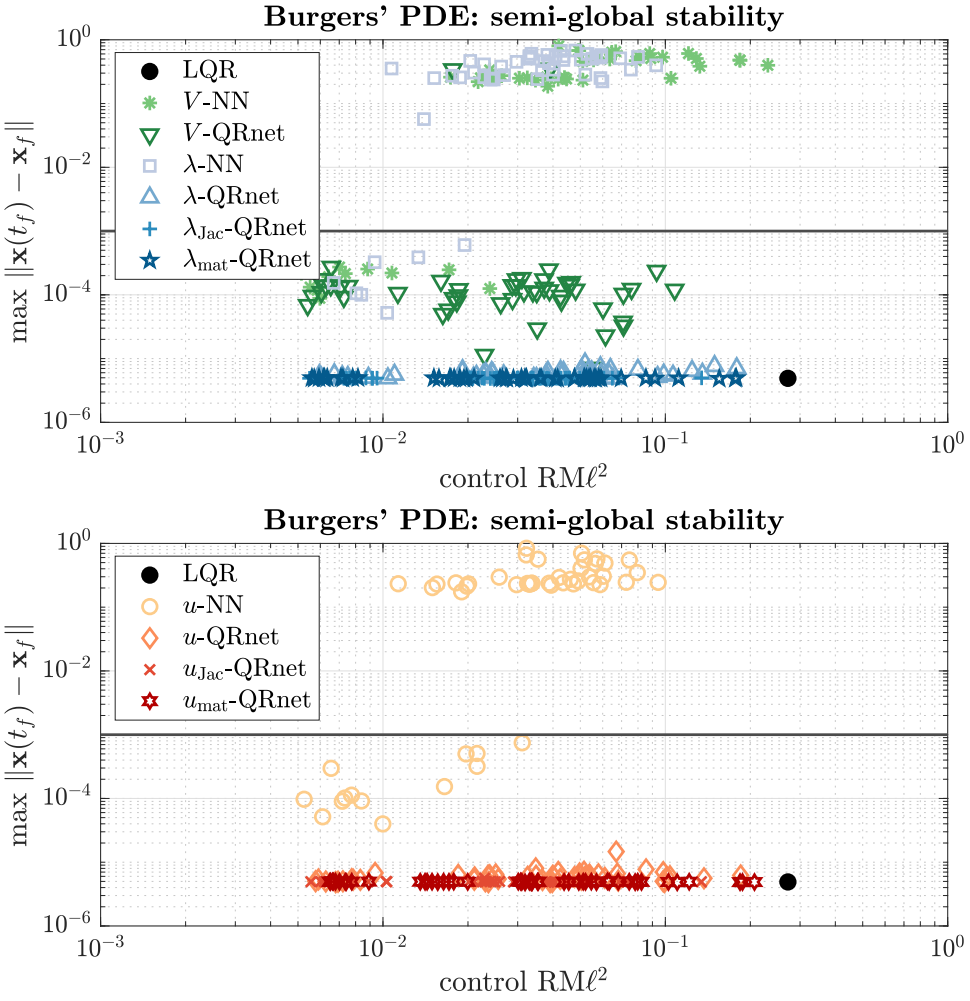


Figure 5.5: Worst case norm of final state over $N_{\text{MC}} = 100$ simulations. Top: value function and value gradient models. Bottom: optimal control models.

5.2.2 Monte Carlo stability and optimality analysis

Next we repeat the MC simulations conducted in Sections 4.1.4 and 4.1.5 with the new controllers. Figure 5.5 shows the worst-case failures for each controller and Figure 5.6 shows the fraction of stabilized runs. While only the most accurate standard NNs stabilize the origin, *all* of the *QRnet* controllers except two *V-QRnet*s stabilized every single initial condition tested. These empirical results suggest that the proposed architectures not only guarantee LAS, but also make

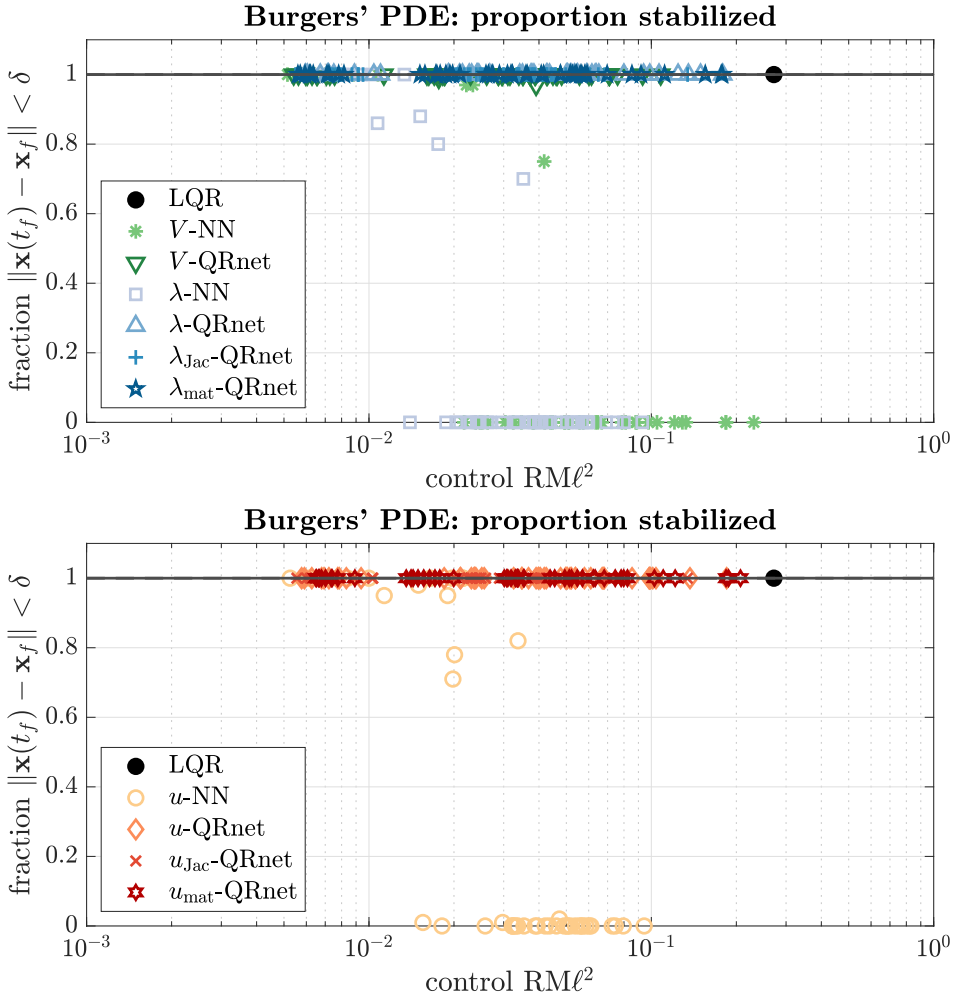


Figure 5.6: Fraction of stabilized initial conditons for $N_{\text{MC}} = 100$ simulations. Top: value function and value gradient models. Bottom: optimal control models.

the control design process more reliable, consistently yielding a stabilizing control law even with small data sets and short training times.

Figure 5.7 shows the results of the optimality calculations for the new models. For this problem, even the most poorly trained *QRnet* performed better than LQR on average. Furthermore, the *QRnet* controllers average performance improved together with approximation accuracy, following the same trend as the standard NNs. These experimental results indicate that the proposed architec-

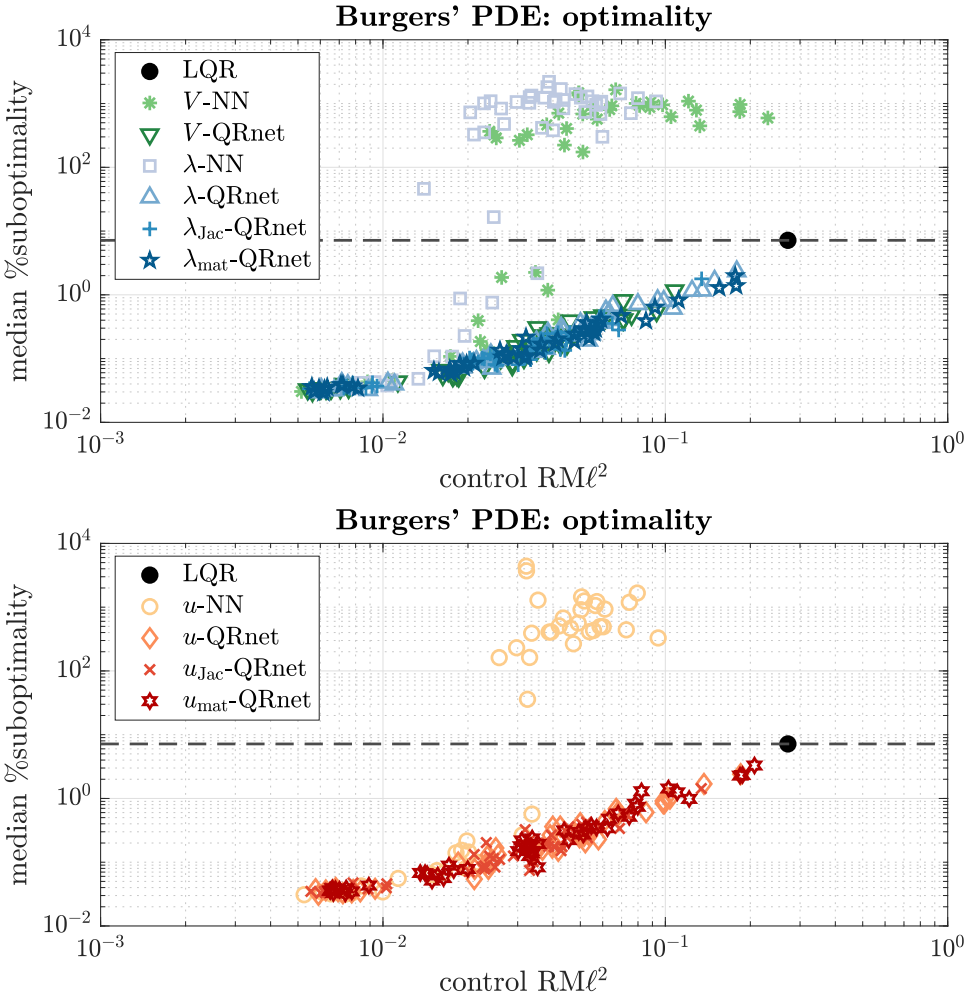


Figure 5.7: Median percent cost more than optimal cost over $N_{\text{MC}} = 100$ simulations. Top: value function and value gradient models. Bottom: optimal control models.

tures improve stability without limiting optimality. In a control design context this makes the method more reliable: we should be able to expect that results improve *consistently* as we add more data, which was not always the case for standard NNs.

We conclude the section with Figure 5.8 which shows the result of one more closed loop simulation with the same initial condition as in Figures 4.1 and 4.7. Here we use a u_{Jac}*QRnet* also trained on $N_{\text{OCP, train}}$. Its control RMℓ^2 is 0.0229

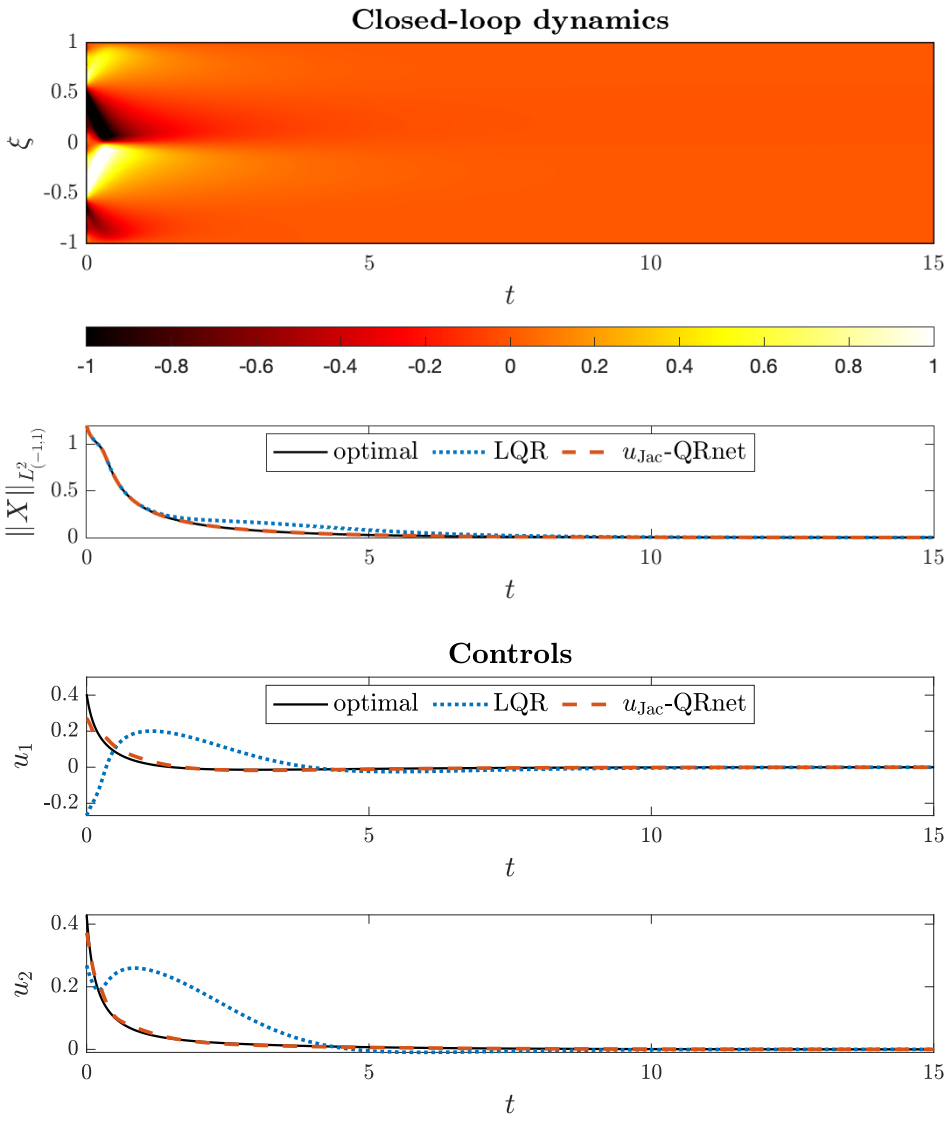


Figure 5.8: Closed loop simulation of the unstable Burgers' PDE (4.6). Feedback control is based on a u_{Jac} *QRnet* trained on the same number of data as the controller shown in Figures 4.1 and 4.7.

which is about as accurate as the *unstable* V-NN used for Figure 4.1, yet it is only 0.20% suboptimal for this trajectory – slightly better than the more accurate V-NN used for Figure 4.7. The precise numbers are of course not very meaningful; our intent here is only to show that the new architectures can reliably deliver high quality feedback control for large nonlinear dynamical systems without noticeable

tradeoffs.

5.3 Application example: Fixed-wing UAV

In this section we illustrate how the proposed control archictures can be used with supervised learning to design an optimal feedback controller a fixed-wing 6DoF UAV. The controller is designed for stabilization from a wide range of flight conditions, as well as tracking for arbitrary altitude and course commands. This is a challenging nonlinear OCP.

For this problem we found that indirect methods were unreliable for generating data, even when given an initial guess from a solution obtained with direct method (see Chapter 3). For this reason we generate data with an LGR PS method. To the best of our knowledge this is the first case of PS methods being used for supervised learning.

To ensure that the open loop OCP data set is of good quality we use a large number of LGR collocation points and set stringent tolerances for SQP [74], our nonlinear programming solver. Unfortunately, even though the state and control data appear accurate the resulting costate data is not accurate enough for reliable learning. For this reason in this section we only show results for u-NN, u-*QRnet*, u_{Jac}-*QRnet*, and u_{mat}-*QRnet*, which directly approximate the optimal control and do not need costate data.

5.3.1 Fixed-wing UAV dynamics

The dynamic model we use is based on the one presented in [7, 8]. We review it here to orient the reader and point out several small differences.

The position of the UAV is described in inertial north-east-down coordinates,

$\mathbf{p} \coloneqq \begin{pmatrix} p_n, & p_e, & p_d \end{pmatrix}^T$. Here p_n denotes the downrange position, p_e denotes cross-range, and $h = -p_d$ is altitude. The velocities in the body x, y, and z directions are denoted as $\mathbf{V} \coloneqq \begin{pmatrix} u, & v, & w \end{pmatrix}^T$. The attitude of the UAV, i.e. its rotation from inertial to body frames, is described using quaternions $\mathbf{q} \coloneqq \begin{pmatrix} q_0, & \bar{\mathbf{q}}^T \end{pmatrix}^T$, where q_0 is the scalar quaternion and $\bar{\mathbf{q}} \coloneqq \begin{pmatrix} q_1, & q_2, & q_3 \end{pmatrix}^T$ is the vector quaternion. The angular velocity of UAV in the body frame is written as $\boldsymbol{\omega} \coloneqq \begin{pmatrix} p, & q, & r \end{pmatrix}^T$. The full state is then

$$\mathbf{x} \coloneqq \begin{pmatrix} \mathbf{p}^T, & \mathbf{V}^T, & \mathbf{q}^T, & \boldsymbol{\omega}^T \end{pmatrix}^T \in \mathbb{R}^{13}. \tag{5.28}$$

The UAV is controlled with a throttle $\delta_t \in [0, 1]$, ailerons $\delta_a \in [-\delta_a^+, \delta_a^+]$, elevator $\delta_e \in [-\delta_e^+, \delta_e^+]$, and rudder $\delta_r \in [-\delta_r^+, \delta_r^+]$. Thus

$$\mathbf{u} \coloneqq \begin{pmatrix} \delta_t, & \delta_a, & \delta_e, & \delta_r \end{pmatrix}^T \in \mathbb{U} \subset \mathbb{R}^4. \tag{5.29}$$

Modeling the UAV as a rigid body we obtain the dynamic equations [7]

$$\dot{\mathbf{x}} = \begin{pmatrix} \dot{\mathbf{p}} \\ \dot{\mathbf{V}} \\ \dot{\mathbf{q}} \\ \dot{\boldsymbol{\omega}} \end{pmatrix} = \begin{pmatrix} \mathcal{R}_{\mathbf{q}}^{-1}(\mathbf{V}) \\ -\boldsymbol{\omega} \times \mathbf{V} + \frac{1}{m}\mathbf{F} \\ \frac{1}{2}\boldsymbol{\omega}\mathbf{q} \\ \mathbf{J}^{-1}\left[-\boldsymbol{\omega} \times (\mathbf{J}\boldsymbol{\omega}) + \mathbf{M}\right] \end{pmatrix}. \tag{5.30}$$

Here $\mathcal{R}_{\mathbf{q}} : \mathbb{R}^3 \to \mathbb{R}^3$ is the rotation (computed using the attitude $\mathbf{q}$) from inertial to body frame, and $\mathcal{R}_{\mathbf{q}}^{-1}(\cdot)$ is the inverse rotation from body to inertial frame. Next, m is the mass of the UAV, $\mathbf{J} \in \mathbb{R}^{3\times 3}$ is the UAV's inertia matrix, and

$$\boldsymbol{\omega} \coloneqq \begin{pmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{pmatrix}.$$

Finally $\mathbf{F} = \mathbf{F}(\mathbf{x}, \mathbf{u})$ and $\mathbf{M} = \mathbf{M}(\mathbf{x}, \mathbf{u})$ are the external forces and moments acting on the vehicle expressed in the body frame. These arise as a result of gravity, aerodynamics, and control inputs. In the following presentation we ignore the effects of wind for simplicity.

The first force is gravity, which can be expressed in the body frame as

$$\mathbf{F}_{\text{gravity}} = \mathcal{R}_{\mathbf{q}} \left[\begin{pmatrix} 0, & 0, & mg \end{pmatrix}^T \right],$$

where g is the gravitational constant. Next we employ a linear propellor model based on [7]:

$$\mathbf{F}_{\text{prop}} = \frac{1}{2} \rho \pi R_{\text{prop}}^2 C_{\text{prop}} \begin{pmatrix} k_{\text{motor}}^2 \delta_t - \|\mathbf{V}\|^2 \\ 0 \\ 0 \end{pmatrix}. \tag{5.31}$$

Here ρ is the air density, R_{prop} is the propellor blade length, and k_{motor} and C_{prop} are parameters modeling thrust efficiency. We choose this simple model so that the dynamics become control affine, making it possible to write down the optimal control explictly as a function of state and costate.

Finally, the aerodynamic forces $\mathbf{F}_{\text{aero}} = \begin{pmatrix} F_x, & F_y, & F_z \end{pmatrix}$ and moments $\mathbf{M} = \begin{pmatrix} M_\ell, & M_m, & M_n \end{pmatrix}$ are in general complicated nonlinear relationships that must be modeled from experimental data. In this work we use the basic models from [7], with slight nonlinear modifications to the drag and pitching moment models to improve their post-stall realism. The longitudinal forces are modeled as

$$\begin{pmatrix} F_x \\ F_z \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} -F_D \\ -F_L \end{pmatrix},$$

where

$$\alpha = \tan^{-1}(w/u)$$

is the angle of attack and

$$F_L = \frac{1}{2}\rho\|\mathbf{V}\|^2 S\left[C_L(\alpha) + \frac{cC_{L_q}}{2\|\mathbf{V}\|}q + C_{L_{\delta_e}}\delta_e\right],$$
$$F_D = \frac{1}{2}\rho\|\mathbf{V}\|^2 S\left[C_D(\alpha) + \frac{cC_{D_q}}{2\|\mathbf{V}\|}q + C_{D_{\delta_e}}\delta_e\right],$$

are the lift and drag forces, respectively. Here S is the wing area and C_{L_q}, $C_{L_{\delta_e}}$, C_{D_q}, $C_{D_{\delta_e}}$ are modeling parameters. As in [7] we model

$$\begin{aligned} C_L(\alpha) = &[1-\sigma_b(\alpha)]\left[C_{L_0} + C_{L_\alpha}\alpha\right] \\ &+ \sigma_b(\alpha)\cdot 2\text{sign}(\alpha)\sin^2\alpha\cos\alpha, \end{aligned} \tag{5.32}$$

where C_{L_0} and C_{L_α} are modeling parameters and $\sigma_b(\alpha)$ is a smooth blending function which is $\sigma_b(\alpha) \approx 0$ for $|\alpha| < \alpha_{\text{stall}}$ and $\sigma_b(\alpha) \approx 1$ for $|\alpha| > \alpha_{\text{stall}}$, with α_{stall} being the stall angle of attack[4]. See [7] for details. For the drag model we use a blend of a quadratic and post-stall flat plate model [38]:

$$\begin{aligned} C_D(\alpha) = &[1-\sigma_b(\alpha)]\left[C_{D_0} + \frac{(C_{L_0} + C_{L_\alpha}\alpha)^2}{\pi e b^2/S}\right] \\ &+ \sigma_b(\alpha)\cdot 2\sin^2\alpha, \end{aligned} \tag{5.33}$$

where C_{D_0} is the parasitic drag, b is the wingspan, and e is another modeling parameter. We similarly modify the pitching moment model from [7] to be nonlinear in α. Let

$$M_m = \frac{1}{2}\rho\|\mathbf{V}\|^2 Sc\left[C_m(\alpha) + \frac{C_{m_q}c}{2\|\mathbf{V}\|}q + C_{m_{\delta_e}}\delta_e\right], \tag{5.34}$$

[4]We set $\alpha_{\text{stall}} = 20°$ which is lower than [7, 8], making the model more realistic and challenging to control.

with

$$C_m(\alpha) = [1 - \sigma_b(\alpha)] \tanh\left(C_{m_0} + C_{m_\alpha}\alpha\right) + \sigma_b(\alpha) \cdot C_{m_\infty} \sin(-\alpha), \tag{5.35}$$

and where C_{m_q}, $C_{m_{\delta_e}}$, C_{m_0}, C_{m_α}, and C_{m_∞} are modeling parameters. The remaining lateral aerodynamics, F_y, M_ℓ, and M_n, are functions of $\|\mathbf{V}\|$, p, r, δ_a, δ_r, and the sideslip $\beta = \sin^{-1}\left(v/\|\mathbf{V}\|\right)$. These models are the same as in [7] and are omitted for brevity. The values of the constants used in this problem are taken from [8], with the exception of $C_{\text{prop}} = 0.45$, $k_{\text{motor}} = 32$, $\alpha_{\text{stall}} = 20°$, and $C_{m_\infty} = 0.8$.

5.3.2 Optimal control problem formulation

We aim to design a feedback controller to stabilize the UAV and track any desired altitude $h_f = -p_{d,f}$ and course angle $\chi_f = \tan^{-1}\left(\dot{p_e}/\dot{p_n}\right)$. Let $\mathbf{x}_f, \mathbf{u}_f$ be the pair of *trim* states and controls computed for a desired airspeed $\|\mathbf{V}_f\|$. The UAV is in trim if $\mathbf{f}\left(\mathbf{x}_f, \mathbf{u}_f\right) = \mathbf{0}$, except for $\dot{p}_n$ and $\dot{p}_e$. Note that the dynamics are invariant to $\mathbf{p}$, so we can choose any arbitrary trim altitude. The dynamics (excepting $\dot{p}_n$ and $\dot{p}_e$) are also invariant to rotations of the inertial reference frame about the inertial z axis, which allows us to use the same trim attitude $\mathbf{q}_f$ to express any desired yaw angle. When the vehicle is in trim (and in the absence of wind) the yaw angle ψ is equal to the course angle χ, and thus this formulation allows arbitrary course tracking.

A suitable running cost for this OCP is

$$\mathcal{L}\left(\mathbf{x}, \mathbf{u}\right) = Q_h \left[h_{\text{ceil}} \tanh\left(\frac{p_d - p_{d,f}}{h_{\text{ceil}}}\right)\right]^2$$

$$
\begin{aligned}
&+ (\mathbf{V} - \mathbf{V}_f)^T \mathbf{Q}_\mathbf{V} (\mathbf{V} - \mathbf{V}_f) \\
&+ (\bar{\mathbf{q}} - \bar{\mathbf{q}}_f)^T \mathbf{Q}_\mathbf{q} (\bar{\mathbf{q}} - \bar{\mathbf{q}}_f)^T \\
&+ (\boldsymbol{\omega} - \boldsymbol{\omega}_f)^T \mathbf{Q}_{\boldsymbol{\omega}} (\boldsymbol{\omega} - \boldsymbol{\omega}_f) \\
&+ (\mathbf{u} - \mathbf{u}_f)^T \mathbf{R} (\mathbf{u} - \mathbf{u}_f), \qquad (5.36)
\end{aligned}
$$

where $Q_h, h_{\text{ceil}} > 0$, $\mathbf{Q}_\mathbf{V}, \mathbf{Q}_\mathbf{q}, \mathbf{Q}_{\boldsymbol{\omega}} \in \mathbb{R}^{3\times 3}$ are positive definite, and $\mathbf{R} \in \mathbb{R}^{4\times 4}$ is positive definite. Notice that the altitude cost is locally quadratic but saturates for $|p_d - p_{d,f}| \geq h_{\text{ceil}}$, preventing extreme maneuvers when the commanded altitude changes. To summarize, we consider the following OCP:

$$
\begin{cases}
\underset{\mathbf{u}(\cdot)}{\text{minimize}} & \mathcal{J}[\mathbf{u}(\cdot)] = \displaystyle\int_0^\infty \mathcal{L}(\mathbf{x}, \mathbf{u})\, dt, \\
\text{subject to} & \dot{\mathbf{p}} = \mathcal{R}_\mathbf{q}^{-1}(\mathbf{V}), \\
& \dot{\mathbf{V}} = -\boldsymbol{\omega} \times \mathbf{V} + \frac{1}{m}\mathbf{F}, \\
& \dot{\mathbf{q}} = \frac{1}{2}\boldsymbol{\omega}\mathbf{q}, \\
& \dot{\boldsymbol{\omega}} = \mathbf{J}^{-1}[-\boldsymbol{\omega} \times (\mathbf{J}\boldsymbol{\omega}) + \mathbf{M}], \\
& \mathbf{u}(t) \in [0,1] \times [-\delta_a^+, \delta_a^+] \times [-\delta_e^+, \delta_e^+] \times [-\delta_r^+, \delta_r^+].
\end{cases} \qquad (5.37)
$$

We set the desired airspeed at $\|\mathbf{V}_f\| = 20$ [m/s] and use the following cost function parameters:

$$
\begin{cases}
h_{\text{ceil}} = 50 \text{ [m]}, \qquad Q_h = 1/h_{\text{ceil}}^2, \\
\mathbf{Q}_\mathbf{V} = \text{diag}\left(10/\|\mathbf{V}_f\|^2, \quad 1, \quad 1\right), \\
\mathbf{Q}_\mathbf{q} = 5\mathbf{I}_{3\times 3}, \\
\mathbf{Q}_{\boldsymbol{\omega}} = \frac{1}{(30^\circ)^2}\mathbf{I}_{3\times 3}, \\
\mathbf{R} = \text{diag}\left(0.1, \quad 0.1/(\delta_a^+)^2, \quad 1/(\delta_e^+)^2, \quad 1/(\delta_r^+)^2\right).
\end{cases} \qquad (5.38)
$$

Initial conditions are uniformly sampled from the following domain to elicit a wide

range of nonlinear dynamics:

$$\mathbb{X}_0 = \left\{ \begin{array}{l} p_{d_0} \in [-3h_{\text{ceil}}, 3h_{\text{ceil}}]\,, \\ \mathbf{V}_0 \in [\mathbf{V}_f - 5\ [\text{m/s}], \mathbf{V}_f + 5\ [\text{m/s}]]\,, \\ \psi_0, \phi_0 \in [-180°, 180°]\,, \quad \theta_0 \in [-90°, 90°]\,, \\ \boldsymbol{\omega}_0 \in [-30\ [\text{deg/s}], 30\ [\text{deg/s}]]\,. \end{array} \right\} \tag{5.39}$$

Here ψ_0, θ_0, ϕ_0 denote the inital yaw, pitch, and roll angles, which are converted to the initial quaternion $\mathbf{q}_0$. Recall that we can set $p_{d_f} = 0$ and $\psi_f = 0$ without loss of generality, and thus the initial condition determines the initial altitude and course errors.

5.3.3 Learning results

For this problem we generate training data sets with $N_{\text{OCP, train}} = 64$, 128, and 256 trajectories each. For each data set we train each type of NN controller with different weight initializations. As before we conduct ten trials for each data set size. We evaluate the RMℓ^2 error (2.34) on an independent data set with $N_{\text{OCP, test}} = 100$ trajectories. As in Section 5.2 all NNs have $L = 5$ hidden layers with $w = 32$ neurons each and $\tanh(\cdot)$ nonlinearities. Because these data sets are too large for full-batch optimization we optimize the loss function (2.27) with the Adam optimizer [73] using a learning rate of 10^{-3}, batch sizes of 256 data points, and 1500 epochs.

Figure 5.9 shows results for the LAS verification. As before we find that even well-trained u-NNs may fail to even locally stabilize the system. Furthermore, for this OCP the closed loop equilibrium under these standard NN controllers is often far from $\mathbf{x}_f$. In the physical system this corresponds to steady state altitude, course, and attitude errors, even when said equilibrium is stable.

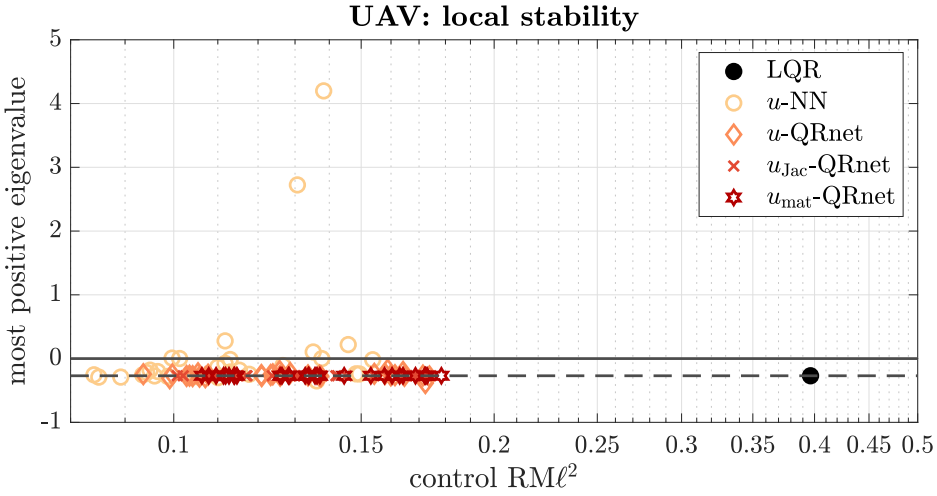


Figure 5.9: Real part of most positive closed loop Jacobian eigenvalue at an equilibrium. For u-NN the equilibrium point is often far from $\mathbf{x}_f$.

Figure 5.10 shows the worst case norm for a set of $N_{\text{MC}} = 100$ closed loop simulations. For this problem we also find that the proportion of stabilized MC trajectories to be informative. Figure 5.11 shows the fraction of $N_{\text{MC}} = 100$ closed loop simulations for which $\left\|\mathbf{x}\left(t_f; \mathbf{x}_0^{(i)}\right) - \mathbf{x}_f\right\| \leq \delta$, where δ is a tolerance that we set to $\delta = 1$.

These MC simulations illustrate how challenging the UAV is to control over this large spatial domain. First we notice that LQR is *not* globally stabilizing for this OCP. Next we observe that most standard u-NNs, even the well-trained ones, do not stabilize $\mathbf{x}_f$. *u-QRnet*, *u_{Jac}-QRnet*, and *u_{mat}-QRnet* also have some difficulty with semi-global stabilization, though they clearly do better than u-NN. Note that these controllers are able to stabilize trajectories LQR fails to stabilize, even though they are built on top of LQR. Curiously, *u_{Jac}-QRnet* and *u_{mat}-QRnet* do better than *u-QRnet* in terms of the proportion stabilized (Figure 5.11) but their worst case performance (Figure 5.10) tends to be worse.

Finally Figure 5.12 shows the average performance of each controller in terms

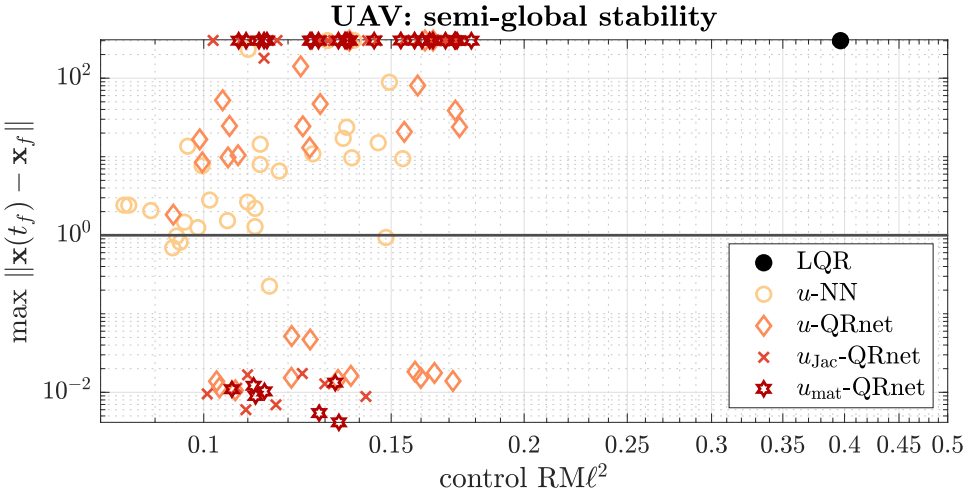


Figure 5.10: Worst-case norm of final state over $N_{\text{MC}} = 100$ simulations. The vertical axis is limited since we stop simulations if the altitude h goes outside of ± 300 [m] with respect to the commanded h_f.

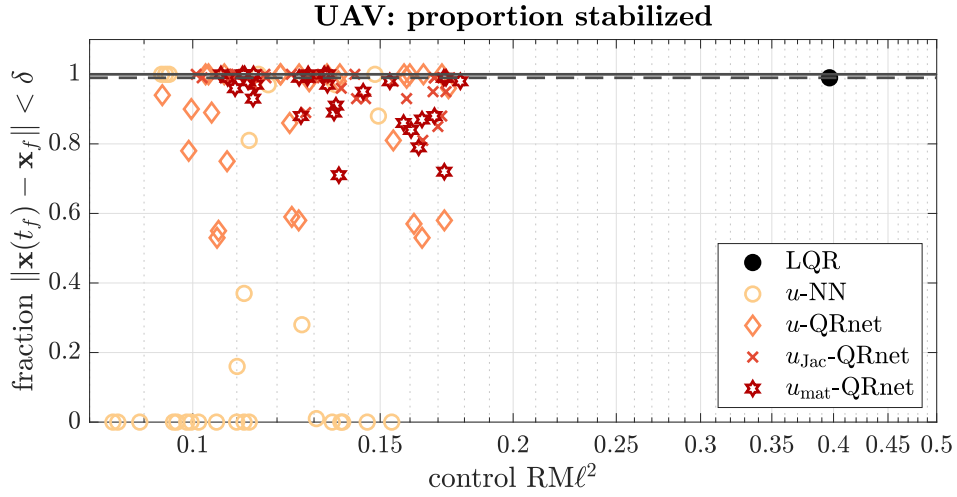


Figure 5.11: Fraction of NN-controlled trajectories with $\|\mathbf{x}(t_f)\| \leq \delta = 1$.

of minimizing the cost functional $\mathcal{J}[\mathbf{u}(\cdot)]$. We again see that most NN controllers perform better than LQR on average, indicating that they do learn the optimal policy reasonably well. We also see that the standard u-NNs have slightly highter test accuracy, suggesting that for this OCP the training loss (2.27) converges faster than the modified architecture (i.e. requires fewer gradient descent steps). Despite this, we can see that *u-QRnet*, *u_{Jac}-QRnet*, and *u_{mat}-QRnet* perform just

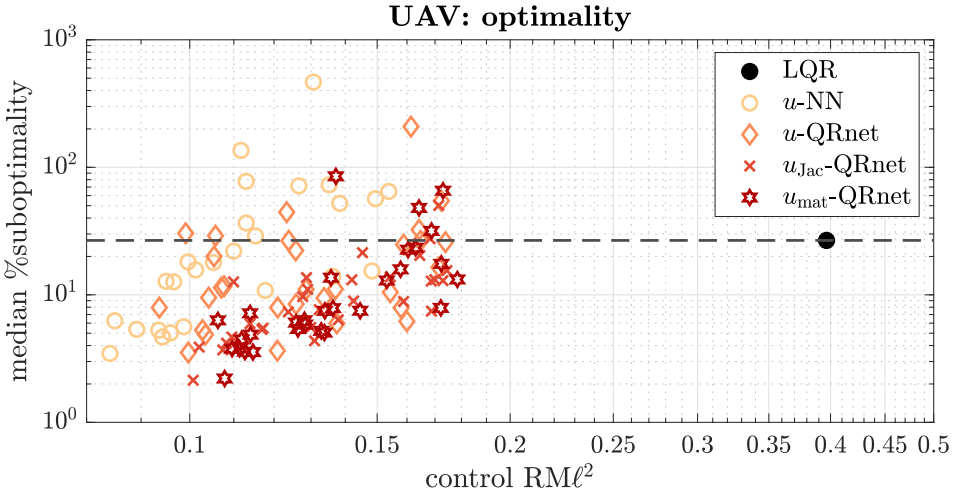


Figure 5.12: Median percent cost more than optimal cost over $N_{\text{MC}} = 100$ simulations.

as good or better in terms of closed loop stability and optimality. We expect that all methods will improve with higher quality data from indirect methods, larger data sets, more training epochs, and hyperparameter tuning.

5.3.4 Example closed loop simulations

We conclude this section with some illustrative examples of NN-in-the-loop simulations. We randomly select an initial condition, for which the UAV begins off course and pitched down with large negative pitch rate. We take two controllers, a u-NN and a u_{mat}-*QRnet*, both trained on $N_{\text{OCP, train}} = 256$ trajectories. Three-dimensional positions are shown in Figure 5.13 and detailed time histories of states and feedback controls are given in Figures 5.14 and 5.15. We see that u-NN initially does a good job before suddenly and unexpectedly going unstable as it nears $\mathbf{x}_f$. The system eventually converges to an equilibrium $\bar{\mathbf{x}} \neq \mathbf{x}_f$ which has non-zero steady state course error, $\chi_f \approx -22.4°$. On the other hand, both LQR and u_{mat}-*QRnet* stabilize the system. For this initial condition LQR is 36.44%

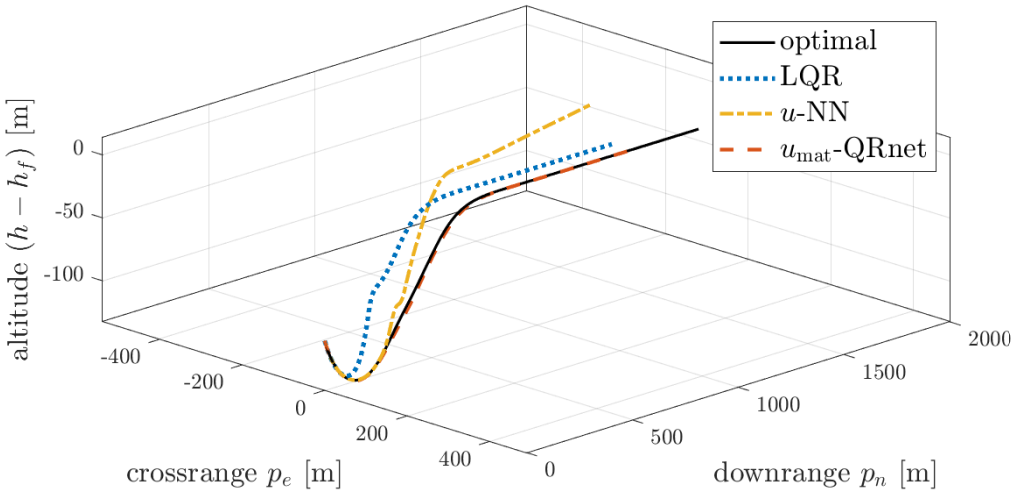


Figure 5.13: Closed loop trajectories with u-NN and u_{mat}-*QRnet* controllers, compared to the LQR and optimal trajectories. Detailed time histories are given in Figures 5.14 and 5.15.

suboptimal while u_{mat}-*QRnet* is only 0.95% suboptimal.

This simulation highlights two key ideas. The unstable simulation makes it clear why recovering exact equilibria and building in LAS is so important when dealing with highly nonlinear systems. This controller has good approximation accuracy and does well for the initial portion of the trajectory which is actually most difficult to control. But it eventually fails just before reaching equilibrium. However, the success of u_{mat}-*QRnet* and its superior performance compared to LQR in this scenario demonstrates the potential benefit of designing nonlinear optimal feedback controllers.

5.4 Summary

Chapters 2 and 3 highlighted the promise of the proposed supervised learning framework to facilitate optimal feedback design for high-dimensional nonlinear

systems. However, in Chapter 4 we found that NN feedback controllers can frequently fail to stabilize a system, even when they are trained to a high degree of accuracy. This occurs frequently enough that it cannot be ignored.

One strategy to make NN feedback control more viable is through the use of specialized NN architectures. In this chapter we have introduced seven new model architectures which smoothly combine an LQR to build in LAS with an NN to learn the nonlinear parts of the optimal control. Four of these also *guarantee* (at least) local stability, while still retaining the approximation capacity necessary to learn the full nonlinear optimal control. and provide nonlinear stability on semi-global domains. In Section 5.2 we evaluated the proposed architectures through a series of practical closed loop stability and optimality tests, demonstrating their advantages over standard NNs. In Section 5.3 we illustrated how the proposed architectures might be used with supervised learning to design optimal feedback controllers for challenging, practical systems.

For problems where the dimension is not too large, the value gradient approximators, λ-*QRnet*, λ_{Jac}-*QRnet*, and λ_{mat}-*QRnet*, can sometimes perform better than the control approximators, u-*QRnet*, u_{Jac}-*QRnet*, and u_{mat}-*QRnet*. This is because they encode additional physical structure and can learn from costate data in addition to control data. On the other hand, the control approximators are generally much faster to train, and they can be implemented even when it is not possible to solve (1.21) for the optimal control, and when it is difficult to generate accurate costate data.

The value gradient models, λ-*QRnet*, λ_{Jac}-*QRnet*, and λ_{mat}-*QRnet*, may have a further drawback in their current form. For some problems, including the UAV OCP (5.37), the Riccati matrix $\mathbf{P}$ may have exceedingly large values. These may be due to poor problem scaling or instances where the linear dynamics are only

barely controllable or observable. In any case it is clear that for such problems, the linear term $2\mathbf{P}(\mathbf{x} - \mathbf{x}_f)$ in (5.5), (5.7) and (5.9) becomes very large away from $\mathbf{x}_f$. This will prevent the NN component from being able to accurately approximate the nonlinear parts of the value gradient. This problem does not affect the optimal control models, *u-QRnet*, u_{Jac}-*QRnet*, and u_{mat}-*QRnet*, since in these the LQR contribution is saturated. To make the value gradient models more generally viable, future work should investigate similar saturation techniques for modifying these architectures.

Among the various proposed architectures, the "Jacobian" and "matrix" *QRnet*s have the obvious advantage of guaranteed LAS. For OCPs where achieving local stability is not so difficult, the simpler λ-*QRnet* and *u-QRnet* can be a reasonable alternative because they almost always yield LAS feedback in practice. At the time of writing it is difficult to judge whether the "Jacobian" or "matrix" *QRnet*s are more effective. In the UAV example (5.37) there is some evidence that u_{Jac}-*QRnet* slightly outperforms u_{mat}-*QRnet* in terms of the fraction of stabilized MC simulations; see Figure 5.11. It may be possible that u_{mat}-*QRnet* is harder to train, either simply because it has more free parameters or because its "loss landscape" [75] is more challenging to navigate by gradient descent.

To summarize these findings, the optimal control models are more generally applicable to a wider range of problems. Still, when they can be implemented the value gradient models can potentially have some performance advantages and so should not be discounted. "Jacobian" and "matrix" *QRnet*s are likely to be preferrable since they guarantee at least LAS and appear to perform just as well as the simpler λ-*QRnet* and *u-QRnet*. Since we have not yet observed a clear performance distinction between "Jacobian" and "matrix" architectures; a thorough investigation of their tradeoffs is an important direction for future study. Of

course there need not be a "best" model: different architectures may be preferable for different problems and it is valuable to have access to a variety of tools.

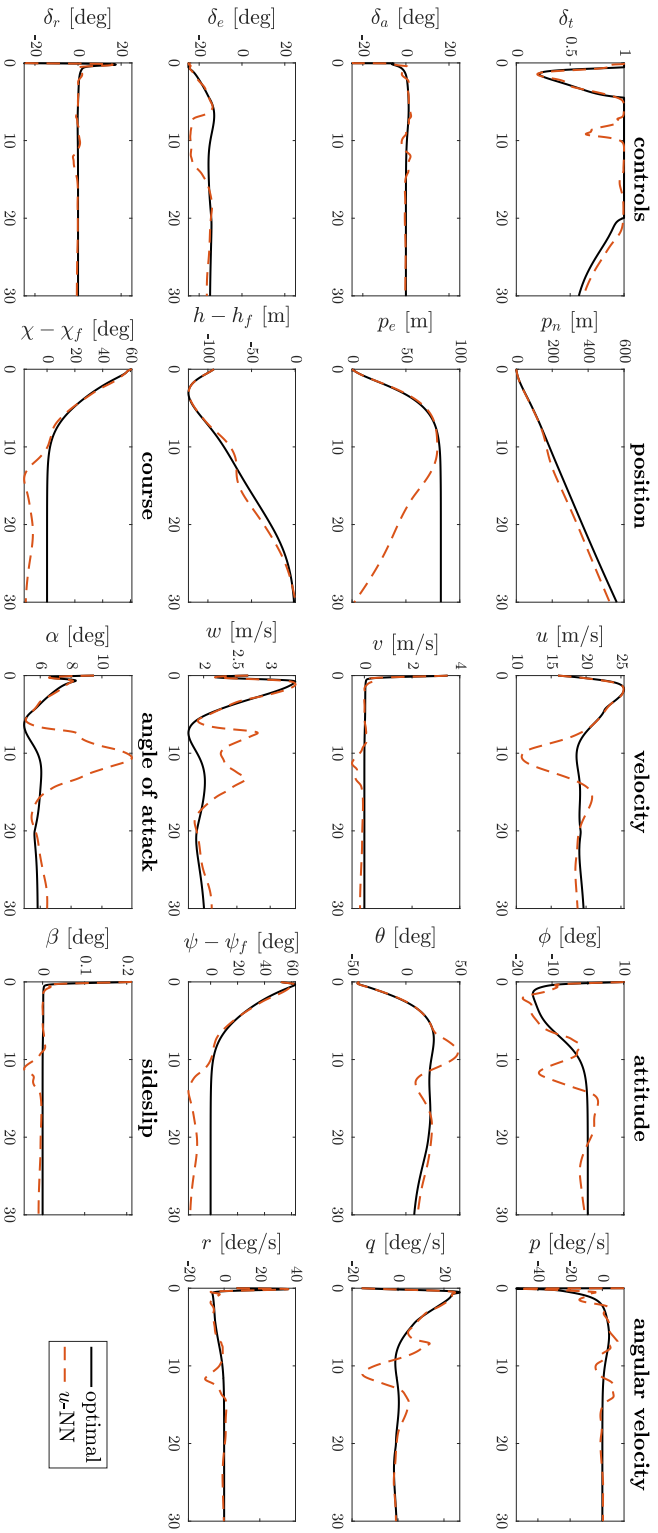


Figure 5.14: Closed loop simulation with a u-NN controller compared to the optimal trajectory. Attitude is given in Euler angles ϕ, θ, ψ (roll, pitch, yaw).

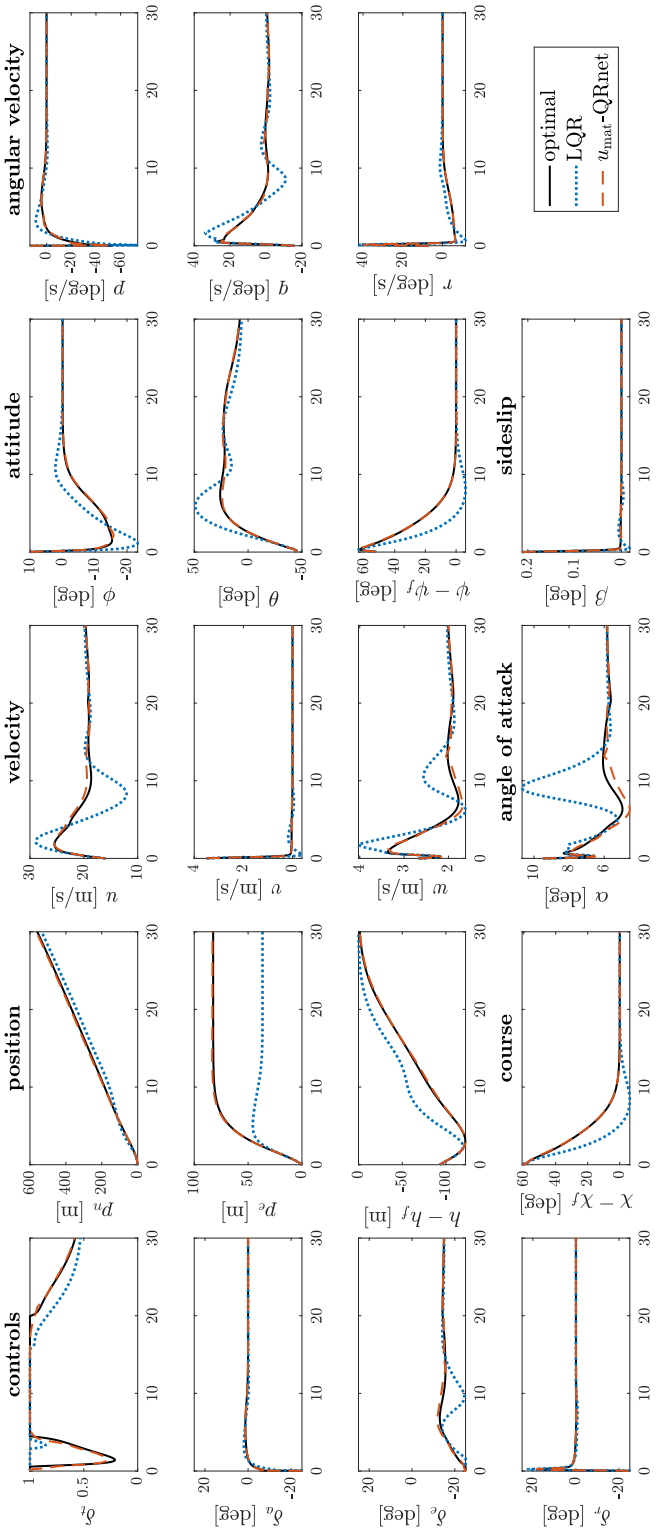


Figure 5.15: Closed loop simulation with a u_{mat}-*QRnet* controller compared to LQR and optimal trajectories. Attitude is given in Euler angles ϕ, θ, ψ (roll, pitch, yaw).

Chapter 6

Conclusions and Future Work

For practicing engineers "optimal control" has long referred only to open loop optimal control or LQR. This is because there were historically no reliable, general methods to synthesize optimal feedback controllers for high-dimensional nonlinear systems. Broadly speaking, computational methods for nonlinear control design have had to sacrifice speed, optimality, or generality. In this dissertation we have developed a computational framework which aims to address this longstanding challenge. Numerical experiments suggest that by combining tools from machine learning with ideas from optimal control theory, we can construct feedback controllers for high-dimensional nonlinear systems which deliver real-time optimal control.

Here we conclude with a short review of our contributions and a discussion of possible directions for future work. First in Section 6.1 we summarize the proposed computational framework and highlight some of its advantages. We present directions for future work in Section 6.2. On a related note, in appendix A we dis-

cuss how some of the ideas we have proposed for optimal feedback design might be used for a different problem, propagation of uncertainty through nonlinear dynamic systems.

6.1 Summary and outlook

In this dissertation we have outlined a computational framework for optimal feedback design. The proposed methodology is based on a combination of supervised learning and well-known optimal control theory. We apply mature computational methods for open loop optimal control to generate data sets, and then use NNs as a computational tool to implement the method of characteristics in high dimensions, thus bridging PMP and HJB. Because basic NN architectures are not reliable when integrated into nonlinear control systems, we propose novel architectures which combine LQRs and standard NNs. LQR provides local stability and optimality, and the NN extends this local solution and learns the nonlinear optimal control on a larger domain.

NNs are fast to evaluate and thus well-suited for online implementation. Although generating data and training the model can take considerable computational resources, in our framework these expensive steps are performed offline. This means that we do not have to compromise optimality or speed when designing the controller. Still, during a typical control design process we will frequently modify the OCP itself to achieve some desired behavior [44]. This is sometimes called the "problem of problems" [119]. Naïvely, one would assume that the proposed framework would be rendered impractical since we would have to generate new data and train new NNs when the OCP is changed. However, if the OCP is only changed incrementally than existing data and NN controllers can be used

for warm start when generating new data sets. Furthermore, model hyperparameters like NN depth, width, and loss function weights, should also work for the updated OCP; and as we have shown, such NN controllers can be trained in a matter of minutes. To summarize, we contend that the proposed framework is efficient enough to be practical for control design, with the reward being an optimal feedback controller that is fast enough for online implementation.

6.1.1 Overview of the proposed computational framework

In this section we review the steps of proposed framework for synthesizing optimal feedback controllers. A further condensed summary is given in Algorithm 1. In each of the following steps, we provide relevant section numbers for further detail.

1. *Initial data generation* (Sections 2.2, 3.1 and 3.2): We uniformly sample a set of initial conditions and, using time marching or LQR warm start, solve an open loop OCP (1.2) for each initial condition. By PMP (1.22) the open loop optimal solution is connected to the closed loop optimal solution, thus providing a data set to learn from. In this initial data generation step we require relatively few data points since more data can be added later at reduced computational cost.

2. *Model training* (Section 2.3.3): Given this data set, we train an NN to approximate the value function, value gradient, or optimal control. As much as possible, learning is guided by the underlying problem structure, asking the NN to satisfy relationships between the value gradient, costate, and optimal control. In doing so we regularize the model and make efficient use out of small data sets.

3. *Adaptive data generation* (Sections 3.2.3 and 3.3.2): In the initial training phase we only have a small data set, so we can only hope for a rough approximation of the value function, value gradient, or optimal control. We now expand the data set by generating data in regions where the value function is likely to be steep or complicated, and thus difficult to learn. Generating additional data is made efficient by good initial guesses obtained from NN warm start.

4. *Model refinement* (Sections 3.3.1 and 3.3.3): We repeat steps 2 and 3, training the model and increasing the size of the data set until we satisfy some convergence criteria.

5. *Model evaluation* (Sections 2.3.4 and 4.1): We check the generalization accuracy of the trained NN on an independent set of test data. We also perform linear stability analysis and nonlinear NN-in-the-loop simulations to verify that the system behaves as desired.

Once the NN has been trained to desired accuracy and thoroughly tested, it can be ported to onboard processors [82]. For optimal control models the control outputs can be used directly. For value gradient models the control is specified by the Hamiltonian minimization condition (1.18), which for many problems can be solved explicitly (see Section 2.3.5). For value function models we also use (1.18) to compute the control, this time based on the gradient of the NN. Using automatic differentiation, this gradient is computed exactly and cheaply for large n. All of these methods are fast, enabling real-time implementation in high-dimensional systems.

6.1.2 Stability-enhancing architectures

The steps above outline a general computational framework for synthesizing optimal feedback controllers through supervised learning. We have seen that this approach is promising, but in Chapter 4 we found that standard NN architectures were unreliable when implemented in the closed loop system. Often, NNs trained to a high level of accuracy would fail to even locally stabilize a system. Thus to achieve our desired goals of semi-global stability and optimality we propose a series of specialized NN architectures with built-in stability properties.

The proposed architectures are presented and evaluated in Chapter 5, and summarized in Table 5.1. They are V-*QRnet*, V_{Hess}-*QRnet*, λ-*QRnet*, u-*QRnet*, λ_{Jac}-*QRnet*, u_{Jac}-*QRnet*, λ_{mat}-*QRnet*, and u_{mat}-*QRnet*. The prefixes "V", "λ", and "u" refer to what each NN models, namely the value function, value gradient, or optimal control; "Hess", "Jac", and "mat" subscripts stand for "Hessian", "Jacobian", and "matrix" architectures respectively; and the name *QRnet* describes the combination of LQR for LAS with an NN for nonlinear approximation power.

The "Hessian", "Jacobian", and "matrix" architectures guarantee at least LAS of a desired equilibrium $\mathbf{x}_f$, and all the NN architectures retain the approximation capacity need to learn the nonlinear optimal control. The V_{Hess}-*QRnet* model has not yet been implemented and tested; this is left for future work. So far no clear evidence has been found to recommend a single architecture, and we believe that having options can be useful as some may perform better than others for specific problems.

In Chapter 5 we train these stability-enhancing architectures using supervised learning. We find that they facilitate reliable synthesis of optimal feedback design for the unstable Burgers' PDE in $n = 64$ dimensions and the difficult, practical

UAV problem in $n = 11$ dimensions. More work can be done to further improve and study the properties of these NNs, but we believe the examples presented in this dissertation demonstrate their potential.

6.2 Directions for future work

Throughout this dissertation we have intentionally used the word "framework" to describe our proposed computational methods. While we have obtained promising results on a number of high-dimensional nonlinear examples, there is still work to be done to translate this proof of concept into practical and robust software that can be implemented in real systems.

First to complement the NN architectures presented in Chapter 5, in future work we intend to conduct further numerical experiments and develop mathematical tools to explain the behavior of NN feedback controllers. In particular, we would like to better understand what causes seemingly-accurate NN models to fail at stabilizing a system, as well as why and to what extent the novel NN architectures improve semi-global system stability. Such experimental and theoretical advances will be necessary if supervised learning is to become a reliable and commonly accepted control design method. We also expect that further study will help to design even better NN architectures.

Generating data remains the most important and possibly most challenging aspect of the framework. We have discussed several tools to make generating data more tractable, and also noted a few alternative methods proposed in similar research. Still, a thorough study must be done to understand the strengths and weaknesses of each approach and how they may be combined to complement one another. Ultimately we cannot expect that any single approach will be general

and powerful enough for all practical problems, and hence generating data for industry strength problems will be time consuming and require specific domain knowledge.

Practical problems will also often come with path constraints (e.g. an aircraft must not crash into the ground) or endpoint constraints (e.g. a spacecraft must land on the moon with enough fuel reserves to reach orbit again), or have different objectives such as time or fuel optimality. Such constraints and objective functions are not immediately described by the OCPs (1.2) and (1.4), and lead to technical challenges including non-uniqueness of BVP solutions and nonsmooth value functions. Extensions to the framework to handle different types of OCP will greatly improve its usability in real world settings.

Finally, perhaps the biggest challenge with practical control problems is *uncertainty*. Uncertainty enters in dynamic model specification since we can never completely describe a physical system with mathematical formulas, so the real world system will always be different from our model. Even if we can construct a good mathematical representation of the system, it will invariably contain parameters that must be estimated from data and are therefore uncertain. Parameter values might also change over time, leading to further deviation. We must also deal with uncertainty in *state estimates* which are reconstructed based on (noisy) measurements and filtering algorithms. A number of *robust control* techniques have been proposed to try to explicitly handle certain types of uncertainty. Some examples include stochastic optimal control [26, 51, 52] – especially linear quadratic Gaussian (LQG) control [130], sliding mode control [71], probability density control [50, 48], nonlinear H_∞ optimal control [137, 63, 61], and optimal control of systems with uncertain parameters [112, 122, 113, 127, 79, 128].

Our supervised learning-based framework may be most straightforwardly ex-

tended to control systems with uncertain parameters. Since NNs are naturally able to handle high-dimensional input spaces, uncertain parameters could be included as additional model inputs and data could be generated over the support of uncertain parameters. For online implementation one could estimate the values of the parameters in real time and implement the control based on this estimation. A potential challenge with this formulation is the construction of appropriate stability-enhancing architectures, since now LQR will be suboptimal and possibly not stabilizing away from nominal parameter values. Still, we are confident that the framework can be extended to explicitly handle uncertainty, which would further enhance its utility in practical applications.

Appendix A

Characteristics-based learning for uncertainty propagation

The idea of *characteristics-based learning* developed in this dissertation might be applied to other interesting problems like uncertainty propagation through nonlinear dynamics. In actuality our work on optimal feedback control design grew out of preliminary research on this topic [139, 101].

Uncertainty propagation through nonlinear dynamical systems remains an outstanding problem in scientific computing. A number of computational approaches have been developed, but many of these are limited in their capability to tackle problems with more than a few uncertain variables or require large amounts of simulation data. In this appendix we present a data-driven method for approximating joint and conditional probability density functions (PDFs) of nonlinear dynamical systems with initial condition and parameter uncertainty.

Similar to our approach for solving HJB equations, we use NNs to model the

time-dependent PDF of the state. We train these NNs by solving the *Liouville PDE* which describes the evolution of an initial PDF through nonlinear dynamics. Here we employ a semi-supervised learning approach which leverages knowledge of the underlying dynamics and the problem physics. Specifically, we generate data by propagating sample initial conditons, while simultaneously fitting the PDF model to satisfy the Liouville PDE at collocation points. Once the NN is trained, it can predict the density at arbitrary points in the computational domain at orders of magnitude more efficiently than numerical integration. We demonstrate good initial results on a benchmark uncertainty propagation problem.

A.1 Problem setting and related work

Consider an n-dimensional system of nonlinear autonomous first-order ODEs with random initial condition:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}), \qquad \mathbf{x}(0) = \mathbf{x}_0 \sim \rho_0(\mathbf{x}_0). \tag{A.1}$$

Here $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ is a Lipschitz continuous vector field and $\rho_0 : \mathbb{R}^n \to [0, \infty)$ is a given finite-valued PDF. This setting includes uncertainty in parameter space if we augment the state with the set of uncertain parameters $\boldsymbol{\beta}$ and the vector field $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$ with $\dot{\boldsymbol{\beta}} = \mathbf{0}$. Randomness in the initial condition $\mathbf{x}_0$ induces randomness in the future state $\mathbf{x}(t; \mathbf{x}_0)$. We can think of this as a stochastic process defined by the *forward flow map*

$$\mathbf{x}(t; \mathbf{x}_0) = \boldsymbol{\Phi}(\mathbf{x}_0, t), \tag{A.2}$$

with a distribution characterized by the PDF $\rho(\mathbf{x}|t)$. Given a model of the dynamics and an initial density, we would like to approximate the PDF over the

time interval $t \in [0, t_f]$ for some $t_f \in (0, \infty)$.

MC simulation is a standard practical technique for uncertainty propagation of stochastic dynamical systems. But since $\mathbf{f}(\cdot)$ is nonlinear, the flow map and thus the state PDF $\rho(\cdot)$ can be quite complicated. Often, statistical information such as the mean and covariance of the state $\mathbf{x}(t)$ provide poor characterizations of the distribution $\rho(\cdot)$ and the sensitivity of the state $\mathbf{x}(t)$ to perturbations in the initial conditions or parameters. Furthermore, propagating sufficiently many samples to characterize the PDF can be prohibitively costly. Thus for many applications it is desirable to obtain a low-cost representation of the PDF itself.

Techniques for estimating $\rho(\mathbf{x}|t)$ include the unscented Kalman filter [60], ensemble Kalman filter [40], kernel density estimation [126, Chapter 6], generalized polynomial chaos [147, 142, 87, 141], probabilistic collocation [135, 43], adaptive Gaussian mixture models [31], and low-dimensional series expansions [24]. Many of these methods suffer from the curse of dimensionality, and even those that scale well to high dimensions can be difficult to implement or data-hungry. To address this challenge, a number of researchers have proposed algorithms based on the method of characteristics for the Liouville PDE [49, 68, 18, 45]. As we have seen in this dissertation, the method of characteristics can be a powerful approach for high-dimensional problems. These algorithms of course have a common theme: *data* gathered along characteristics. Beyond that, however, one can come up with any number of schemes for synthesizing these data into a single PDF model. Our semi-supervised learning method appears to be promising for certain problems, but we make no claim of having solved all uncertainty propagation problems.

A.2 The Liouville equation

It is well-known (see e.g. [36]) that the PDF $\rho(\mathbf{x}|t)$ evolving according to the nonlinear dynamics (A.1) satisfies the *Liouville transport equation*,

$$\begin{cases} \rho_t + \nabla_{\mathbf{x}} \cdot [\rho \mathbf{f}] = 0, \\ \rho(\mathbf{x}|t=0) = \rho_0(\mathbf{x}). \end{cases} \tag{A.3}$$

Solving (A.3) directly requires a discretization of space and time which must be extremely fine because the support of the PDF can twist into thin curves within state space. Consequently, solving (A.3) can be challenging even in low dimensions, and, like the HJB equation (1.17), intractable when n is large.

On the other hand, using the method of characteristics one can derive a formal expression for the solution of (A.3):

$$\rho(\mathbf{x}|t) = \rho_0\left(\mathbf{\Phi}^{-1}(\mathbf{x},t)\right) \exp\left[-\int_0^t \nabla_{\mathbf{x}} \cdot \mathbf{f}\left(\mathbf{\Phi}\left(\mathbf{x}_0,s\right)\right) ds\right]. \tag{A.4}$$

Here $\mathbf{\Phi}^{-1}(\mathbf{x},t) = \mathbf{x}_0$ is the *inverse flow map* which maps the state $\mathbf{x}\left(t;\mathbf{x}_0\right)$ back to the initial condition $\mathbf{x}_0$ which generated it. For finite time t, this map is the unique inverse of the forward flow map (A.2) under standard smoothness assumptions on the vector field $\mathbf{f}(\cdot)$. This representation effectively decouples *pointwise* solutions (A.3), allowing these to be computed independently.

To use (A.4) without numerically integrating (A.1), one must first find representations of the forward and inverse flow maps, which are high-dimensional nonlinear functions. Approximating these maps is the objective of *reduced order modeling*, which is closely related to *system identification*. This task is perhaps more difficult than solving (A.3) and a subject of ongoing research (see e.g.

[87, 89, 131, 22, 59]). Still, (A.4) allows us to evaluate the density along characteristics quite easily. Concretely, if we rearrange terms in (A.3) then we see that $\rho(\mathbf{x}|t)$ evolves along flows according to

$$\dot{\rho} = -\rho \, \nabla_{\mathbf{x}} \cdot \mathbf{f}(\mathbf{x}). \tag{A.5}$$

This means that every time a sample trajectory of (A.1) is computed, we can simultaneously propagate $\rho(\mathbf{x}|t)$. Having such density data readily available suggests a data-driven approach, but one which is augmented by knowledge of the underlying physics, i.e. that $\rho(\mathbf{x}|t)$ obeys the Liouville equation (A.3).

A.3 Learning probability density functions

As we have seen in this dissertation, deep learning offers an efficient way to approximate high-dimensional nonlinear functions. Once trained, NNs could enable us to estimate joint and conditional densities at millions of spatio-temporal coordinates in seconds. This in turn allows computation of statistics like means and covariances in high dimensions through Markov Chain Monte Carlo (MCMC) techniques.

Unfortunately, deep NNs notoriously require enormous quantities of data to train, but we consider the case where data is not necessarily abundant. This situation can arise when numerical integration of a system is expensive. To overcome a relative lack of data, we adapt the physically-motivated machine learning strategies introduced in Section 2.3.3.1 for solving high-dimensional HJB equations, and [116] and [129] for other PDEs. The remainder of this section outlines the training process.

Throughout this appendix we model the *logarithm* of the density using standard fully-connected NNs. Thus we construct the PDF approximation as

$$\widehat{\rho}\left(\mathbf{x}|t,\boldsymbol{\theta}\right) = \exp\left[\mathcal{N}\left(\mathbf{x},t;\boldsymbol{\theta}\right)\right], \tag{A.6}$$

where $\mathcal{N} : \mathbb{R}^n \times [0,t_f] \times \mathbb{R}^p \to \mathbb{R}$. This construction naturally *preserves non-negativity of the PDF.*

A.3.1 Generating data

Analagously to data generation for OCPs as introduced in Section 2.2, we start by sampling a set of $N_{\mathbf{x}_0}$ initial conditions distributed from the given initial distribution:

$$\left\{\mathbf{x}_0^{(i)}\right\}_{i=1}^{N_{\mathbf{x}_0}}, \qquad \mathbf{x}_0^{(j)} \sim \rho_0\left(\mathbf{x}_0\right). \tag{A.7}$$

For each initial condition $\mathbf{x}_0^{(j)}$ we propagate the dynamics (A.1) together with the density (A.5) up to the desired final time t_f. As in standard MC simulations this process is *causality free* and embarassingly parallelizable.

For simplicity let us evaluate all sample trajectories at $N_t + 1$ sample time instances,

$$t_0, \ldots, t_{N_t} \in [0,t_f], \qquad t_0 = 0, \quad t_{N_t} = t_f, \tag{A.8}$$

which can be chosen by the numerical ODE solver or according to some other criteria. Each sample time t_k is associated with a sampled state $\mathbf{x}_k^{(i)} \coloneqq \boldsymbol{\Phi}\left(\mathbf{x}_0^{(i)}, t_k\right)$ and probability density $\rho_k^{(i)} \coloneqq \rho\left(\boldsymbol{\Phi}\left(\mathbf{x}_0^{(i)}, t_k\right) \middle| t_k\right)$. This yields a data set

$$\mathcal{D} = \left\{t_k, \left\{\mathbf{x}_k^{(i)}, \rho_k^{(i)}\right\}_{k=0}^{N_t}\right\}_{i=1}^{N_{\mathbf{x}_0}}. \tag{A.9}$$

Notice that $\mathcal{D}$ consists of associated input-output pairs, namely inputs $\left(t_k, \mathbf{x}_k^{(i)}\right)$ and outputs $\rho_k^{(i)}$. This gives rise to a nonlinear regression problem which we describe in appendix A.3.2 below.

A.3.2 Characteristics-based learning and model evaluation

We have experimented with a number of possible loss functions to optimize the NN density model. We have found that the expected square error of the log density performs well. For any $t \in \mathbb{R}$ this is given by

$$\begin{aligned} &\mathbb{E}_{\rho(\cdot|t)}\left\{\left[\log \rho\left(\mathbf{x}|t\right) - \log \widehat{\rho}\left(\mathbf{x}|t, \boldsymbol{\theta}\right)\right]^2\right\} \\ =&\int_{\mathbb{R}^n} \left[\log \rho\left(\mathbf{x}|t\right) - \log \widehat{\rho}\left(\mathbf{x}|t, \boldsymbol{\theta}\right)\right]^2 \rho\left(\mathbf{x}|t\right) d\mathbf{x} \qquad \text{(A.10)} \\ \approx&\frac{1}{N_{\mathbf{x}_0}} \sum_{i=1}^{N_{\mathbf{x}_0}} \left[\log \rho_k^{(i)} - \log \widehat{\rho}\left(\mathbf{x}_k^{(i)}|t_k, \boldsymbol{\theta}\right)\right]^2. \qquad \text{(A.11)} \end{aligned}$$

Here we have obtained the MC approximation (A.11) of (A.10) based on sample pairs $\left(t_k, \mathbf{x}_k^{(i)}\right) \in \mathcal{D}_{\text{train}}$ and the observation that at each t_k the samples $\mathbf{x}_k^{(i)}$ are i.i.d. according to $\rho\left(\mathbf{x}|t_k\right)$. Then to get a scalar loss function we simply average over sample times:

$$\underset{\rho}{\text{loss}}\left(\boldsymbol{\theta}; \mathcal{D}_{\text{train}}\right) \coloneqq \frac{1}{N_{\mathbf{x}_0}\left(N_t+1\right)} \sum_{i=1}^{N_{\mathbf{x}_0}} \sum_{k=0}^{N_t} \left[\log \rho_k^{(i)} - \log \widehat{\rho}\left(\mathbf{x}_k^{(i)}|t_k, \boldsymbol{\theta}\right)\right]^2. \qquad \text{(A.12)}$$

As we did for NN optimal controllers, we can evaluate an NN PDF model on a test data set, $\mathcal{D}_{\text{test}}$, generated from *independently sampled* initial condtions. For

this purpose we report the symmetric Kullback-Leibler (KL) divergence [76],

$$\mathbb{KL}\left\{\rho \| \widehat{\rho}\right\}(t_k, \boldsymbol{\theta}) \coloneqq \int_{\mathbb{R}^n} \left[\rho\left(\mathbf{x}|t_k\right) - \widehat{\rho}\left(\mathbf{x}|t_k, \boldsymbol{\theta}\right)\right] \left[\log \rho\left(\mathbf{x}|t_k\right) - \log \widehat{\rho}\left(\mathbf{x}|t_k, \boldsymbol{\theta}\right)\right] d\mathbf{x} \tag{A.13}$$

$$\approx \frac{1}{N_{\mathbf{x}_0}} \sum_{i=1}^{N_{\mathbf{x}_0}} \left[\rho_k^{(i)} - \widehat{\rho}\left(\mathbf{x}_k^{(i)}|t_k, \boldsymbol{\theta}\right)\right] \left[\log \rho_k^{(i)} - \log \widehat{\rho}\left(\mathbf{x}_k^{(i)}|t_k, \boldsymbol{\theta}\right)\right]. \tag{A.14}$$

This provides a statistically-relevant measure of generalization accuracy for our model for each sample time t_k. We have also experimented with using (A.14) as a training loss function but have found the MSE loss (A.11) to work better in practice. We should point out that the MC approximation (A.14) should in principal use uniform samples in space, but for practical reasons we use the test set $\mathcal{D}_{\text{test}}$ which has samples distributed according to $\rho\left(\mathbf{x}|t\right)$.

A.3.3 Physics-informed learning

In Chapter 2 we saw that we could significantly improve data-efficiency in training by augmenting the regression loss with an additional term which encourages the NN to learn the value gradient. Unfortunately in this context we do not have an analogous gradient data source. Thus for physically-motivated regularization we turn to the least squares methods proposed by [116, 129]. Simply put, we would like $\widehat{\rho}(\cdot)$ to satisfy the Liouville equation (A.3).

To this end we seek to minimize the PDE residual in the L^2 sense:

$$\begin{aligned} &\left\|\widehat{\rho}_t\left(\mathbf{x}|t, \boldsymbol{\theta}\right) + \nabla_{\mathbf{x}} \cdot \left[\widehat{\rho}\left(\mathbf{x}|t, \boldsymbol{\theta}\right) \mathbf{f}(\mathbf{x})\right]\right\|_{L^2}^2 \\ &= \int_{\mathbb{R}^n} \int_0^{t_f} \left(\widehat{\rho}_t\left(\mathbf{x}|t, \boldsymbol{\theta}\right) + \nabla_{\mathbf{x}} \cdot \left[\widehat{\rho}\left(\mathbf{x}|t, \boldsymbol{\theta}\right) \mathbf{f}(\mathbf{x})\right]\right)^2 dt d\mathbf{x}, \end{aligned} \tag{A.15}$$

which of course is zero if $\widehat{\rho} \equiv \rho$. In practice we approximate (A.15) by MC integration over a set of collocation points

$$\mathcal{C} \coloneqq \left\{\mathbf{x}^{(i)}, t^{(i)}\right\}_{i=1,\ldots,N_{\text{col}}}, \qquad t^{(i)} \sim \mathcal{U}\left[0, t_f\right], \qquad \mathbf{x}^{(i)} \sim \mathcal{U}\left(\mathbb{X}\left(t^{(i)}\right)\right). \tag{A.16}$$

Here $\mathbb{X}(t)$ is the spatial computational domain at time t, typically a convex hull of the approximate support of the PDF. For a simple overapproximation we can take $\mathbb{X}$ as a (time-varying) hypercube enclosing the sample trajectories $\mathcal{D}$. Note that collocation points can include include the training data as well as randomly sampled points which require no numerical integration to generate. This approximation yields the physics-informed loss function

$$\underset{\mathcal{L}}{\text{loss}}(\boldsymbol{\theta}; \mathcal{C}) \coloneqq \frac{1}{N_{\text{col}}} \sum_{i=1}^{N_{\text{col}}} \left(\widehat{\rho}_t\left(\mathbf{x}^{(i)} | t^{(i)}, \boldsymbol{\theta}\right) + \nabla_{\mathbf{x}} \cdot \left[\widehat{\rho}\left(\mathbf{x}^{(i)} | t^{(i)}, \boldsymbol{\theta}\right) \mathbf{f}\left(\mathbf{x}^{(i)}\right)\right]\right)^2. \tag{A.17}$$

As with our value functions NNs, the partial derivatives of $\widehat{\rho}(\cdot)$ appearing in (A.17) can be calculated using automatic differentiation.

Approximating (A.3) at collocation points can reduce the need for numerical integration, as well as yielding a representation of the PDF which is guided by the underlying physics. The main difference between our method and least squares approaches is that we exploit the ability to generate data along the characteristics of the PDF. We find that this two-pronged strategy is more effective than direct minimization of the PDE residual and boundary conditions.

We now introduce the physics-informed learning problem which we use to train PDF models:

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \left\{\underset{\rho}{\text{loss}}\left(\boldsymbol{\theta}; \mathcal{D}\right) + \mu_{\mathcal{L}} \underset{\mathcal{L}}{\text{loss}}\left(\boldsymbol{\theta}; \mathcal{C}\right)\right\}. \tag{A.18}$$

Note the scalar weight $\mu_{\mathcal{L}} \geq 0$ must be carefully chosen to balance the impact

of the PDE residual term (A.17) with the regression loss (A.12), which serves as a *boundary condition* in the present context. Good choices of $\mu_{\mathcal{L}}$ are highly problem-dependent, and for some problems we have found success with increasing $\mu_{\mathcal{L}}$ over the course of training.

A.4 Example: Kraichnan-Orszag problem

To illustrate the proposed methodology we study the Kraichnan-Orszag system [106], which is a canonical test problem in uncertainty quantification [142, 24, 18]. The state dynamics are

$$\begin{cases} \dot{x}_1 = x_1 x_3, \\ \dot{x}_2 = -x_2 x_3, \\ \dot{x}_3 = -x_1^2 + x_2^2. \end{cases} \tag{A.19}$$

We consider independently normally distributed initial conditions such that the joint PDF $\rho_0(\mathbf{x}_0)$ straddles the "stochastic discontinuities" on the $x_2 = 0$ axes:

$$\begin{cases} x_1(0) = x_{1,0} \sim \mathcal{N}\left(1, 1/4^2\right), \\ x_2(0) = x_{2,0} \sim \mathcal{N}\left(0, 1/2^2\right), \\ x_3(0) = x_{3,0} \sim \mathcal{N}\left(0, 1/2^2\right). \end{cases} \tag{A.20}$$

We seek to model the time-evolving PDF for $t \in [0, 10]$.

Using TensorFlow 1.11 [1], we implement a fully-connected feedforward NN with $L = 6$ hidden layers with $w = 64$ neurons each. All hidden layers use $\tanh(\cdot)$ activation functions. For training data we integrate $N_{\mathbf{x}_0} = 100$ sample trajectories evaluated at $N_t + 1 = 101$ time snapshots each, totaling 10100 training data. For testing we construct a test data set with $N_{\mathbf{x}_0} = 1000$ trajectories. To evaluate

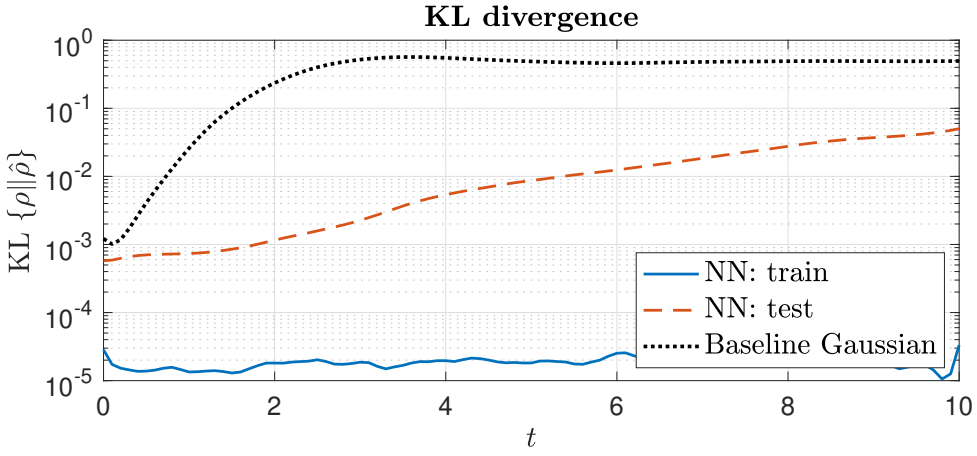


Figure A.1: Training and test error (A.14) for an NN model of a PDF (A.20) evolving according to the Kraichnan-Orszag dynamics (A.19), with respect to simulation time t. Approximation error for a multivariate normal is shown for comparison.

the PDE residual loss (A.17) we take $\mathcal{C}$ as the union of $\mathcal{D}_{\text{train}}$ and an additional 90900 collocation points uniformly sampled from a cube containing the trajectory data. We set $\mu_{\mathcal{L}} = 0.1$ and optimize using L-BFGS [85]. This implementation is thus similar to the method proposed by [116], except that simulation data replace randomly sampled boundary collocation points.

We plot the training and test error of the model with respect to simulation time t in Figure A.1. To contextualize the scale of the error, we fit a multivariate normal distribution to the test data at each time t_k, and also show the KL divergence between this and the test data for t_k. In this plot we can see that the NN model improves over this prediction by at least an order of magnitude.

In Figure A.2 we plot the predicted marginal density, $\widehat{\rho}\,(x_1, x_2|t)$, at a few time snapshots. To produce these figures we use the NN to predict the joint density on a tensor grid, which we then marginalize by quadrature integration. While dense grid integration is impractical in higher dimensions, here we use it only to visualize the complexity of the PDF captured by the NN.

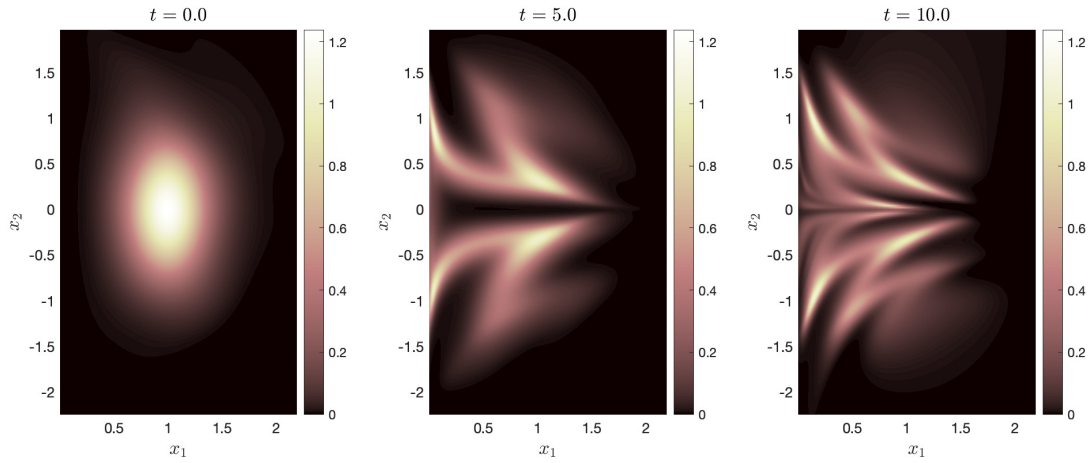


Figure A.2: Predicted marginal PDF, $\widehat{\rho}\,(x_1, x_2|t)$, of the Kraichnan-Orszag system.

A.5 Outlook

In this appendix we have discussed how the characteristics and physics-based learning methodology introduced in this dissertation for optimal feedback design can be applied to propagate uncertainty through nonlinear dynamics. This is an important open research problem, which also has close connections with optimal control [112, 122, 113, 139, 79]. The preliminary results presented here for the Kraichnan-Orszag benchmark problem suggest that the method can effectively approximate some complicated PDFs from moderate amounts of MC simulation data.

The proposed uncertainty quantification approach has a few main limitations, which are somewhat related. We note, however, that *these limitations appear to affect most methods which seeks to explictly approximate the density as a function of space and time*. These obstacles can be avoided by *generative* methods like [148] which *implicitly* learn $\rho\,(\cdot)$ and allow one to sample this distribution. The tradeoff with such methods as that the density is typically not immediately available. Still,

such generative methods may be more flexible and practically useful overall.

The first challenge is dealing with problems where the support of the PDF is very small, or collapses over time in systems which have stable manifolds. In such situations the density grows exponentially, and even when taking the logarithm and applying scaling, it is generally not possible to approximate accurately. It is typically possible to judge whether the proposed method can handle a given problem based on a few MC simulations: if the density becomes extremely large then this method will not work.

The second challenge is that the model must be re-trained for new initial distributions $\rho_0(\cdot)$. This difficulty can be mitigated if all initial distributions of interest can be parameterized by a (small) set of values (e.g. Gaussians). In this case we can add the parameterization of the initial distribution as an additional input to the NN model. Then as long as we know the initial distribution we can make a prediction for future densities. Furthermore, we can recycle training data by taking a single MC simulation and evolving the density by (A.5) for different parameterizations of $\rho_0(\cdot)$.

The final challenge is that the proposed method (and many others which explicitly approximate the density) does not have a native way to sample from the PDF. This presents difficulties when trying to perform integrations in high-dimensional settings. Such integrations are needed to compute expectations and other moments, as well as to marginalize out different variables. To perform high-dimensional integration with the method as presented here, one can use MCMC methods as these depend on a representation of the density.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, et al., *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2016, https://arxiv.org/abs/1603.04467.

[2] M. Abu-Khalaf and F. L. Lewis, *Nearly optimal control laws for nonlinear systems with saturating actuators using a neural network HJB approach*, Automatica, 41 (2005), pp. 779–791, https://doi.org/10.1016/j.automatica.2004.11.034.

[3] G. Albi, S. Bicego, and D. Kalise, *Gradient-augmented supervised learning of optimal feedback laws using state-dependent Riccati equations*, IEEE Control Syst. Lett., 6 (2022), pp. 836–841, https://doi.org/10.1109/LCSYS.2021.3086697.

[4] E. Al'brekht, *On the optimal stabilization of nonlinear systems*, J. Appl. Math. Mech., 25(5) (1961), pp. 1254–1266, https://doi.org/10.1016/0021-8928(61)90005-3.

[5] A. Alla, M. Falcone, and D. Kalise, *An efficient policy iteration algorithm for dynamic programming equations*, SIAM J. Sci. Comput., 37 (2015), pp. A181–A200, https://doi.org/10.1137/130932284.

[6] B. Azmi, D. Kalise, and K. Kunisch, *Optimal feedback law recovery by gradient-augmented sparse polynomial regression*, J. Mach. Learn. Res., 22 (2021), pp. 1–32.

[7] R. W. Beard and T. W. McLain, *Small Unmanned Aircraft: Theory and Practice*, Princeton University Press, Princeton, NJ, 2012.

[8] R. W. Beard and T. W. McLain, *Small Unmanned Aircraft: Theory and Practice [Supplement]*, Princeton University Press, Princeton, NJ, 2nd ed., 2022.

[9] R. W. Beard, G. N. Saridis, and J. T. Wen, *Approximate solutions to the time-invariant Hamilton–Jacobi–Bellman equation*, J. Optim. Theory Appl., 96 (1998), pp. 589–626, https://doi.org/10.1023/A:1022664528457.

[10] R. E. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, NJ, 1957.

[11] P. Benner, *Symplectic balancing of Hamiltonian matrices*, SIAM J. Sci. Comput., 22 (2001), pp. 1885–1904, https://doi.org/10.1137/S1064827500367993.

[12] A. Bensoussan, J. Han, S. C. P. Yam, and X. Zhou, *Value-gradient based formulation of optimal control problem and machine learning algorithm*, 2021, https://arxiv.org/abs/2103.09280.

[13] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, Athena Scientific, 1996.

[14] J. T. Betts, *Survey of numerical methods for trajectory optimization*, J. Guid. Control Dyna., 21 (1998), pp. 193–207, https://doi.org/10.2514/2.4231.

[15] O. Bokanowski, J. Garcke, M. Griebel, and I. Klompmaker, *An adaptive sparse grid semi-Lagrangian scheme for first order Hamilton-Jacobi Bellman equations*, J. Sci. Comput., 55 (2013), pp. 575–605, https://doi.org/10.1007/s10915-012-9648-x.

[16] J. Borggaard and L. Zietsman, *The quadratic-quadratic regulator problem: Approximating feedback controls for quadratic-in-state nonlinear systems*, in American Control Conference (ACC), 2020, pp. 818–823, https://doi.org/10.23919/ACC45564.2020.9147286.

[17] L. Bottou, F. E. Curtis, and J. Nocedal, *Optimization methods for large-scale machine learning*, SIAM Rev., 60 (2018), pp. 223–311, https://doi.org/10.1137/16M1080173.

[18] C. Brennan and D. Venturi, *Data-driven closures for stochastic dynamical systems*, J. Comput. Phys., 372 (2018), pp. 281–298, https://doi.org/10.1016/j.jcp.2018.06.038.

[19] R. H. Byrd, G. M. Chin, J. Nocedal, and Y. Wu, *Sample size selection in optimization methods for machine learning*, Math. Program., 134 (2012), pp. 127–155, https://doi.org/10.1007/s10107-012-0572-5.

[20] S. Cacace, E. Cristiani, M. Falcone, and A. Picarelli, *A patchy dynamic programming scheme for a class of Hamilton-Jacobi-Bellman equations*, SIAM J. Sci. Comput., 34 (2012), pp. A2625–A2649, https://doi.org/10.1137/110841576.

[21] G. Chen, *Deep neural network approximations for the stable manifolds of the Hamilton-Jacobi equations*, 2020, https://arxiv.org/abs/2007.15350.

[22] Z. Chen and D. Xiu, *On generalized residual network for deep learning of unknown dynamical systems*, J. Comput. Phys., 438 (2021), p. 110362, https://doi.org/10.1016/j.jcp.2021.110362.

[23] T. Cheng, F. L. Lewis, and M. Abu-Khalaf, *Fixed-final-time-constrained optimal control of nonlinear systems using neural network HJB approach*, IEEE Trans. Neural Netw., 18 (2007), pp. 1725–1737, https://doi.org/10.1109/TNN.2007.905848.

[24] H. Cho, D. Venturi, and G. E. Karniadakis, *Numerical methods for high-dimensional probability density function equations*, J. Comput. Phys., 305 (2016), pp. 817–837, https://doi.org/10.1016/j.jcp.2015.10.030.

[25] Y. T. Chow, J. Darbon, S. Osher, and W. Yin, *Algorithm for overcoming the curse of dimensionality for state-dependent Hamilton-Jacobi equations*, J. Comput. Phys., 387 (2019), pp. 376–409, https://doi.org/10.1016/j.jcp.2019.01.051.

[26] M. G. Crandall and P.-L. Lions, *Viscosity solutions of Hamilton-Jacobi equations*, Trans. Amer. Math. Soc., 277 (1983), pp. 1–42, https://doi.org/10.2307/1999343.

[27] P. E. Crouch, *Spacecraft attitude control and stabilization: Applications of geometric control theory to rigid body models*, IEEE Trans. Automat. Control, 29 (1984), pp. 321–331, https://doi.org/10.1109/TAC.1984.1103519.

[28] J. Darbon, G. P. Langlois, and T. Meng, *Overcoming the curse of dimensionality for some Hamilton–Jacobi partial differential equations via neural network architectures*, Res. Math. Sci., 7 (2020), pp. 1–50, https://doi.org/10.1007/s40687-020-00215-6.

[29] J. Darbon and T. Meng, *On some neural network architectures that can represent viscosity solutions of certain high dimensional Hamilton–Jacobi partial differential equations*, J. Comput. Phys., 425 (2021), p. 109907, https://doi.org/10.1016/j.jcp.2020.109907.

[30] J. Darbon and S. Osher, *Algorithms for overcoming the curse of dimensionality for certain Hamilton-Jacobi equations arising in control theory and elsewhere*, Res. Math. Sci., 3 (2016), https://doi.org/10.1186/s40687-016-0068-7.

[31] K. J. DeMars, R. H. Bishop, and M. K. Jah, *Entropy-based approach for uncertainty propagation of nonlinear dynamical systems*, J. Guid. Control Dyna., 36 (2013), pp. 1047–1057, https://doi.org/10.2514/1.58987.

[32] J. Diebel, *Representing attitude: Euler angles, unit quaternions, and rotation vectors*, 2006, https://www.astro.rug.nl/software/kapteyn-beta/_downloads/attitude.pdf (accessed 2020-05-16).

[33] S. Dolgov, D. Kalise, and K. K. Kunisch, *Tensor decomposition methods for high-dimensional Hamilton–Jacobi–Bellman equations*, SIAM J. Sci. Comput., 43 (2021), pp. A1625–A1650, https://doi.org/10.1137/19M1305136.

[34] S. Dolgov, D. Kalise, and L. Saluzzi, *Data-driven tensor train gradient cross approximation for Hamilton-Jacobi-Bellman equations*, 2022, https://arxiv.org/abs/2205.05109.

[35] J. Drgoňa, K. Kiš, A. Tuor, D. Vrabie, and M. Klaučo, *Deep learning alternative to explicit model predictive control for unknown nonlinear systems*, 2020, https://arxiv.org/abs/2011.03699.

[36] M. Ehrendorfer, *The Liouville equation and its potential usefulness for the prediction of forecast skill. part i: Theory*, Monthly Weather Rev., 122 (1994), pp. 703–713, https://doi.org/10.1175/1520-0493(1994)122<0703:TLEAIP>2.0.CO;2.

[37] M. Eigel, R. Schneider, and D. Sommer, *Dynamical low-rank approximations of solutions to the Hamilton-Jacobi-Bellman equation*, 2021, https://arxiv.org/abs/2111.14540.

[38] G. H. Elkaim, *Lecture notes for ECE 263: UAV Modeling and Control*. University of California, Santa Cruz, 2021.

[39] L. Erlenbach, *Active learning for Bayesian neural networks with Gaussian processes*, master's thesis, Institut für Numerische Simulation, Universität Bonn, 2020.

[40] G. Evensen, *Data Assimilation: The Ensemble Kalman Filter*, Springer, Berlin-Heidelberg, 2nd ed., 2009, https://doi.org/0.1007/978-3-642-03711-5.

[41] F. Fahroo and I. M. Ross, *Pseudospectral methods for infinite-horizon nonlinear optimal control problems*, J. Guid. Control Dyna., 31 (2008), pp. 927–936, https://doi.org/10.2514/1.33117.

[42] M. Falcone and R. Ferretti, *Semi-Lagrangian Approximation Schemes for Linear and Hamilton-Jacobi Equations*, Society for Industrial and Applied Mathematics, 2013, https://doi.org/10.1137/1.9781611973051.

[43] J. Foo, X. Wan, and G. E. Karniadakis, *The multi-element probabilistic collocation method (ME-PCM): Error analysis and applications*, J. Comput. Phys., 227 (2008), pp. 9572–9595, https://doi.org/10.1016/j.jcp.2008.07.009.

[44] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, Pearson Higher Education, Upper Saddle River, NJ, 7th ed., 2015.

[45] S. Frey, C. Colombo, and S. Lemmens, *Application of density-based propagation to fragment clouds using the Starling Suite*, in First International Orbital Debris Conference, no. 6089, 2019, pp. 1–10.

[46] D. Garg, W. W. Hager, and A. V. Rao, *Pseudospectral methods for solving infinite-horizon optimal control problems*, Automatica, 47 (2011), pp. 829–837, https://doi.org/10.1016/j.automatica.2011.01.085.

[47] L. Grüne, *Computing Lyapunov functions using deep neural networks*, J. Comput. Dyna., 8 (2021), pp. 131–152, https://doi.org/10.3934/jcd.2021006.

[48] S. Haddad, K. F. Caluya, A. Halder, and B. Singh, *Prediction and optimal feedback steering of probability density functions for safe automated driving*, in American Control Conference (ACC), 2021, pp. 2956–2961, https://doi.org/10.23919/ACC50511.2021.9482927.

[49] A. Halder and R. Bhattacharya, *Dispersion analysis in hypersonic flight during planetary entry using stochastic Liouville equation*, J. Guid. Control Dyna., 34 (2011), pp. 459–474, https://doi.org/10.2514/1.51196.

[50] A. Halder and E. D. Wendel, *Finite horizon linear quadratic Gaussian density regulator with Wasserstein terminal cost*, in American Control Conference (ACC), 2016, pp. 7249–7254, https://doi.org/10.1109/ACC.2016.7526817.

[51] J. Han and W. E, *Deep learning approximation for stochastic control problems*, 2016, https://arxiv.org/abs/1611.07422.

[52] J. Han, A. Jentzen, and W. E, *Solving high-dimensional partial differential equations using deep learning*, Proc. Natl. Acad. Sci. USA, 115 (2018), pp. 8505–8510, https://doi.org/10.1073/pnas.1718942115.

[53] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger, *Learning-based model predictive control: Toward safe learning in control*, Ann. Rev. Control Robot. Auton. Syst., 3 (2020), pp. 269–296, https://doi.org/10.1146/annurev-control-090419-075625.

[54] K. Hornik, *Approximation capabilities of multilayer feedforward networks*, Neural Netw., 4 (1991), pp. 251–257, https://doi.org/10.1016/0893-6080(91)90009-T.

[55] D. Izzo and E. Öztürk, *Real-time guidance for low-thrust transfers using deep neural networks*, J. Guid. Control Dyna., (2021), pp. 1–13, https://doi.org/10.2514/1.G005254.

[56] D. Izzo, E. Öztürk, and M. Märtens, *Interplanetary transfers via deep representations of the optimal policy and/or of the value function*, in Genetic and Evolutionary Computation Conference, 2019, pp. 1971—1979, https://doi.org/10.1145/3319619.3326834.

[57] D. Izzo, D. Tailor, and T. Vasileiou, *On the stability analysis of deep neural network representations of an optimal state-feedback*, IEEE Trans. Aerosp. Electron. Syst., 57 (2021), pp. 145–154, https://doi.org/10.1109/TAES.2020.3010670.

[58] F. Jiang, G. Chou, M. Chen, and C. J. Tomlin, *Using neural networks to compute approximate and guaranteed feasible Hamilton-Jacobi-Bellman PDE solutions*, 2016, https://arxiv.org/abs/1611.03158.

[59] B. Johnson, M. Gomez, and S. B. Munch, *Leveraging spatial information to forecast nonlinear ecological dynamics*, Methods Ecol. Evol., 12 (2021), pp. 266–279, https://doi.org/10.1111/2041-210X.13511.

[60] S. J. Julier and J. K. Uhlmann, *New extension of the Kalman filter to nonlinear systems*, in Signal Processing, Sensor Fusion, and Target Recognition VI, vol. 3068, 1997, pp. 182–193, https://doi.org/10.1117/12.280797.

[61] D. Kalise, S. Kundu, and K. Kunisch, *Robust feedback control of nonlinear PDEs by numerical approximation of high-dimensional Hamilton–Jacobi–Isaacs equations*, SIAM J. Appl. Dyn. Syst., 19 (2020), pp. 1496–1524, https://doi.org/10.1137/19M1262139.

[62] D. Kalise and K. Kunisch, *Polynomial approximation of high-dimensional Hamilton-Jacobi-Bellman equations and applications to feedback control of semilinear parabolic PDEs*, SIAM J. Sci. Comput., 40 (2018), pp. A629–A652, https://doi.org/10.1137/17M1116635.

[63] W. Kang, P. De, and A. Isidori, *Flight control in a windshear via nonlinear H_∞ methods*, in Proceedings of the 31st IEEE Conference on Decision and Control, vol. 1, 1992, pp. 1135–1142.

[64] W. Kang, Q. Gong, T. Nakamura-Zimmerer, and F. Fahroo, *Algorithms of data generation for deep learning and feedback design: A survey*, Phys. D, (2021), p. 132955, https://doi.org/10.1016/j.physd.2021.132955.

[65] W. Kang, K. Sun, and L. Xu, *Data-driven computational methods for the domain of attraction and Zubov's equation*, https://arxiv.org/abs/2112.14415.

[66] W. Kang and L. C. Wilcox, *A causality free computational method for HJB equations with application to rigid body satellites*, in AIAA Guidance, Navigation, and Control Conference, 2015, pp. 1–10, https://doi.org/10.2514/6.2015-2009.

[67] W. Kang and L. C. Wilcox, *Mitigating the curse of dimensionality: Sparse grid characteristics method for optimal feedback control and HJB equations*, Comput. Optim. Appl., 68 (2017), pp. 289–315, https://doi.org/10.1007/s10589-017-9910-0.

[68] W. Kang and L. C. Wilcox, *Solving 1D conservation laws using Pontryagin's minimum principle*, J. Sci. Comput., 71 (2017), pp. 144–165, https://doi.org/10.1007/s10915-016-0294-6.

[69] W. Kang, O. Yakimenko, and L. Wilcox, *Optimal control of UAVs using the sparse grid characteristic method*, in 3rd International Conference on Control, Automation and Robotics (ICCAR), 2017, pp. 771–776, https://doi.org/10.1109/ICCAR.2017.7942802.

[70] M. Kelly, *An introduction to trajectory optimization: How to do your own direct collocation*, SIAM Rev., 59 (2017), pp. 849–904, https://doi.org/10.1137/16M1062569.

[71] H. Khalil, *Nonlinear Systems*, Prentice Hall, Upper Saddle River, NJ, 3rd ed., 2002.

[72] J. Kierzenka and L. F. Shampine, *A BVP solver based on residual control and the MATLAB PSE*, ACM Trans. Math. Softw., 27 (2001), pp. 299–316, https://doi.org/10.1145/502800.502801.

[73] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014, https://arxiv.org/abs/1412.6980.

[74] D. Kraft, *A software package for sequential quadratic programming*, Tech. Report DFVLR-FB 88-28, Deutsche Forschungs und Versuchsanstalt für Luft und Raumfahrt, 1989.

[75] A. Krishnapriyan, A. Gholami, S. Zhe, R. Kirby, and M. W. Mahoney, *Characterizing possible failure modes in physics-informed neural networks*, in Advances in Neural Information Processing Systems, vol. 34, 2021, pp. 26548–26560.

[76] S. Kullback and R. Leibler, *On information and sufficiency*, Ann. Math. Statist., 22 (1951), pp. 79–86, https://doi.org/10.1214/aoms/1177729694.

[77] S. Kundu and K. Kunisch, *Policy iteration for Hamilton–Jacobi–Bellman equations with control constraints*, Comput. Optim. Appl., (2021), https://doi.org/10.1007/s10589-021-00278-3.

[78] K. Kunisch and D. Walter, *Semiglobal optimal feedback stabilization of autonomous systems via deep neural network approximation*, ESAIM Control Optim. Calc. Var., 27 (2021), p. 59, https://doi.org/10.1051/cocv/2021009.

[79] P. Lambrianides, Q. Gong, and D. Venturi, *A new scalable algorithm for computational optimal control under uncertainty*, J. Comput. Phys., 420 (2020), p. 109710, https://doi.org/10.1016/j.jcp.2020.109710.

[80] A. J. Laub, *A Schur method for solving algebraic Riccati equations*, IEEE Trans. Automat. Control, 24 (1979), pp. 913–921, https://doi.org/10.1109/TAC.1979.1102178.

[81] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, Nature, 521 (2015), pp. 436–444, https://doi.org/10.1038/nature14539.

[82] S. Li, E. Öztürk, C. D. Wagter, G. C. H. E. de Croon, and D. Izzo, *Aggressive online control of a quadrotor via deep network representations of optimality principles*, in IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 6282–6287, https://doi.org/10.1109/ICRA40945.2020.9197443.

[83] D. Liberzon, *Calculus of Variations and Optimal Control Theory: A Concise Introduction*, Princeton University Press, Princeton, NJ, 2011, https://doi.org/10.2307/j.ctvcm4g0s.

[84] D. Liu, Q. Wei, D. Wang, X. Yang, and H. Li, *Adaptive Dynamic Programming with Applications in Optimal Control*, Advances in Industrial Control, Springer, Cham, Switzerland, 2017, https://doi.org/10.1007/978-3-319-50815-3.

[85] D. C. Liu and J. Nocedal, *On the limited memory BFGS method for large scale optimization*, Math. Program., 45 (1989), pp. 503–528, https://doi.org/10.1007/BF01589116.

[86] M. Locatelli and F. Schoen, *Global Optimization*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2013, https://doi.org/10.1137/1.9781611972672.

[87] D. M. Luchtenburg, S. L. Brunton, and C. W. Rowley, *Long-time uncertainty propagation using generalized polynomial chaos and flow map composition*, J. Comput. Phys., 274 (2014), pp. 783–802, https://doi.org/10.1016/j.jcp.2014.06.029.

[88] D. Lukes, *Optimal regulation of nonlinear dynamical systems*, SIAM J. Control, 7 (1969), pp. 75–100, https://doi.org/10.1137/0307007.

[89] B. Lusch, J. N. Kutz, and S. L. Brunton, *Deep learning for universal linear embeddings of nonlinear dynamics*, Nature Commun., 9 (2018), p. 4950, https://doi.org/10.1038/s41467-018-07210-0.

[90] M. Lutter, B. Belousov, K. Listmann, D. Clever, and J. Peters, *HJB optimal feedback control with deep differential value functions and action constraints*, in Proceedings of the Conference on Robot Learning, vol. 100, Proceedings of Machine Learning Research, 2020, pp. 640–650.

[91] G. Manek and J. Z. Kolter, *Learning stable deep dynamics models*, in Advances in Neural Information Processing Systems, no. 998, 2019, pp. 11128—11136.

[92] O. L. Mangasarian, *Sufficient conditions for the optimal control of nonlinear systems*, SIAM J. Control, 4 (1966), pp. 139–152, https://doi.org/10.1137/0304013.

[93] T. Meng, Z. Zhang, J. Darbon, and G. M. Karniadakis, *SympOCnet: Solving optimal control problems with applications to high-dimensional multi-agent path planning problems*, 2022, https://arxiv.org/abs/2201.05475.

[94] S. Mukherjee, J. Drgoňa, A. Tuor, M. Halappanavar, and D. Vrabie, *@miscDrgona2020, author = J'an Drgoňa and Karol Kiš and Aaron Tuor and Draguna Vrabie and Martin Klaučo, date-added = 2022-06-11 16:15:13 -0400, date-modified = 2022-06-11 16:19:48 -0400, eprint = 2011.03699, eprintclass = eess.SY, title = Deep Learning Alternative to Explicit Model Predictive Control for Unknown Nonlinear Systems, year = 2020 neural Lyapunov differential predictive control*, 2022, https://arxiv.org/abs/2205.10728.

[95] J. J. Murray, C. J. Cox, G. G. Lendaris, and R. Saeks, *Adaptive dynamic programming*, IEEE Trans. Syst. Man Cybernet. C (Appl. Rev.), 32 (2002), pp. 140–153, https://doi.org/10.1109/TSMCC.2002.801727.

[96] T. Nakamura-Zimmerer, Q. Gong, and W. Kang, *A causality-free neural network method for high-dimensional Hamilton-Jacobi-Bellman equations*, in American Control Conference (ACC), 2020, pp. 787–793, https://doi.org/10.23919/ACC45564.2020.9147270.

[97] T. Nakamura-Zimmerer, Q. Gong, and W. Kang, *Adaptive deep learning for high-dimensional Hamilton–Jacobi–Bellman equations*, SIAM J. Sci. Comput., 43 (2021), pp. A1221–A1247, https://doi.org/10.1137/19M1288802.

[98] T. Nakamura-Zimmerer, Q. Gong, and W. Kang, *QRnet: Optimal regulator design with LQR-augmented neural networks*, IEEE Control Syst. Lett., 5 (2021), pp. 1303–1308, https://doi.org/10.1109/LCSYS.2020.3034415.

[99] T. Nakamura-Zimmerer, Q. Gong, and W. Kang, *Neural network optimal feedback control with enhanced closed loop stability*, in American Control Conference (ACC, to appear), 2022, pp. 2373–2378, https://doi.org/10.48550/arXiv.2109.07466.

[100] T. Nakamura-Zimmerer, Q. Gong, and W. Kang, *Neural network optimal feedback control with guaranteed local stability*, 2022, https://arxiv.org/abs/2205.00394.

[101] T. Nakamura-Zimmerer, D. Venturi, Q. Gong, and W. Kang, *Density propagation with characteristics-based deep learning*, 2019, https://arxiv.org/abs/1911.09311.

[102] C. Navasca and A. J. Krener, *Solution of Hamilton Jacobi Bellman equations*, in Proceedings of the 39th IEEE Conference on Decision and Control (CDC), vol. 1, 2000, pp. 570–574, https://doi.org/10.1109/CDC.2000.912825.

[103] C. Navasca and A. J. Krener, *Patchy solutions of Hamilton-Jacobi-Bellman partial differential equations*, in Modeling, Estimation and Control, Springer, Berlin-Heidelberg, 2007, pp. 251–270, https://doi.org/10.1007/978-3-540-73570-0_20.

[104] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer Series in Operations Research and Financial Engineering, Springer, New York, NY, 2nd ed., 2006, https://doi.org/10.1007/978-0-387-40065-5.

[105] D. Onken, L. Nurbekyan, X. Li, S. W. Fung, S. Osher, and L. Ruthotto, *A neural network approach for high-dimensional optimal control*, 2021, https://arxiv.org/abs/2104.03270.

[106] S. A. Orszag and L. Bissonnette, *Dynamical properties of truncated Wiener-Hermite expansions*, Phys. Fluids, 10 (1967), pp. 2603–2613, https://doi.org/10.1063/1.1762082.

[107] S. Osher and J. A. Sethian, *Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations*, J. Comput. Phys., 79 (1988), pp. 12–49, https://doi.org/10.1016/0021-9991(88)90002-2.

[108] M. Oster, L. Sallandt, and R. Schneider, *Approximating the stationary Bellman equation by hierarchical tensor products*, 2019, https://arxiv.org/abs/1911.00279.

[109] M. Oster, L. Sallandt, and R. Schneider, *Approximating optimal feedback controllers of finite horizon control problems using hierarchical tensor formats*, 2021, https://arxiv.org/abs/2104.06108.

[110] T. Otsu, I. Maruta, and K. Fujimoto, *Globally optimal control of nonlinear input-affine systems under quadratic objective functions based on trajectory database*, in 3rd IFAC Conference on Modeling, Identification and Control of Nonlinear Systems (MICNON), vol. 54, 2021, pp. 37–42, https://doi.org/10.1016/j.ifacol.2021.10.325.

[111] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *PyTorch: An imperative style, high-performance deep learning library*, in Advances in Neural Information Processing Systems, 2019, pp. 8024–8035.

[112] C. Phelps, Q. Gong, J. O. Royset, C. Walton, and I. Kaminer, *Consistent approximation of a nonlinear optimal control problem with uncertain parameters*, Automatica, 50 (2014), pp. 2987–2997, https://doi.org/10.1016/j.automatica.2014.10.025.

[113] C. Phelps, J. O. Royset, and Q. Gong, *Optimal control of uncertain systems using sample average approximations*, SIAM J. Control Optim., 54 (2016), pp. 1–29, https://doi.org/10.1137/140983161.

[114] L. S. Pontryagin, *Mathematical Theory of Optimal Processes*, vol. 4 of L.S. Pontryagin selected works, Taylor and Francis, 1987.

[115] W. B. Powell, *Approximate Dynamic Programming: Solving the Curses of Dimensionality*, John Wiley & Sons, Inc., Hoboken, NJ, 2nd ed., 2011, https://doi.org/10.1002/9781118029176.

[116] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, J. Comput. Phys., 378 (2019), pp. 686–707, https://doi.org/10.1016/j.jcp.2018.10.045.

[117] S. M. Richards, F. Berkenkamp, and A. Krause, *The Lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems*, in 2nd Conference on Robot Learning, vol. 87, Proceedings of Machine Learning Research, 2018, pp. 466–476.

[118] I. M. Ross, *A historical introduction to the covector mapping principle*, in Astrodynamics Specialists Conference, vol. 123, 2005, pp. 1257–1278.

[119] I. M. Ross, *A Primer on Pontryagin's Principle in Optimal Control*, Collegiate Publishers, San Francisco, CA, 2nd ed., 2015.

[120] I. M. Ross, Q. Gong, F. Fahroo, and W. Kang, *Practical stabilization through real-time optimal control*, in American Control Conference (ACC), 2006, pp. 304–309, https://doi.org/10.1109/ACC.2006.1655372.

[121] I. M. Ross and M. Karpenko, *A review of pseudospectral optimal control: From theory to flight*, Annu. Rev. Control, 36 (2012), pp. 182–197, https://doi.org/10.1016/j.arcontrol.2012.09.002.

[122] I. M. Ross, R. J. Proulx, M. Karpenko, and Q. Gong, *Riemann–Stieltjes optimal control problems for uncertain dynamic systems*, J. Guid. Control Dyna., 38 (2015), pp. 1251–1263, https://doi.org/10.2514/1.G000505.

[123] I. M. Ross, P. Sekhavat, A. Fleming, and Q. Gong, *Optimal feedback control: Foundations, examples, and experimental results for a new approach*, J. Guid. Control Dyna., 31 (2008), pp. 307–321, https://doi.org/10.2514/1.29532.

[124] C. Sánchez-Sánchez and D. Izzo, *Real-time optimal control via deep neural networks: Study on landing problems*, J. Guid. Control Dyna., 41 (2018), pp. 1122–1135, https://doi.org/10.2514/1.G002357.

[125] M. Schwenzer, M. Ay, T. Bergs, and D. Abel, *Review on model predictive control: An engineering perspective*, Int. J. Adv. Manuf. Tech., 117 (2021), pp. 1327–1349, https://doi.org/10.1007/s00170-021-07682-3.

[126] D. W. Scott, *Multivariate Density Estimation: Theory, Practice, and Visualization*, John Wiley & Sons, Inc., Hoboken, NJ, 2nd ed., 2015, https://doi.org/10.1002/9780470316849.

[127] R. Shaffer, M. Karpenko, and Q. Gong, *Unscented guidance for waypoint navigation of a fixed-wing UAV*, in American Control Conference (ACC), 2016, pp. 473–478, https://doi.org/10.1109/ACC.2016.7524959.

[128] W. Sirichotiyakul, N. A. Ashenafi, and A. C. Satici, *Robust data-driven passivity-based control of underactuated systems via neural approximators and Bayesian inference*, in American Control Conference (ACC, to appear), 2022, pp. 3266–3272.

[129] J. Sirignano and K. Spiliopoulos, *DGM: A deep learning algorithm for solving partial differential equations*, J. Comput. Phys., 375 (2018), pp. 1339–1364, https://doi.org/10.1016/j.jcp.2018.08.029.

[130] R. F. Stengel, *Optimal Control and Estimation*, Dover, New York, NY, 2nd ed., 1994.

[131] P. Stinis, *Enforcing constraints for time series prediction in supervised, unsupervised and reinforcement learning*, 2019, https://arxiv.org/abs/1905.07501.

[132] M. H. Stone, *The generalized Weierstrass approximation theorem*, Math. Mag., 21 (1948), pp. 167–184, https://doi.org/10.2307/3029750.

[133] D. Tailor and D. Izzo, *Learning the optimal state-feedback via supervised imitation learning*, Astrodynamics, 3 (2019), pp. 361–374, https://doi.org/10.1007/s42064-019-0054-0.

[134] Y. Tassa and T. Erez, *Least squares solutions of the HJB equation with neural network value-function approximators*, IEEE Trans. Neural Netw., 18 (2007), pp. 1031–1041, https://doi.org/10.1109/TNN.2007.899249.

[135] M. Tatang, W. Pan, R. Prinn, and G. McRae, *An efficient method for parametric uncertainty analysis of numerical geophysical model*, J. Geophys. Res., 102 (1997), pp. 21925–21932, https://doi.org/10.1029/97JD01654.

[136] L. N. Trefethen, *Spectral Methods in MATLAB*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2000, https://doi.org/10.1137/1.9780898719598.

[137] A. van der Schaft, *L_2-gain analysis of nonlinear systems and nonlinear state-feedback H_∞ control*, IEEE Trans. Automat. Control, 37 (1992), pp. 770–784, https://doi.org/10.1109/9.256331.

[138] P. Van Dooren, *A generalized eigenvalue approach for solving Riccati equations*, SIAM J. Sci. Stat. Comput., 2 (1981), pp. 121–135, https://doi.org/10.1137/0902010.

[139] D. Venturi and Q. Gong, *Data-driven computational optimal control for uncertain nonlinear systems*, Tech. Report AFRL-RQ-WP-TR-2019-0197, Air Force Research Laboratory, 2019.

[140] P. Virtanen, R. Gommers, T. E. Oliphant, and et. al., *SciPy 1.0: Fundamental algorithms for scientific computing in Python*, Nat. Methods, 17 (2020), pp. 261–272, https://doi.org/10.1038/s41592-019-0686-2.

[141] V. Vittaldev, R. P. Russell, and R. Linares, *Spacecraft uncertainty propagation using Gaussian mixture models and polynomial chaos expansions*, J. Guid. Control Dyna., 39 (2016), pp. 2615–2626, https://doi.org/10.2514/1.G001571.

[142] X. Wan and G. E. Karniadakis, *Multi-element generalized polynomial chaos for arbitrary probability measures*, SIAM J. Sci. Comput., 28 (2006), pp. 901–928, https://doi.org/10.1137/050627630.

[143] S. Wang, S. Sankaran, and P. Perdikaris, *Respecting causality is all you need for training physics-informed neural networks*, 2022, https://arxiv.org/abs/2203.07404.

[144] S. Wang, X. Yu, and P. Perdikaris, *When and why PINNs fail to train: A neural tangent kernel perspective*, J. Comput. Phys., 449 (2022), p. 110768, https://doi.org/10.1016/j.jcp.2021.110768.

[145] P. K. Wong and M. Athans, *Closed-loop structural stability for linear-quadratic optimal systems*, IEEE Trans. Automat. Control, 22 (1977), pp. 94–99, https://doi.org/10.1109/TAC.1977.1101414.

[146] M. H. Wright, *The interior-point revolution in optimization: History, recent developments, and lasting consequences*, Bull. Amer. Math. Soc., 42 (2005), pp. 39–56, https://doi.org/0.1090/S0273-0979-04-01040-7.

[147] D. Xiu and G. E. Karniadakis, *The Wiener–Askey polynomial chaos for stochastic differential equations*, SIAM J. Sci. Comput., 24 (2002), pp. 619–644, https://doi.org/10.1137/S1064827501387826.

[148] Y. Yang and P. Perdikaris, *Adversarial uncertainty quantification in physics-informed neural networks*, J. Comput. Phys., 394 (2019), pp. 136–152, https://doi.org/10.1007/s00466-019-01718-y.

[149] I. Yegorov and P. M. Dower, *Perspectives on characteristics based curse-of-dimensionality-free numerical approaches for solving Hamilton-Jacobi equations*, Appl. Math. Optim., (2018), https://doi.org/10.1007/s00245-018-9509-6.

[150] Y. Zang, J. Long, X. Zhang, W. Hu, W. E, and J. Han, *A machine learning enhanced algorithm for the optimal landing problem*, 2022, https://arxiv.org/abs/2203.06753.

[151] C. Zhai and H. D. Nguyen, *Estimating the region of attraction for power systems using gaussian process and converse lyapunov function*, IEEE Trans. Control Syst. Tech., 30 (2022), pp. 1328–1335, https://doi.org/10.1109/TCST.2021.3098167.