HUMBOLDT-UNIVERSITÄT ZU BERLIN
MATHEMATISCH-NATURWISSENSCHAFTLICHE FAKULTÄT
INSTITUT FÜR MATHEMATIK

Solving high-dimensional Hamilton-Jacobi-Bellman Equations for Optimal Feedback Control via Adaptive Deep Learning Approach

MASTERARBEIT

zur Erlangung des akademischen Grades
Master of Science (M. Sc.)

eingereicht von: Elisa Giesecke
geboren am: 15.12.1995
geboren in: Hannover
Gutachter/innen: Prof. Dr. Falk Hante
Prof. Dr. Daniel Walter

Eingereicht am Institut für Mathematik der Humboldt-Universität zu Berlin am:
06.06.2023

Contents

List of Abbreviations

AD	**A**utomatic **D**ifferentiation
BVP	**B**oundary **V**alue **P**roblem
DL	**D**eep **L**earning
DNN	**D**eep **N**eural **N**etwork
GL	**G**eneralization **L**oss
HJB	**H**amilton-**J**acobi-**B**ellman
IVP	**I**nitial **V**alue **P**roblem
LQR	**L**inear-**Q**uadratic **R**egulator
MLP	**M**ulti-**L**ayer **P**erceptron (a. k. a. Feedforward Network)
MSE	**M**ean **S**quared **E**rror
NN	**N**eural **N**etwork
NODE	**N**eural **O**rdinary **D**ifferential **E**quation
OCP	**O**ptimal **C**ontrol **P**roblem
ODE	**O**rdinary **D**ifferential **E**quation
PDE	**P**artial **D**ifferential **E**quation
PMP	**P**ontryagin's **M**inimum **P**rinciple
RDE	**R**iccati **D**ifferential **E**quation
ReLU	**Re**ctified **L**inear **U**nit
ResNet	**Res**idual **Net**work
RMAE	**R**elative **M**ean **A**bsolute **E**rror
RML^1	**R**elative **M**ean **L^1** Error
RMSE	**R**oot **M**ean **S**quared **E**rror
SGD	**S**tochastic **G**radient **D**escent
TP	**T**raining **P**rogress

List of Symbols

a	diffusion coefficient
b	coefficient of reaction term
$b^{[l]}$	bias vector of l-th layer
c	coefficient of control term
C	convergence tolerance for RMSE criterion
Cost	cost function of learning problem
d_l	width of l-th layer, shortened to d for constant width
D	data set
$D_{\text{train}}, D_{\text{val}}, D_{\text{test}}$	training, validation and test set
E	performance measure
$E_{\text{train}}, E_{\text{val}}, E_{\text{test}}$	training, validation and test error
$F^{\text{NN}}, F^{\text{ResNet}}, F^{\text{MLP}}$	neural, residual and feedforward network function
H	Hamiltonian function
J	cost function of optimal control problem
K_{init}	number of candidate initial conditions for adaptive sampling
L	depth, i. e., number of hidden layers
$\mathcal{L}$	Lagrangian
m	dimension of control space
M	upper bound for progressive data generation
n	dimension of state space, or equiv. number of discretization points
$\mathbf{p}$	adjoint state or costate
R	reaction term, e. g., hyperbolic growth R_1 or logistic growth R_2
S	source or sink term, e. g., additive control S_1 or bilinear control S_2
t_0	initial time
t_f	final time
T	threshold for early stopping
$\mathbf{u}$	control
$\mathbf{u}^{\text{NN}}$	neural network control
$\mathcal{U}$	control space
V	value function
V^{NN}	neural network approximation of value function
$W^{[l]}$	weight matrix of l-th layer
$\mathbf{x}$	state, denoted as X before spatial discretization
x_0	initial condition, denoted as X_0 before spatial discretization
$\mathcal{X}$	state space
$\mathbb{X}$	semi-global domain
$\mathbb{X}_0$	initial condition domain
y	target, denoted as Y before spatial discretization
α	coefficient of running state cost
β	coefficient of running control cost
γ	coefficient of final state cost

θ	model parameters
ι	strip length for early stopping
λ	gradient regularization weight
σ	activation function
ϕ	final cost
ψ	running cost
Ω	spatial domain

Chapter 1

Introduction

1.1 Motivation

Optimal control of non-linear dynamical systems is arising in various fields such as physics, engineering and finance. To name only a few examples, applications include controlling a rigid-body satellite equipped with momentum wheels [36, Secs. 5.1 and 5.2; 49, Sec. 5], navigating the interplanetary transfers with a spacecraft [32], path-planning of autonomous or unmanned vehicles [33, Sec. 6], as well as developing an investment strategy in safe or risky assets to maximize an investor's utility [25; 56]. However, optimal control problems (OCPs) of systems governed by non-linear ordinary differential equations (ODEs) continue to be a major challenge and an active subject of research. While open-loop optimal controls are commonly feasible to compute via the Pontryagin's Minimum Principle (PMP), see, for example, [55, Sec. 2.2.2; 44, Ch. 4; 6, Sec. 2.1.3; 7, Ch. 3], they are not robust to model uncertainty or disturbances of the dynamics. Moreover, the open-loop system needs to be solved all over again when the initial condition is changed. These drawbacks motivate to seek closed-loop controllers, also known as feedback controllers, for most applications.

Based on the dynamic programming method, the feedback control law involves the value function which can be characterized as the solution of the Hamilton–Jacobi–Bellman (HJB) equation, see, e. g., [44, Ch. 5; 57, Ch. 8; 3]. This is a partial differential equation (PDE) whose spatial dimension is equal to that of the state space. Once the solution to the HJB equation is known, the optimal control in feedback form can be implemented in real-time with minimum computational costs.

However, this approach suffers under the curse of dimensionality, a term coined by Bellman in [4]. This means that solving the HJB equation becomes extremely difficult for general high-dimensional non-linear systems. Traditional methods that typically involve discretizing the state space become computationally intractable because the size of the discretized problem scales exponentially with the state dimension n. For dense grids the size of the resulting problem increases on the order of $\mathcal{O}(N^n)$ where N is the number of grid points in each dimension. But even for approaches relying on sparse grids considered in [36, Sec. 3; 22, Sec. 4; 5, Sec. 2] only moderately large systems can be handled since the number of grid points grows like $\mathcal{O}(N(\log N)^{n-1})$.

In the special case of linear dynamics and quadratic costs without control or state constraints, the HJB equation simplifies to a Riccati differential equation (RDE) whose solution has been treated thoroughly from an analytical and numerical perspective, see, e. g., [55, Secs. 2.1 and 2.4; 44, Sec. 6.1; 57, Sec. 8.2; 43, Ch. 1]. The optimal feedback controller is then provided by the linear–quadratic regulator (LQR) resulting from the solution of the RDE. Thus, for non-linear systems, one could obtain

an approximate optimal feedback solution by applying the LQR-based approach locally to the linear-quadratic approximation of the problem, see [41, Sec. 8]. However, computing linearization based feedback laws over large spatial domains and longer time intervals may become inaccurate.

Consequently, other methods are required to overcome the difficulties of numerically solving HJB equations for non-linear dynamics which have been investigated extensively in the literature on optimal control theory. Early attempts have been made in [1; 46] based on power series expansions. Furthermore, polynomial approximations of the HJB equations were proposed in [34]. Other methods focus on Hopf formulas, e. g., [16; 15; 45] and the method of characteristics [60; 36] where the latter cited paper combines the characteristics method with polynomial interpolation. Besides that, there are approaches using semi-Lagrangian schemes such as [19; 5]. In [10; 50] the domain of computation is divided into subdomains which leads to patchy dynamic programming. Unfortunately, all of these classical approaches come with at least one of the following disadvantages, see [49, Sec. 1] and also [36, Sec. 1; 33, Sec. 1]:

- limited scalibility of the method allowing merely for small or moderate space dimensions
- heavy restrictions on the structure of the dynamical system
- validity of approximate solution only over small spatial domains or in the local neighborhood of some trajectory
- difficulty of verifying the accuracy of the approximate solution

In order to find semi-global solutions to HJB equations and the optimal feedback control for general non-linear systems, data-driven approaches have become increasingly popular among recent publications, see, e. g., [35, Sec. 1]. Given the success of machine learning and particularly deep learning (DL) in a wide range of complex tasks, such as image processing, speech recognition and text generation, the interest arises to apply deep neural networks (DNNs) to optimal control as well. Since DNNs have proven very effective for modelling non-linear functions with high-dimensional inputs, there is high potential in using neural network models for approximating the solution of the HJB equation. Since deep learning based methods do not rely on the discretization of the state space, i. e., are entirely mesh-free, they provide an opportunity to overcome the curse of dimensionality.

1.2 Current State of Research

When [48] introduced the idea of neural network based control in 1990 with the hope of solving OCPs in critical application domains, particularly in robotics, a new research direction was set. Since then, numerous algorithms have been proposed that leverage deep learning.

Several state-of-the-art techniques, presented for instance in [58; 13; 12], are using the method of least squares, i. e., a neural network is trained by minimizing the HJB residual in a least-squares sense over a set of sampled points. Alternatively, [33] proposes a neural network approximator of the value function in a region around a feasible trajectory which is guaranteed to be conservative. Other DNN-based methods learn the value function and sometimes also its gradient via imitation or supervised learning, see, e. g., [32; 49]. The latter approach requires data sets of open-loop solutions generated by existing algorithms surveyed in [35].

Furthermore, specialized techniques have been developed for stochastic optimal control problems. Among them are the works of [2; 31] exploring DNN approximations of optimal control policies closely related to reinforcement learning, a hot topic in the artificial intelligence research community. In this context, the control problem is recasted as the interaction between an agent, the controller, and the environment determined by the dynamics, with the negative cost considered as the reward which the agent strives to maximize.

Besides that, there is vast literature on approximating the solution to general high-dimensional PDEs with DNNs. One popular example is [56] introducing an algorithm called Deep Galerkin Method because it is similar in spirit to Galerkin methods, but with the solution approximated by a neural network instead of a linear combination of basis functions. The article also includes the application of this algorithm to the HJB PDE corresponding to the optimal control of a stochastic heat equation.

While all mentioned papers contain numerical examples to show the effectiveness of their proposed methods, only few of them consider formal convergence analysis. Therefore, [41] is an important contribution towards developing a rigorous mathematical background. Based on universal approximation properties, the existence of optimal neural networks and the convergence of the associated feedback laws are proven. These theoretical results could be achieved due to some restrictive assumptions on the OCP. However, one can argue that DL-based methods are likely to provide promising results in more general cases as well.

1.3 Contribution

In the scope of this thesis, we focus on unconstrained OCPs with finite time horizons. However, we do not impose any restrictive structure for the cost function or system dynamics, allowing for high-dimensional non-linear states and state-dependent controls. Our adaptive deep learning approach for designing feedback controllers is mainly based on the work of Nakamura-Zimmerer et al. presented in [49]. Filling in the gaps of their optimal control theoretical considerations, we particularly provide the derivation of the two-point boundary value problem (BVP) via the method of characteristics. This BVP plays a central role in computing open-loop solutions serving as training data for the neural network (NN) model which is then used for real-time feedback control. We adopt the proposed strategy of progressive data generation with adaptive sampling, but extend Nakamura-Zimmerer's algorithmic framework by incorporating state-of-the-art techniques from DL. This results in the following key advancements:

- We refine the approximator for the value function which guarantees the satisfaction of the final condition instead of directly using the NN model.
- Furthermore, we suggest a residual network (ResNet) presented in [29] in place of the basic feed-forward network in order to prevent vanishing gradients and facilitate theoretical analysis via the neural ordinary differential eqaution (NODE) approach in [11; 17].
- We also replace the second-order deterministic optimization method L-BFGS by the first-order stochastic optimizer Adam developed in [39]. Although this likely slows down convergence, we consider Adam to be an overall better choice in this context. First of all, it requires less computation and memory

and can handle situations in which training sets become very large. Moreover, noise is introduced into the optimization process since the cost gradient is estimated on different batches for each iteration. This helps to escape local minima and, thus, leads to better generalization, as investigated in [37]. As opposed to the simpler stochastic gradient descent (SGD) method, Adam provides adaptive learning rates which reduces the need for manual tuning.

- In addition to physics-informed gradient regularization proposed in [53], we use L^2 regularization to penalize large weight matrices. The resulting NN models are less complex and therefore less prone to overfitting.
- The main difference to the original approach consists in the application of the early stopping technique tested in, e. g., [52] in order to prevent computing the sample variance for the convergence test and the sample size selection rule too often. The evaluation of the early stopping criterion is not only much cheaper but also avoids data generation after each iteration when there are still no signs of overfitting. The latter aspect is particularly relevant for first-order stochastic optimization methods which may require a larger number of iterations to reach the same level of accuracy compared to a second-order deterministic method. Nevertheless, we also expect to reduce the computational costs when using early stopping in combination with the L-BFGS method by delaying expensive heuristical tests and data generation.

This work is complemented by the implementation of our algorithm for solving HJB equations arising from optimal control for various reaction-diffusion systems. The code is provided on GitHub:

`https://github.com/ElisaGiesecke/AdaDL-for-HJB-Equations.`

Using this framework, we conduct an extensive numerical analysis to demonstrate and compare the effectiveness of our NN method with traditional approaches. The experiments yield promising results even for complicated dynamical systems with non-linear states and bilinear controls, in high-dimensional problem settings and in the presence of noise, as well as for simulations beyond the training domain.

1.4 Outline

The thesis is structured as follows. In Chapter 2, the mathematical background on optimal control is presented, including the PMP and HJB theory, along with their relationship to each other. Building upon this foundation, Chapter 3 introduces the adaptive deep learning approach for solving general OCPs. In order to develop a sophisticated algorithm, we begin by constructing a neural network model that incorporates not only fundamental concepts of DL but also certain specifications tailored to the context of learning the value function. Subsequently, we establish a technique for progressive data generation in selected regions. In Chapter 4, our approach is then applied to controlled reaction-diffusion systems to test its performance on system dynamics of increasing difficulty and to assess its scalability through the discretization of the spatial domain. The analysis of the obtained experimental results is given in Chapter 5. Finally, Chapter 6 summarizes the key findings of the research, and outlines potential future directions for further investigation.

Chapter 2

Optimal Control Theory

2.1 Problem Formulation

In the following, we consider a controlled dynamical system of the form

$$\begin{cases} \dot{\mathbf{x}}(t) = f(t, \mathbf{u}(t), \mathbf{x}(t)) \quad \text{for a. a. } t \in [t_0, t_f], \\ \mathbf{x}(t_0) = x_0, \end{cases} \tag{2.1}$$

where $\mathbf{u} : [t_0, t_f] \to \mathbb{R}^m$ is the control and $\mathbf{x} : [t_0, t_f] \to \mathbb{R}^n$ is the state and the vector field $f : [t_0, t_f] \times \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^n$ describes the dynamics over a fixed time interval ranging from the initial time $t_0 \in \mathbb{R}$ to the final time $t_f \in \mathbb{R}$ with a given initial condition $x_0 \in \mathbb{R}^n$, see [49, Sec. 2; 35, Sec. 1] and also [44, Sec. 3.3.1]. Hence, this general formulation allows for a non-linear and non-autonomous ordinary differential equation (ODE), state-dependent controls, in particular bilinear controls, as well as high-dimensional states.

We further introduce the function spaces for the control and the state as proposed in [6, Sec. 2.1.1]. The control space is a Lebesgue space which is a special case of the more general Bochner spaces, cf. [41, Sec. 2], and is defined by

$$\mathcal{U} := L^\infty(t_0, t_f; \mathbb{R}^m) := \{\mathbf{u} : [t_0, t_f] \to \mathbb{R}^m \,|\, \|\mathbf{u}\|_{L^\infty(t_0, t_f; \mathbb{R}^m)} < \infty\} \tag{2.2}$$

with its associated norm

$$\|\mathbf{u}\|_{L^\infty(t_0, t_f; \mathbb{R}^m)} := \operatorname*{ess\,sup}_{t \in [t_0, t_f]} \|\mathbf{u}(t)\| .$$

Thus, $\mathcal{U}$ contains all measurable functions bounded almost everywhere. Furthermore, it is a Banach space equipped with the norm given by the essential supremum of some norm in $\mathbb{R}^n$. Since all norms in $\mathbb{R}^n$ are equivalent, the choice of $\|\cdot\|$ is irrelevant. Furthermore, the state space is the Sobolev space defined by

$$\mathcal{X} := W^{1,\infty}(t_0, t_f; \mathbb{R}^n) := \{\mathbf{x} \in L^\infty(t_0, t_f; \mathbb{R}^n) | \dot{\mathbf{x}} \in L^\infty(t_0, t_f; \mathbb{R}^n)\} \tag{2.3}$$

and is endowed with the norm

$$\|\mathbf{x}\|_{W^{1,\infty}(t_0, t_f; \mathbb{R}^n)} := \|\mathbf{x}\|_{L^\infty(t_0, t_f; \mathbb{R}^n)} + \|\dot{\mathbf{x}}\|_{L^\infty(t_0, t_f; \mathbb{R}^n)} .$$

Furthermore, we assume the dynamics f to be continuous, i. e., $f \in \mathcal{C}^0(\mathbb{R} \times \mathbb{R}^m \times \mathbb{R}^n)$. In order to guarantee the existence of a unique solution to the initial value problem (IVP) given by (2.1), we also require f to satisfy a certain Lipschitz condition stated in the theorem below.

Theorem 2.1. *Let f be continuous and Lipschitz continuous w. r. t. the state variable x unifomly in the time t and control variable u with sufficiently small Lipschitz constant. More precisely, we assume that*

$$\begin{aligned}&\forall\, t \in [t_0, t_f], u \in \mathbb{R}^m \;\exists\, 0 < L < \frac{1}{t_f - t_0} \;\; s.\,t.\; \forall\, x_1, x_2 \in \mathbb{R}^n : \\ &\|f(t, u, x_1) - f(t, u, x_2)\| \leq L \|x_1 - x_2\|\,.\end{aligned}$$

Then, for every control $\mathbf{u} \in \mathcal{U}$, there exists a unique solution $\mathbf{x} \in \mathcal{X}$ to the IVP (2.1).

The proof uses similar arguments as the one for the Picard-Lindelöf theorem, which can be found in, e. g., [54, Satz 1.5; 55, Thm. B.1.2 and Lem. B.1.3]. It is based on the Banach fixed-point theorem using a suitable contraction mapping whose unique fixed-point solves the IVP (2.1). Adapting this proof to the given setting, we require an upper bound on the Lipschitz constant in order to ensure existence and uniqueness of the solution on the entire interval $[t_0, t_f]$. We also note that the Lipschitz continuity can be derived from assuming f to be continuously differentiable in x with the partial derivatives $f_x := \frac{\partial f}{\partial x}$ continuous in all variables (t, u, x), see [55, Lem. B.1.2].

In the scope of this thesis, we solely focus on problems of the following form.

Problem 2.2. The fixed-time free-endpoint optimal control problem (OCP) is given by

$$\begin{aligned}\min_{(\mathbf{u},\mathbf{x}) \in \mathcal{U} \times \mathcal{X}} J(\mathbf{u}, \mathbf{x}) &:= \int_{t_0}^{t_f} \psi(t, \mathbf{u}(t), \mathbf{x}(t))\, \mathrm{d}t + \phi(\mathbf{x}(t_f)) \\ \text{subject to} \quad \dot{\mathbf{x}}(t) &= f(t, \mathbf{u}(t), \mathbf{x}(t)) \quad \text{for a. a. } t \in [t_0, t_f]; \quad \mathbf{x}(0) = x_0; \\ \mathbf{u}(t) &\in U \quad \text{for a. a. } t \in [t_0, t_f],\end{aligned} \tag{2.4}$$

where the cost function $J : \mathcal{U} \times \mathcal{X} \to \mathbb{R}$ for the controlled system consists of the running cost $\psi : [t_0, t_f] \times \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}$ and the final cost $\phi : \mathbb{R}^n \to \mathbb{R}$, and $U \subseteq \mathbb{R}^m$ is the control set, see, e. g., [49, Sec. 2; 6, Sec. 2.1.1] and also [55, Sec. 2.2.1].

We assume that both ψ and ϕ are non-negative and continuous. The state evolves according to the dynamical system (2.1) satisfying all conditions mentioned above. Besides covering the non-autonomous case with the dynamics and running cost depending explicitly on time, this problem formulation may also involve nonlinearities of the dynamics. The OCP does not incorporate any further constraints on the state variable, but requires the control to take values in some non-empty subset U of $\mathbb{R}^m$. In order to simplify notation in the subsequent considerations, we define the feasible set

$$F_{ad} := \{(\mathbf{u}, \mathbf{x}) \in \mathcal{U}_{ad} \times \mathcal{X} | (\mathbf{u}, \mathbf{x}) \text{ satisfies (2.1)}\} \tag{2.5}$$

where

$$\mathcal{U}_{ad} := \{\mathbf{u} \in \mathcal{U} | \mathbf{u}(t) \text{ for a. a. } t \in [t_0, t_f]\} \tag{2.6}$$

is the admissible set containing the control constraints. We will discuss under which conditions the existence of a solution to the OCP in (2.4) can be shown.

Step 1: First, we note that F_{ad} is not empty since U is non-empty by assumption and (2.1) has a solution by Theorem 2.1. The cost function J is bounded from below by zero since we assumed $\psi, \phi \geq 0$. Then, there exists an infimizing sequence

$(\mathbf{u}_k, \mathbf{x}_k) \subset F_{ad}$ such that

$$\lim_{k\to\infty} J(\mathbf{u}_k, \mathbf{x}_k) = \inf_{(\mathbf{u},\mathbf{x})\in F_{ad}} J(\mathbf{u}, \mathbf{x}) =: J^*.$$

Step 2: Either U is bounded or we need a coercivity hypothesis on ψ of the type

$$\exists\, \beta \in \mathbb{R}, \alpha > 0 \text{ s.t. } \forall\, t \in [t_0, t_f], u \in \mathbb{R}^m, x \in \mathbb{R}^n : \psi(t, u, x) \geq \alpha \|u\|^2 - \beta,$$

cf. [6, Rem. 2.1]. In the latter case, we find some $K \in \mathbb{N}$ such that

$$J(\mathbf{u}_K, \mathbf{x}_K) \geq J(\mathbf{u}_k, \mathbf{x}_k) \geq (t_f - t_0)\left(\alpha \|\mathbf{u}_k\|_\infty^2 - \beta\right) \quad \text{for all } k \geq K.$$

Consequently, $(\mathbf{u}_k)$ is a bounded sequence in $\mathcal{U} = L^\infty(t_0, t_f; \mathbb{R}^m)$ which is the dual space of the separable Banach space $L^1(t_0, t_f; \mathbb{R}^m)$, see [8, Thm. 4.13 and 4.14]. By [8, Cor. 3.30], it has a weakly* converging subsequence $(\mathbf{u}_{k_i}) \subset \mathcal{U}$. If we assume that U is closed and convex, then $\mathcal{U}_{ad} \subseteq \mathcal{U}$ is also closed and convex and therefore weakly sequentially closed. Hence, the weak limit $\bar{\mathbf{u}}$ of $(\mathbf{u}_{k_i})$ lies in $\mathcal{U}_{ad}$.

Step 3: Theorem 2.1 ensures that the control-to-state operator $\mathbf{x}[\cdot] : \mathcal{U} \to \mathcal{X}, \mathbf{u} \mapsto \mathbf{x}[\mathbf{u}]$ mapping a control to its corresponding state is well-defined. In this framework, however, we cannot guarantee that the control-to-state operator is bounded. So, to achieve that $(\mathbf{x}_k)$ has a weakly converging subsequence $(\mathbf{x}_{k_i}) \subset \mathcal{X}$, we require stronger assumptions.

Step 4: We need to show that the weak limits $\bar{\mathbf{u}}$ and $\bar{\mathbf{x}}$ satisfy (2.1), so that $(\bar{\mathbf{u}}, \bar{\mathbf{x}}) \in F_{ad}$. We also require the cost function J to be sequentially weakly lower semicontinuous, which follows, for instance, from being continuous and convex. Indeed, J is continuous because $\psi, \phi \in \mathcal{C}^0$ by assumption. For the convexity of J, however, we have to impose further conditions on ψ and ϕ. Then, we finally get the result

$$J^* = \liminf J(\mathbf{u}_{k_i}, \mathbf{x}_{k_i}) \geq J(\bar{\mathbf{u}}, \bar{\mathbf{x}}) \geq J^* \implies J(\bar{\mathbf{u}}, \bar{\mathbf{x}}) = J^*.$$

The minimum of J subject to the given constraints is obtained in $(\bar{\mathbf{u}}, \bar{\mathbf{x}})$, which are thus the optimal control and the corresponding optimal state.

As we have seen, existence is generally difficult to establish and needs to be checked for specific cases with additional assumptions on f, ψ and ϕ. During the further considerations, we will always assume that a solution to the optimal control problem exists. Hence, we can now derive necessary and sufficient conditions for optimality which will provide the basis for the proposed numerical method of finding the optimal control in Chapter 3. Generally, there are two approaches in optimal control theory: variational methods using an adjoint state, and dynamic programming based on a value function, see, for instance, [57, Ch. 8]. While the first approach results in open-loop solutions as described in Section 2.2, the second approach leads to solutions in closed-loop or feedback form presented in Section 2.3.

2.2 Open-loop Control via Pontryagin's Minimum Principle

First, we will consider the open-loop solution to Problem 2.2 which we denote by $\mathbf{u}^*(t; x_0)$ to stress its dependency on a fixed initial condition x_0. Although we are interested in determining optimal closed-loop controls, which is ultimately the goal of this thesis, open-loop controls will still be needed in order to generate data for the supervised learning approach. The computation of candidate optimal open-loop controllers relies mainly on the Pontryagin's Minimum Principle (PMP) providing

necessary conditions for optimality. The PMP leads to a two-point boundary value problem (BVP) with a minimization condition which can be solved by a suitable technique.

For a more detailed analysis, we start by introducing the Hamiltonian formalism.

Definition 2.3. The Hamiltonian function $H : \mathbb{R} \times \mathbb{R}^m \times \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is defined by

$$H(t, u, x, p) \coloneqq \psi(t, u, x) + p \cdot f(t, u, x),$$

see, e. g., [6, Sec. 2.1.2; 7, Sec. 3.3] and also [55, Def. 2.2.3; 44, Sec. 4.1.1].

The variational approach is based on the Lagrangian $\mathcal{L} : \mathcal{U} \times \mathcal{X} \times \mathcal{X} \times \mathbb{R}^n \to \mathbb{R}$ defined by

$$\mathcal{L}(\mathbf{u}, \mathbf{x}, \mathbf{p}, q) \coloneqq J(\mathbf{u}, \mathbf{x}) + \int_{t_0}^{t_f} \mathbf{p}(t) \cdot [f(t, \mathbf{u}(t), \mathbf{x}(t)) - \dot{\mathbf{x}}(t)] \, \mathrm{d}t + q \cdot (\mathbf{x}(t_0) - x_0),$$

with Lagrange multipliers $\mathbf{p}$ and q. After reformulating the Lagrangian by plugging in the Hamiltonian function and using integration by parts, its derivative w. r. t. the state $\mathbf{x}$ in an arbitrary direction is set to zero in order to derive first-order necessary conditions, see [7, Sec. 3.3]. This leads to the equation for the adjoint state $\mathbf{p} \in \mathcal{X}$, which is also referred to as costate, given by

$$\dot{\mathbf{p}}(t) = -H_x(t, \mathbf{u}(t), \mathbf{x}(t), \mathbf{p}(t)) \quad \text{for a. a. } t \in [t_0, t_f]; \quad \mathbf{p}(t_f) = \phi_x(\mathbf{x}(t_f)). \tag{2.7}$$

Here and in the following, we use the short-hand notation for the partial derivatives written as column vectors, that is, for instance, $H_x \coloneqq \frac{\partial H}{\partial x}$.

We note that the dynamical system (2.1), also known as state equation, can be expressed in terms of the Hamiltonian, too, yielding

$$\dot{\mathbf{x}}(t) = H_p(t, \mathbf{u}(t), \mathbf{x}(t), \mathbf{p}(t)) \quad \text{for a. a. } t \in [t_0, t_f]; \quad \mathbf{x}(t_0) = x_0. \tag{2.8}$$

Now, we will state a minimum principle, pioneered by Pontryagin and his collaborators in the late 1950s and first published in [51]. For this result, we need some further regularity conditions, see [55, Sec. 2.2.1], cf. [7, Sec. 1.1.1]: Let f and ψ be continuous in (t, u, x), differentiable in x for fixed $(t, u) \in [t_0, t_f] \times U$, and the partial derivatives f_x and ψ_x be continuous as a function of all variables (t, u, x). Furthermore, we assume that ϕ is continuously differentiable and that U is a non-empty closed set in $\mathbb{R}^m$.

Theorem 2.4 (Pontryagin's minimum principle)**.** *Let $\mathbf{u}^* \in \mathcal{U}$ be a globally optimal control of the OCP* (2.4) *with initial condition x_0. Furthermore, let $\mathbf{x}^* \in \mathcal{X}$ be the corresponding optimal state trajectory and $\mathbf{p}^* \in \mathcal{X}$ the associated costate, i. e., the solutions to the Hamiltonian system*

$$\begin{cases} \dot{\mathbf{x}}(t) = H_p(t, \mathbf{u}^*(t; x_0), \mathbf{x}(t), \mathbf{p}(t)) & \text{for a. a. } t \in [t_0, t_f]; \quad \mathbf{x}(t_0) = x_0, \\ \dot{\mathbf{p}}(t) = -H_x(t, \mathbf{u}^*(t; x_0), \mathbf{x}(t), \mathbf{p}(t)) & \text{for a. a. } t \in [t_0, t_f]; \quad \mathbf{p}(t_f) = \phi_x(\mathbf{x}(t_f)). \end{cases} \tag{2.9}$$

Then the Hamiltonian minimization condition below holds:

$$\mathbf{u}^*(t) \in \operatorname*{argmin}_{u \in U} H(t, u, \mathbf{x}^*(t), \mathbf{p}^*(t)) \quad \text{for a. a. } t \in [t_0, t_f]. \tag{2.10}$$

This statement is adapted from standard formulations in [6, Thm. 2.5; Thm. 3.3 55, Thm. 2.2.1; 7; 44, Sec. 4.1.1]. Since the proof of this theorem is highly technical and does not provide insights which are significant for the approach considered in this work, it is not given here. It can be found in, for example, [44, Sec. 4.2] in full detail.

Summarizing the necessary optimality conditions given by Theorem 2.4, we note two central aspects: Firstly, it combines the state equation (2.8) with a given initial condition and the adjoint equation (2.7) with a final condition, also called transversality condition, to a two-point BVP. And secondly, it includes a minimum condition of the control. Thus, instead of optimizing over a function space, we can minimize the Hamiltonian which is much easier to do since we only need to optimize pointwise in a finite-dimensional space $U \subseteq \mathbb{R}^m$. This yields an open-loop expression for the optimal control $\mathbf{u}^*(t; x_0)$ since it depends not only on the optimal state $\mathbf{x}^*$ but also on the corresponding costate $\mathbf{p}^*$ which has to be recomputed for every initial condition x_0 via the adjoint ODE.

These necessary conditions become sufficient under certain assumptions including convexity of the cost function J w. r. t. the variables $\mathbf{u}$ and $\mathbf{x}$. We refer to, e. g., [47, Sec. 2] for a precise formulation of the setting in which sufficiency is attained. In practice, it is often hard to verify that these and all aforementioned assumptions are fulfilled globally. Nevertheless, the PMP still serves as a key tool for solving the open-loop problem numerically since it provides at least candidate optimal solutions.

2.3 Closed-loop Feedback Control via Hamilton-Jacobi-Bellman Equation

Following the alternative dynamic programming-based approach, we will now examine the optimal feedback control which is the closed-loop solution to Problem 2.2. It is denoted by $\mathbf{u}^*(t, x)$ because this control specification takes the current state x into account at all times. Hence, such feedback control laws do not rely on the knowledge of the initial condition and remain valid also in the presence of perturbations of the system. Therefore, those controls are highly desired for practical implementations and specifically their approximation via deep learning techniques seem promising. Central to their computation is the Hamilton-Jacobi-Bellman (HJB) equation. Its solution is the value function which will be modelled by a neural network in later chapters. In the HJB-theory, another minimization condition arises which is not only necessary but also sufficient for optimality.

As a starting point, we introduce the value function which can be interpreted as the optimal cost-to-go from any given state x at any given time t.

Definition 2.5. The value function $V : [t_0, t_f] \times \mathbb{R}^n \to \mathbb{R}$ is defined by

$$V(t, x) \coloneqq \inf_{(\mathbf{u}, \mathbf{x}) \in \mathcal{U}_{ad} \times \mathcal{X}} \left\{ \int_t^{t_f} \psi(t, \mathbf{u}(t), \mathbf{x}(t)) \, \mathrm{d}t + \phi(\mathbf{x}(t_f)) \right\}$$

where $\mathcal{U}_{ad}$ is the admissible control set (2.6) and the state $\mathbf{x}$ satisfies

$$\dot{\mathbf{x}}(t) = f(t, \mathbf{u}(t), \mathbf{x}(t)) \quad \text{for a. a. } t \in [t, t_f]; \quad \mathbf{x}(t) = x,$$

see, e. g., [44, Sec. 5.1.2; 57, Def. 8.1.4; 55, Def. 5.1.1] and also [3, Sec. 2.1].

We note that the infimum in the definition above turns into a minimum if an optimal control on $[t, t_f]$ with associated state trajectory starting at x exists.

The dynamic programming method divides the problem of finding an optimal control into smaller subproblems by searching for a control that minimizes the cost over a small time interval and adding the subsequent optimal cost-to-go. Bellman's principle of optimality makes this idea more rigorous, see [44, Sec. 5.1.2], cf. [57, Lem. 8.1.5]: The value function V satisfies the relation

$$\begin{aligned} &\forall\, (t,x) \in [t_0, t_f) \times \mathbb{R}^n, \Delta t \in (0, t_f - t] : \\ &V(t,x) = \inf_{\mathbf{u}|_{[t,t+\Delta t]}} \left\{ \int_t^{t+\Delta t} \psi(s, \mathbf{u}(s), \mathbf{x}(s))\, \mathrm{d}s + V(t+\Delta t, \mathbf{x}(t+\Delta t)) \right\} \end{aligned} \tag{2.11}$$

where the infimum is taken over the control $\mathbf{u}$ restricted to the interval $[t, t+\Delta t]$ and the state trajectory $\mathbf{x}$ on the right-hand side corresponds to the control $\mathbf{u}$ on this interval starting at $\mathbf{x}(t) = x$. If V is of class $\mathcal{C}^1$, the infinitesimal version of the principle of optimality yields a partial differential equation (PDE) given by

$$-V_t(t,x) = \inf_{u \in U} \{\psi(t,u,x) + V_x(t,x) \cdot f(t,x,u)\} \quad \text{for all } t \in [t_0, t_f] \text{ and } x \in \mathbb{R}^n,$$

or reformulated using the Hamiltonian from Definition 2.3

$$-V_t(t,x) = \inf_{u \in U} H(t,u,x,V_x(t,x)) \quad \text{for all } t \in [t_0, t_f] \text{ and } x \in \mathbb{R}^n. \tag{2.12}$$

This PDE was named Hamilton-Jacobi-Bellman, or briefly HJB, equation by Kalman who connected the work done by Hamilton and Jacobi, as well as Carathéodory and Bellman, in the early 1960s. Its derivation relies mainly on first-order Taylor series expansions and is provided in, for example, [44, Sec. 5.1.3; 55, Prop. 5.1.1]. The HJB equation is accompanied with a boundary condition at final time

$$V(t_f, x) = \phi(x) \quad \text{for all } x \in \mathbb{R}^n \tag{2.13}$$

which follows directly from Definition 2.5 of the value function.

Plugging in a globally optimal control $\mathbf{u}^* \in \mathcal{U}$ and its corresponding optimal state trajectory $\mathbf{x}^* \in \mathcal{X}$ into the principle of optimality (2.11) and proceeding as in [44, Sec. 5.1.3], we obtain the Hamiltonian minimization condition

$$\mathbf{u}^*(t) \in \operatorname*{argmin}_{u \in U} H(t, u, \mathbf{x}^*(t), V_x(t, \mathbf{x}^*(t))) \quad \text{for all } t \in [t_0, t_f]. \tag{2.14}$$

It is analogous to the one formulated in Theorem 2.4 as part of the PMP and has also been derived as necessary condition so far. However, minimizing the Hamiltonian leads here to an optimal closed-loop control $\mathbf{u}^*(t,x)$ because it depends solely on the actual state x at time t, provided that the value function is known. Moreover, one can show that the HJB PDE combined with the minimization condition are not only necessary but also sufficient for optimality as stated in the following theorem taken from [44, Sec. 5.1.4; 55, Thm. 5.1.1; 57, Cor. 8.5.2].

Theorem 2.6. *Consider the OCP* (2.4) *and suppose that a continuously differentiable function* $V : [t_0, t_f] \times \mathbb{R}^n \to \mathbb{R}$ *is a classical solution to the HJB equation* (2.12) *with its final condition* (2.13)*, i. e., satisfying*

$$\begin{cases} -V_t(t,x) = \inf_{u \in U} H(t,u,x,V_x(t,x)) & \text{for all } t \in [t_0, t_f] \text{ and } x \in \mathbb{R}^n, \\ V(t_f, x) = \phi(x) & \text{for all } x \in \mathbb{R}^n. \end{cases}$$

Furthermore, suppose that there exists a control $\mathbf{u}^* \in \mathcal{U}$ *and a state* $\mathbf{x}^* \in \mathcal{X}$ *fulfilling the state equation with initial condition given by* (2.1)*, for which the Hamiltonian minimization condition* (2.14) *holds. Then* $\mathbf{u}^*$ *is a globally optimal control.*

Proof. Using the HJB equation (2.12) for $x = \mathbf{x}^*(t)$ in the first step and the Hamiltonian minimization condition (2.14) in the second step, we obtain

$$\begin{aligned} -V_t(t, \mathbf{x}^*(t)) &= \inf_{u \in U} H(t, u, \mathbf{x}^*(t), V_x(t, \mathbf{x}^*(t))) \\ &= H(t, \mathbf{u}^*(t), \mathbf{x}^*(t), V_x(t, \mathbf{x}^*(t))). \end{aligned}$$

Next, we plug in Definition 2.3 of the Hamiltonian as well as the state equation (2.1), and move all terms to the right-hand side of the equation, that is

$$\begin{aligned} 0 &= \psi(t, \mathbf{u}^*(t), \mathbf{x}^*(t)) + V_t(t, \mathbf{x}^*(t)) + V_x(t, \mathbf{x}^*(t)) \cdot \dot{\mathbf{x}^*}(t) \\ &= \psi(t, \mathbf{u}^*(t), \mathbf{x}^*(t)) + \frac{\mathrm{d}}{\mathrm{d}t} V(t, \mathbf{x}^*(t)). \end{aligned}$$

Hence, the total time derivative of V is attained which we integrate over the entire time interval $[t_0, t_f]$ yielding

$$0 = \int_{t_0}^{t_f} \psi(t, \mathbf{u}^*(t), \mathbf{x}^*(t)) \, \mathrm{d}t + V(t_f, \mathbf{x}^*(t_f)) - V(t_0, \mathbf{x}^*(t_0)).$$

Finally, we apply the initial condition of the state equation contained in (2.1) and the final condition of the HJB equation (2.13) resulting in

$$V(t_0, x_0) = \int_{t_0}^{t_f} \psi(t, \mathbf{u}^*(t), \mathbf{x}^*(t)) \, \mathrm{d}t + \phi(\mathbf{x}^*(t_f)) = J(\mathbf{u}^*, \mathbf{x}^*).$$

Now, we consider the HJB equation for an arbitrary control $\mathbf{u} \in \mathcal{U}$ with corresponding state trajectory $\mathbf{x} \in \mathcal{X}$ starting at x_0. This leads to

$$\begin{aligned} -V_t(t, \mathbf{x}(t)) &= \inf_{u \in U} H(t, u, \mathbf{x}(t), V_x(t, \mathbf{x}(t))) \\ &\leq H(t, \mathbf{u}(t), \mathbf{x}(t), V_x(t, \mathbf{x}(t))), \end{aligned}$$

and analogously to the steps above, we get

$$V(t_0, x_0) \leq \int_{t_0}^{t_f} \psi(t, \mathbf{u}(t), \mathbf{x}(t)) \, \mathrm{d}t + \phi(\mathbf{x}(t_f)) = J(\mathbf{u}, \mathbf{x}).$$

Thus, we obtain

$$J(\mathbf{u}^*, \mathbf{x}^*) \leq J(\mathbf{u}, \mathbf{x}) \quad \text{for all } (\mathbf{u}, \mathbf{x}) \in F_{ad},$$

where F_{ad} is the feasible set (2.5). This concludes the proof. □

Remark 2.7. From the proof, we can also deduce that $V(t_0, x_0) = J(\mathbf{u}^*, \mathbf{x}^*)$ is the minimum cost of the OCP (2.4). However, $\mathbf{u}^*$ is not necessarily unique since there might be other optimal controls yielding the same cost. Most importantly, this theorem can be used to verify optimality of candidate controls.

The HJB theory developed above has been based on the restrictive assumption that $V \in \mathcal{C}^1$. However, the common scenario is that the value function has singularities and therefore fails to be differentiable, as shown by the example constructed

in [44, Sec. 5.2.1]. In fact, this theory can be extended using a generalized concept of a solution to PDEs, namely viscosity solutions as defined in, e. g., [55, Def. 5.1.4; 3, Sec. 1.3; 44, Sec. 5.3.2]. Then, the necessary and sufficient conditions for optimality can be reformulated under some technical assumptions, such as f, ψ, and ϕ being uniformly continuous, f_x, ψ_x and ϕ_x being bounded, and $U \subseteq \mathbb{R}^m$ being compact. The following theorem cited from [44, Sec. 5.3.3] states the necessary conditions extended to this more general setting.

Theorem 2.8. *Let V be the value function from Definition 2.5 associated to the OCP* (2.4). *Then V is a unique viscosity solution of the HJB equation* (2.12) *with its final condition* (2.13). *It is also locally Lipschitz.*

We note that at all points where V is differentiable, the PDE must hold in the classical sense as we would expect from our previous derivation. The proof, as well as the generalized sufficient condition from Theorem 2.6 can be found in, for instance, [44, Sec. 5.3.3].

2.4 Relation between PMP and HJB Approach

We have seen that finding an optimal control can be based on the minimum principle as well as on the Hamilton-Jacobi-Bellman theory. Thus, it is naturally to ask to what extent both approaches are interconnected, especially because the two Hamiltonian minimization conditions (2.10) and (2.14) strongly resemble each other. Indeed, we can establish some relations between both optimal control theories which we will exploit for the numerical closed-loop solution presented in Chapter 3.

2.4.1 Costate as Gradient of the Value Function

We start by deducing the PMP from the HJB framework. More precisely, solving the HJB equation leads, in principle, to a solution of the adjoint equation, i. e., the costate can be expressed in terms of the value function. Comparing the Hamiltonian minimization conditions, one can guess that

$$\mathbf{p}^*(t) = V_x(t, \mathbf{x}^*(t)). \tag{2.15}$$

Under the assumption that $V \in \mathcal{C}^2$, it can be verified that the suggested term for the costate indeed satisfies the adjoint equation, see [44, Sec. 5.2]. Besides that, the boundary condition (2.13) of the value function matches the one for the costate contained in (2.7). Although computing the costate vector as the gradient of the value function is, in practice, unreasonable since solving the adjoint ODE is generally more tractable than solving the HJB PDE, the formula (2.15) still provides an interesting insight: The costate, originally introduced as a Lagrange multiplier, can also be interpreted as the sensitivity of the optimal cost-to-go with respect to the state x. Moreover, the link between costate and value function provides an important tool that bridges the gap between the necessary and sufficient optimality conditions. This means that the candidate open-loop solutions resulting from the PMP can be verified to be optimal by the sufficiency conditions of the HJB theory. Detailed claims and proofs under which additional assumptions the necessary conditions of the PMP become also sufficient are given in [3, Sec. 3.3.4; 55, Sec. 5.2.2].

2.4.2 Hamiltonian System as Characteristics of the HJB Equation

Now, we will consider the opposite direction, that is how the PMP leads to a solution of the HJB equation. This can be achieved via the method of characteristics which reduces a PDE to a suitable system of ODEs along which the solution can be integrated from some initial curve. The solution curves of these ODEs are called characteristics and with their help a solution for the original PDE can be found, see [44, Sec. 7.2.1; 55, Ch. 5].

For applying this technique to the HJB equation (2.12), we assume that the control minimizing the Hamiltonian has been already plugged into the equation. Therefore, the infimum vanishes and we will omit the control variable u in the Hamiltonian function H for simplicity. Besides that, we introduce an additional state dimension $x_{n+1} \coloneqq t$, so that we obtain an augmented state

$$\tilde{x} \coloneqq (x, t)^\top = (x_1, \dots, x_{n+1})^\top \in \mathbb{R}^{n+1}.$$

Under a slight abuse of notation w. r. t. the arguments of the functions V and H, setting $V(\tilde{x}) = V(t, x)$ and $H(\tilde{x}, p) = H(t, x, u, p)$, we can then rewrite the HJB equation as

$$F(\tilde{x}, V(\tilde{x}), V_{\tilde{x}}(\tilde{x})) \coloneqq V_{x_{n+1}}(\tilde{x}) + H(\tilde{x}, V_x(\tilde{x})) = 0 \quad \text{for all } \tilde{x} \in \mathbb{R}^{n+1} \tag{2.16}$$

where we defined $F : \mathbb{R}^{n+1} \times \mathbb{R} \times \mathbb{R}^{n+1} \to \mathbb{R}$.

We will not derive the characteristic ODEs for the general formulation of a first-order PDE, but refer to [21, Sec. 2.1] instead. Before examining how the characteristic equations look like in the specific case of the HJB PDE, we define

$$\mathbf{p}_i \coloneqq V_{x_i} \quad \text{for all } i = 1, \dots, n+1,$$

which coincides for the first n components denoted by $\mathbf{p} \coloneqq (\mathbf{p}_1, \dots, \mathbf{p}_n)$ with the relation (2.15) between costate and value function found above. For the last component, we can plug in the simplified HJB equation (2.16) yielding

$$\mathbf{p}_{n+1} = V_{x_{n+1}} = -H \tag{2.17}$$

To further simplify notation, we omit the arguments of the functions involved in the following characteristic ODEs. According to [44, Sec. 7.2.1], these are then given by

$$\begin{aligned} \dot{\mathbf{x}}_i &= F_{p_i} && \text{for all } i = 1, \dots, n+1, \\ \dot{\mathbf{p}}_i &= -F_{x_i} - \mathbf{p}_i F_V && \text{for all } i = 1, \dots, n+1 \quad \text{and} \\ \dot{V} &= \textstyle\sum_{i=1}^{n+1} \mathbf{p}_i F_{p_i}. \end{aligned} \tag{2.18}$$

The first set of characteristic ODEs leads to $\dot{\mathbf{x}}_i = H_{p_i}$ for all $i = 1, \dots, n$ yielding the state equation (2.8), as well as to $\dot{\mathbf{x}}_{n+1} = 1$, or equivalently $\dot{t} = 1$ which clearly holds true. The second set simplifies to $\dot{\mathbf{p}}_i = -H_{x_i}$ for all $i = 1, \dots, n+1$ since F does actually not depend on V. The first n equations assemble the adjoint equation (2.7) whereas the last one gives $\dot{\mathbf{p}}_{n+1} = -H_t$. Due to (2.17), this is merely the simplified HJB equation (2.16) differentiated w. r. t. the time.

So far, we have seen that the state and adjoint equation summarized in the Hamiltonian system (2.9) arise as characteristic ODEs. Moreover, there is another

ODE

$$\dot{V} = \sum_{i=1}^{n} \mathbf{p}_i H_{p_i} + \mathbf{p}_{n+1} = \mathbf{p} \cdot H_p - H = -\psi. \tag{2.19}$$

which we computed from the last characteristic equation in (2.18) using (2.17) and Definition 2.3 of the Hamiltonian function simplified to $H(\tilde{x}, p) = \psi(\tilde{x}) + p \cdot f(\tilde{x})$. It describes the evolution of the value function V and is key for transforming the characteristic curves obtained from the state and adjoint equation into a solution to the HJB equation.

Last but no least, we need to specify a curve through which the solution surface passes in order to obtain suitable boundary conditions for the ODEs. For the HJB equation, this curve is given by the condition (2.13). The resulting Cauchy problem can now be solved in the following way, see [44, Sec. 7.2.2]: In order to determine the value $V(t, x)$, we have to find a state $x_f \in \mathbb{R}^n$ and the characteristics $\mathbf{x} \in \mathcal{X}$ and $\mathbf{p} \in \mathcal{X}$ solving the Hamiltonian system (2.9) with $\mathbf{u} \in \mathcal{U}$ satisfying the Hamiltonian minimization condition (2.10), such that $\mathbf{x}(t) = x$ as well as $\mathbf{x}(t_f) = x_f$ and $\mathbf{p}(t_f) = \phi_x(x_f)$. Then, equation (2.19) enables us to calculate the value function along these characteristics. The fundamental theorem of calculus yields

$$V(t_f, \mathbf{x}(t_f)) - V(t, \mathbf{x}(t)) = \int_t^{t_f} \dot{V}(s, \mathbf{u}(s), \mathbf{x}(s)) \, \mathrm{d}s = -\int_t^{t_f} \psi(s, \mathbf{x}(s)) \, \mathrm{d}s,$$

and plugging in the boundary conditions as well as rearranging the terms result in

$$V(t, x) = \phi(x_f) + \int_t^{t_f} \psi(s, \mathbf{x}(s)) \, \mathrm{d}s.$$

For the purpose of this work, we will proceed slightly differently: For a given initial condition x_0, we can already determine a candidate optimal open-loop control $\mathbf{u}^*(t; x_0)$ with corresponding optimal state $\mathbf{x}^*$ and associated costate $\mathbf{p}^*$ by using the necessary conditions of Theorem 2.4. The trajectories $\mathbf{x}^*$ and $\mathbf{p}^*$ serve us as characteristics curves, along which (2.19) is integrated. Setting $\mathbf{v}(t) \coloneqq V(t, \mathbf{x}^*(t))$, we arrive at the ODE

$$\dot{\mathbf{v}}(t) = -\psi(t, \mathbf{u}^*(t; x_0), \mathbf{x}^*(t)) \quad \text{for a. a. } t \in [t_0, t_f],$$

with boundary condition

$$\mathbf{v}(t_f) = \phi(\mathbf{x}(t_f))$$

deduced from (2.13).

Adding the dynamics of $\mathbf{v}$ to the system (2.9) provided by the PMP, we obtain an extended BVP for each initial condition, cf. [36, Sec. 2; 49, Sec. 2.1], taking the form

$$\begin{cases} \dot{\mathbf{x}}(t) = H_p(t, \mathbf{u}^*(t; x_0), \mathbf{x}(t), \mathbf{p}(t)) & \text{for a. a. } t \in [t_0, t_f]; \quad \mathbf{x}(t_0) = x_0, \\ \dot{\mathbf{p}}(t) = -H_x(t, \mathbf{u}^*(t; x_0), \mathbf{x}(t), \mathbf{p}(t)) & \text{for a. a. } t \in [t_0, t_f]; \quad \mathbf{p}(t_f) = \phi_x(\mathbf{x}(t_f)), \\ \dot{\mathbf{v}}(t) = -\psi(t, \mathbf{u}^*(t; x_0), \mathbf{x}(t)) & \text{for a. a. } t \in [t_0, t_f]; \quad \mathbf{v}(t_f) = \phi(\mathbf{x}(t_f)). \end{cases} \tag{2.20}$$

We will solve this system by a causality-free BVP solver explained in Section 3.3.1 in order to adaptively generate data required for training and validating the neural network model.

Chapter 3

Adaptive Deep Learning

3.1 DL Approach for Solving HJB Equations

As initially discussed in Section 1.1, numerically finding closed-loop solutions to Problem 2.2 for high-dimensional non-linear systems is challenging due to the curse of dimensionality. Since traditional techniques only become feasible when imposing restrictive assumptions on the system dynamics and cost, we take a model-based and data-driven approach coming from the fast evolving field of deep learning (DL). More precisely, a feedback control law is designed based on the prediction of a neural network (NN) which models the value function over a semi-global domain and has been trained through supervised learning. The data is generated adaptively during the training phase by using a numerical method providing open-loop optimal controls. In the scope of this work, we focus on the characteristic method suggested in [35, Sec. 2] which requires solving the two-point boundary value problem (BVP) derived earlier in Section 2.4.2. Combining the training of the NN model with the data generation through the BVP solver via sophisticated heuristics, we develop an algorithm for approximating semi-global solutions to Hamilton–Jacobi–Bellman (HJB) equations. Hence, this adaptive DL approach allows us to overcome the curse of dimensionality, and thereby to compute candidate optimal feedback controllers for general optimal control problems in real-time.

In this chapter, we provide a detailed description of this adaptive deep learning approach including a short introduction to DL concepts and how they are used in the context of solving HJB equations, as well as a sophisticated guideline for progressive and adaptive data generation. Continuing the development of the algorithm presented in [49, Alg. 4.1], we suggest and discuss key ideas contributing to a more efficient computation and accurate approximation of optimal feedback controls.

3.2 Neural Network Model

At the core of our approach to solve the HJB equation (2.12) lies an artificial neural network. In this section, we examine various strategies to approximate the value function and to design a suitable network architecture, as well as promising techniques to train the model efficiently by supervised learning and to validate its accuracy.

3.2.1 Value Function Approximation for Feedback Control

A straightforward way of approximating the value function of Definition 2.5 over a semi-global domain $\mathbb{X} \subset \mathbb{R}^n$ is to directly use a neural network as proposed in [49, Sec. 3.1]. This means that we aim to train a neural network model represented by the

function F^{NN} such that

$$F^{\mathrm{NN}}(t,x) \approx V(t,x) \quad \text{for all } t \in [t_0, t_f] \text{ and } x \in \mathbb{X}.$$

By harnessing the network's inherent capacity for dealing with high-dimensional data, we are able to also handle problems with high-dimensional state spaces.

We take this idea yet a step further in order to ensure that the final condition (2.13) is always satisfied. Hence, we seek a model such that

$$F^{\mathrm{NN}}(t,x) - F^{\mathrm{NN}}(t_f,x) + \phi(x) \approx V(t,x) \quad \text{for all } t \in [t_0, t_f] \text{ and } x \in \mathbb{X}. \tag{3.1}$$

In both cases, the value function approximator on the left-hand side will be abbreviated by V^{NN} to simplify notation. Then, the NN feedback controller is determined via the Hamiltonian minimization condition (2.14), that is

$$\mathbf{u}^{\mathrm{NN}}(t,x) \in \operatorname*{argmin}_{u \in \mathbb{R}^m} H(t,u,x,V_x^{\mathrm{NN}}(t,x)) \quad \text{for all } t \in [t_0, t_f] \text{ and } x \in \mathbb{X}. \tag{3.2}$$

Here, V_x^{NN} is the gradient of the NN model with respect to the state variable, which can be computed by automatic differentiation (AD). This technique is based on the chain rule and is commonly used to differentiate neural networks w. r. t. their model parameters for gradient-based optimization, see, e. g., [27, Sec. 6.5.9], but it works equally for evaluating the derivative of a neural network w. r. t. its input.

3.2.2 Residual Network Architecture

Generally, any particular neural network architecture and training algorithm for supervised learning could be used within the adaptive DL framework. In fact, one might even choose them in accordance with the properties of the specific problem and the desired solution accuracy. A simple and common choice are deep feedforward networks, also known as multilayer perceptrons (MLPs). For a review of this basic NN structure we refer to [27, Ch. 6; 23, Ch. 10; 30, Sec. 3].

However, in this work, we will focus on residual neural networks (ResNets) developed in [29] because they have shown an improved performance, particularly with regard to NNs of high depth as tested experimentally in [24, Sec. 4.2]. Since ResNets allow information to jump over layers via skip connections as illustrated in Figure 3.1, information loss along the data-processing flow is prevented. This helps to avoid vanishing gradients and representational bottlenecks, enhancing both training and expressivity of the network, see [14, Sec. 7.1.4; 23, Ch. 13]. We also note that ResNets can be interpreted as explicit forward Euler discretization of a so-called neural ordinary differential equation (NODE) as investigated in [11] and also [17, Sec. 2; 24, Sec. 2]. Thus, leveraging knowledge about continuous dynamical systems opens up further possibilities for theoretical analysis of the network's properties carried out, for instance, in [59], as well as in [28], particularly with respect to stability of the forward propagation.

In the following, we specify the structure of the ResNet along with its notation which fits our purposes best. Let F^{ResNet} be a residual network with an input layer of $d_0 = n + 1$ neurons, $L \in \mathbb{N}$ hidden layers of width $d_l \in \mathbb{N}, l = 1, \dots, L$, and an one-dimensional output layer, i. e., $d_{L+1} = 1$. We note that the dimension is deliberately kept constant across all hidden layers, that is $d_1 = \cdots = d_L =: d$, in order to allow for skip connections characterizing the residual architecture. Following the suggestion in [17] of augmenting the input space, we always choose $d > n + 1$. Then, the model

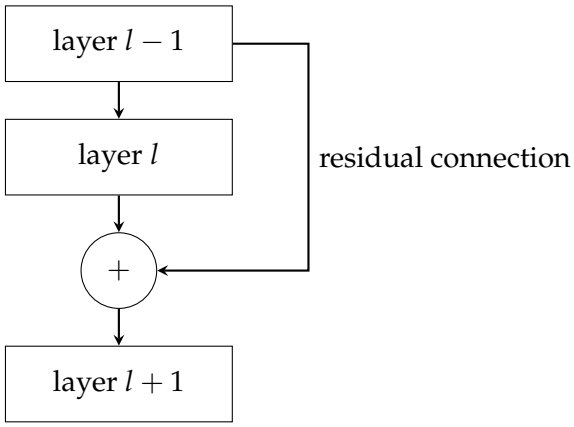

FIGURE 3.1: Residual block with skip connection constituting the architecture of a ResNet. Adapted from [14, Fig. 7.5; 23, Fig. 13-12; 29, Fig. 2].

parameters containing the weights and biases of all layers are given by

$$\theta = \{W^{[l]}, b^{[l]}\}_{l=0}^{L} \in \bigtimes_{l=0}^{L} \left(\mathbb{R}^{d_{l+1} \times d_l} \times \mathbb{R}^{d_{l+1}}\right). \tag{3.3}$$

In each layer, we first apply an affine map determined by the weight matrix $W^{[l]}$ and the bias vector $b^{[l]}$ to the input. Afterwards, a non-linear activation function $\sigma : \mathbb{R} \to \mathbb{R}$ is applied componentwise. Here, we use the softplus function

$$\sigma(x) = \log(1 + \exp(x))$$

which is a smooth approximation to the popular rectified linear unit (ReLU) function. Finally, the output of the previous hidden layer is passed over via a residual connection to be summed with the current activation as shown in Figure 3.1. In summary, the network function is a composition of all layers

$$F^{\text{ResNet}} = f^{[L]} \circ \cdots \circ f^{[0]} \tag{3.4}$$

where the layer inputs $y \in \mathbb{R}^{d_l}$ are transformed as follows:

$$f^{[l]}(y) = \begin{cases} \sigma\left(W^{[l]}y + b^{[l]}\right), & l = 0 \\ y + \sigma\left(W^{[l]}y + b^{[l]}\right), & l = 1, \ldots, L-1 \\ W^{[l]}y + b^{[l]}, & l = L. \end{cases}$$

In contrast, a conventional feedforward network $F^{\text{MLP}} = g^{[L]} \circ \cdots \circ g^{[0]}$ simply passes the data through the layers formed by the weights, biases and activation, without adding any skip connections yielding

$$g^{[l]}(y) = \begin{cases} \sigma\left(W^{[l]}y + b^{[l]}\right), & l = 0, \ldots, L-1 \\ W^{[l]}y + b^{[l]}, & l = L. \end{cases}$$

3.2.3 Physics-Informed Training with Adam

After initializing some neural network F^{NN} for a value function approximator V^{NN}, the model has to be trained by optimizing over its parameters θ. For supervised learning, we require labelled data of the form

$$D_{\text{train}} = \{(t^{(k)}, x^{(k)}), V^{(k)}\}_{k=1}^{K}.$$

Each of the K data points consists of a specific time $t^{(k)} \in [t_0, t_f]$ and state value $x^{(k)} \in \mathbb{X}$ as input, as well as the value function output $V^{(k)} = V(t^{(k)}, x^{(k)})$ as label. Such data sets are generated by a BVP solver used for solving (2.20) for multiple initial conditions as explained later in Section 3.3.

Using the mean squared error (MSE) loss to measure the error on a single training sample $\{(t^{(k)}, x^{(k)}), V^{(k)}\}$, the overall cost is the arithmetic mean of the losses across the whole training set D_{train}, i. e.,

$$\mathrm{MSE}_V(\theta; D_{\text{train}}) \coloneqq \frac{1}{K} \sum_{k=1}^{K} \left| V^{\mathrm{NN}}(t^{(k)}, x^{(k)}; \theta) - V^{(k)} \right|^2.$$

Typically, some penalty term, such as an L^2 (weight decay or ridge regression) or L^1 (lasso regression) regularizer, is added to this cost in order to prevent overfitting, see [27, Sec. 7.1; 14, Sec. 4.4.2]. These techniques lead to smaller weight matrices, and therefore reduce the complexity of the neural network, which makes it easier to generalize on unseen data. But an even more effective regularization can be achieved by incorporating information about the known solution structure into the training. Respecting the underlying physical laws of the problem, we can construct more data-efficient and physically consistent models also known as physics-informed neural networks, see [53].

In our context, we can leverage the relation (2.15) between costate and gradient of the value function as proposed in [49, Sec. 3.2]. To enable this, we augment the training set by adding the costate $p^{(k)} = \mathbf{p}(t^{(k)})$ to the label of each data point yielding

$$D_{\text{train}} = \{(t^{(k)}, x^{(k)}), (V^{(k)}, p^{(k)})\}_{k=1}^{K}. \tag{3.5}$$

Since the costate data is collected naturally as part of solving the BVP (2.20), this does not require any additional computational effort. We can then define the gradient cost by again using the MSE loss

$$\mathrm{MSE}_p(\theta; D_{\text{train}}) \coloneqq \frac{1}{K} \sum_{k=1}^{K} \left\| V_x^{\mathrm{NN}}(t^{(k)}, x^{(k)}; \theta) - p^{(k)} \right\|_2^2, \tag{3.6}$$

which is minimized along with the value function cost MSE_V. Hence, the physics-informed deep learning problem is given as

$$\min_{\theta} \mathrm{Cost}(\theta; D_{\text{train}}) \coloneqq \mathrm{MSE}_V(\theta; D_{\text{train}}) + \lambda \cdot \mathrm{MSE}_p(\theta; D_{\text{train}}), \tag{3.7}$$

where $\lambda > 0$ is the regularization parameter.

This gradient regularization technique yields several benefits: First and foremost, it promotes a more accurate approximation of the gradient of the value function. This is crucial because the NN controller (3.2) depends explicitly on the gradient approximation V_x^{NN}. Secondly, it makes more efficient use of small data sets by

including the already available costate data, which is particularly helpful when data generation is expensive.

During the training phase, the model parameters will be updated according to some optimization method in order to gradually improve the accuracy of the value function approximator V^{NN}. In deep learning, stochastic gradient descent (SGD) in combination with backpropagation has become the most popular algorithm for training neural networks, see [27, Sec. 8.3.1; 14, Sec. 2.4.3; 30, Sec. 4]. This algorithm typically splits the data set into mini-batches and iterates through them epoch by epoch. As opposed to deterministic full-batch optimization methods, the cost gradient for each descent is approximated only on a small fraction of the training data. Following the argumentation in [37], this approach leads to better generalization because the inherent noise in the gradient estimation allows the optimizer to explore more areas of the cost landscape. As a result, this tends to converge to so-called flat minimizers. In contrast, methods that use the entire training data for gradient computation often get trapped in a sharp minimum which corresponds to a narrow and steep ravine of the cost function. Hence, the resulting NN model is more prone to overfitting compared to the same network architecture trained with mini-batches. This degradation of the model's quality with respect to its ability to generalize to unseen data is known as generalization gap.

However, we suggest to use another stochastic first-order optimization method called Adam which was presented in [39]. This optimizer is an extension to the classical SGD. Since it also operates on mini-batches, Adam does not suffer from the generalization gap. Moreover, it computes individual adaptive learning rates for different parameters using estimations of first and second moments of the gradients. Therefore, Adam promises more efficient training while requiring less hyperparameter tuning.

3.2.4 Empirical Validation and Testing

Furthermore, the data-based learning approach described in the previous section allows us to empirically validate the model accuracy. This is a significant advantage because we can assess the performance of our algorithm not only for problems where semi-analytical closed-loop solutions are available, but also for more challenging, general high-dimensional systems.

We measure the accuracy of the NN solution on the basis of two error metrics, inspired by [49, Sec. 3.3]: The relative mean absolute error (RMAE) w. r. t. the value function prediction given by

$$\mathrm{RMAE}_V(\theta; D) \coloneqq \frac{\sum_{k=1}^{K} \left| V^{\mathrm{NN}}(t^{(k)}, x^{(k)}; \theta) - V^{(k)} \right|}{\sum_{k=1}^{K} \left| V^{(k)} \right|},$$

and the relative mean L^1 (RML^1) error w. r. t. the prediction of the gradient, that is

$$\mathrm{RM}L^1{}_p(\theta; D) \coloneqq \frac{\sum_{k=1}^{K} \left\| V_x^{\mathrm{NN}}(t^{(k)}, x^{(k)}; \theta) - p^{(k)} \right\|_1}{\sum_{k=1}^{K} \left\| p^{(k)} \right\|_1}.$$

Since our main interest lies in an accurate approximation of the optimal control, we additionally benchmark the resulting NN controller against the control data $u^{(k)} =$

$\mathbf{u}(t^{(k)})$ from the open-loop solution via

$$\mathrm{RMAE}_u(\theta; D) := \frac{\sum_{k=1}^{K} \left| u^{\mathrm{NN}}(t^{(k)}, x^{(k)}; \theta) - u^{(k)} \right|}{\sum_{k=1}^{K} \left| u^{(k)} \right|}.$$

All of these error estimates are based on a data set D of the form (3.5). Apart from the training set D_{train}, we also generate a validation set D_{val}, as well as a test set D_{test} using the same BVP solver from Section 3.3.1. Both sets contain independent data from different trajectories starting at randomly drawn initial conditions.

Applying the above error metrics to those three data sets, we obtain the training, validation and test errors, respectively, see [52, Sec. 2.1; 14, Sec. 4.2.1]. The first one allows us to measure the training progress and the second to monitor the generalization loss during the training phase. Therefore, training and validation error are especially relevant for determining an early stopping criterion, which provides valuable information for when new data should be added to the training set as examined in Section 3.3.5. The latter error is computed on the test set, also known as held-out validation set, because it remains unknown to the optimization algorithm. The test error estimates the generalization error after training is completed and, thus, enables us to evaluate the quality of the final NN model.

3.3 Progressive and Adaptive Data Generation

The availability of data, particularly big data, is central to any machine learning algorithm. In our optimal control framework, data sets are not given beforehand, but need to be generated by some numerical method. A comprehensive overview of suitable data development algorithms is given in [35] including the method of characteristics applied to the HJB equation which has been used in [49, Sec. 2.1; 36, Sec. 3.1] and is taken as a basis for our considerations as well. As derived in Section 2.4.2 this approach leads to the boundary value problem (2.20) consisting of the state and adjoint equation, augmented by a third ODE describing the evolution of the value function.

3.3.1 Causality-Free BVP Solver

For solving this two-point BVP, we use the SciPy implementation based on a three-point Lobatto method, see [38]. This algorithm solves a collocation formula by a damped Newton method with an affine-invariant criterion function providing a solution that is fourth-order accurate.

An essential feature of this BVP solver is that the computation is not based on spatial causality. This means that the solution for each initial condition x_0 is computed individually without using an approximation at other nearby points in the state space. Hence, the algorithm does not require a grid on some initial condition domain $\mathbb{X}_0 \subset \mathbb{R}^n$ but can be applied to single independently chosen initial conditions. This property distinguishes our BVP solver from many other numerical methods for generating data, such as the backpropagation technique described in [35, Sec. 2.3] and used in [32]. Most importantly, the causality-free approach provides several major advantages summarized in [35, Sec. 1], namely:

- Being entirely grid-free, the solver can also be applied to BVPs arising from optimal control problems with high-dimensional state spaces.

- The training data can be generated adaptively, i. e., in selected regions where the value function tends to be more difficult to learn.
- The accuracy of the trained NN can be validated empirically by means of independently generated validation and test data.
- Data generation is perfectly parallelizable, so that the power of parallel computers can be fully exploited.

Although spatially causality-free methods are known for being comparatively slow, we are less affected by this drawback since solving BVPs for generating data sets can be done offline along with training the neural network as described in Section 3.2.3. Hence, the online computation merely comprises in evaluating the gradient of the trained NN model and plugging it into the feedback law (3.2) yielding real-time controllers.

3.3.2 Initialization by Time-Marching and NN Warm Start

Besides this favorable causality-free property, we face the difficulty that the BVP solver is highly sensitive to the initial guesses for the optimal state $\mathbf{x}$, as well as the costate $\mathbf{p}$ and value function $\mathbf{v}$ along the respective trajectory. In particular, its convergence depends heavily on an accurate initialization of these three variables at several points in time. If we simply initialize the state values with x_0, and the costate and value function with zero at all times, the algorithm often fails, especially for more complicated problems. To confront this challenge, we employ two methods of constructing better guesses, namely time-marching and neural network warm start.

At first, we consider the time-marching approach from [35, Sec. 2.1; 49, Sec. 2.2] which is based on the continuation of solutions calculated on time intervals of increasing length. Choosing a time sequence $t_0 < t_1 < \cdots < t_J = t_f$, we start by solving the BVP on $[t_0, t_1]$. For this short interval, the BVP solver mostly converges when initializing all variables as suggested above. Now, the solution is extended iteratively to longer time intervals as follows: Given a solution $\mathbf{x}^j, \mathbf{p}^j$ and $\mathbf{v}^j$ on the interval $[t_0, t_j], j = 1, \dots, J-1$, we expect the value at t_{j+1} to be close to the one at t_j, and therefore

$$(\mathbf{x}^{j+1}, \mathbf{p}^{j+1}, \mathbf{v}^{j+1})(t) \approx \begin{cases} (\mathbf{x}^j, \mathbf{p}^j, \mathbf{v}^j)(t), & t \in [t_0, t_j] \\ (\mathbf{x}^j, \mathbf{p}^j, \mathbf{v}^j)(t_j), & t \in (t_j, t_{j+1}]. \end{cases}$$

serves us as an initial guess for solving the BVP on $[t_0, t_{j+1}]$. When reaching $t_J = t_f$, we obtain the solution for the whole interval. However, we have to ensure that convergence is successfully achieved on all intermediate time intervals by choosing $t_{j+1} - t_j$ small enough. At the same time, increasing the number of time steps J leads to a significantly slower computation. Thus, we have to appropriately tune the sequence $\{t_j\}_{j=0}^J$ trading off convergence against speed. Another proposal is to start with a relatively large convergence tolerance and, while progressively extending the time interval, decreasing it so that we reach the desired accuracy for the final solution.

If we already have a partially trained neural network at hand, we can turn to a more computationally efficient method which does not require any parameter tuning. With the aid of the NN model, we can simulate the system dynamics yielding an approximation of the optimal state trajectory $\mathbf{x}^{\mathrm{NN}}$ along which the prediction of

the value function and its gradient is provided. Exploiting the relation (2.15) once again, we can immediately obtain a good initialization for all $t \in [t_0, t_f]$ by observing

$$\begin{aligned}\mathbf{x}(t) &\approx \mathbf{x}^{\mathrm{NN}}(t),\\ \mathbf{p}(t) &\approx V_x^{\mathrm{NN}}(t, \mathbf{x}^{\mathrm{NN}}(t)) \text{ and}\\ \mathbf{v}(t) &\approx V^{\mathrm{NN}}(t, \mathbf{x}^{\mathrm{NN}}(t)).\end{aligned}$$

This strategy of informing the initialization by NN-in-the-loop simulations is also known as NN warm start introduced in [35, Sec. 2.2; 49, Sec. 4.2]. Since we can only leverage information encoded by the neural network, the quality of the generated initial guesses depends on what the model has learned so far. But even for rarely optimized NN parameters, the approximation is usually accurate enough to prevent divergence of the BVP solver for most initial conditions. As training progresses, initial guesses to warm start the BVP solver get better, thereby speeding up the computation and improving the convergence rate even more.

3.3.3 Specification of Data Set Generation

In order to build up data sets, we solve the BVP for multiple initial conditions lying in some domain $\mathbb{X}_0 \subset \mathbb{R}^n$ and sample data points along the respective trajectories. Hence, each initial condition $x_0 \in \mathbb{X}_0$ yields hundreds of data points on the time interval $[t_0, t_f]$. Due to the causality-free algorithm, these initial conditions can be chosen either uniformly at random in $\mathbb{X}_0$ or, as we explore later in Section 3.3.6, in specific domains. Sampling initial conditions and solving the respective BVPs will be repeated until we obtain the desired amount of data. How computationally expensive this data generation process is depends clearly on the data set size, as well as the problem difficulty, but also on the initialization for the BVP solver, as we have investigated in Section 3.3.2.

Initially, we generate only a small set of training data D^0_{train} and validation data D_{val} with randomly drawn initial conditions because solving many BVPs is particularly expensive when we have to rely on the time-marching technique. During training, additional data can be generated much faster by initializing via the NN warm start method instead. We determine heuristically when and how many new data points will be added by making use of two interconnected criteria, early stopping and sample size selection explained in Sections 3.3.4 and 3.3.5, respectively. Besides that, we apply another heuristic which is examined in Section 3.3.6 to select initial conditions adaptively in regions where the value function is steep and the required control effort is large. Combining the progressive data generation with the adaptive sampling approach, we obtain a nested sequence of training sets

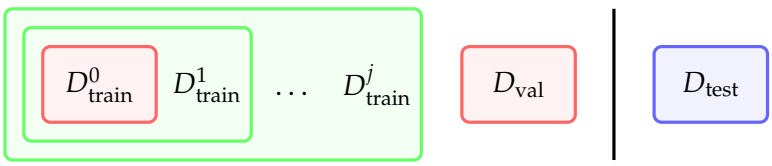

FIGURE 3.2: Overview of training, validation and test sets involved in the adaptive deep learning approach. Initial data generation (before training) is colored in red, adaptive data generation (during training) in green and final data generation (after training) in blue. Data on the left-hand side of the vertical line is known to the optimization algorithm whereas data on the right-hand side remains unseen.

$D^0_{\text{train}} \subset D^1_{\text{train}} \subset \cdots \subset D^j_{\text{train}}$. Finally, we create a test data set D_{test} after training is completed. This can be achieved in a very efficient way employing NN warm start once again because we expect the NN model to provide particularly well initial guesses. Depending on the application purpose, we can sample the initial conditions independently and identically distributed (i. i. d.) or choose them intentionally wherever we are interested in a high level of accuracy. Figure 3.2 summarizes the key features of all data sets for a better overview.

3.3.4 Variance Estimation based Sample Size Selection

As training progresses, we dynamically augment the initially generated small training set D^0_{train}. Thus, we face the decision of when and how much the data set size should be increased. Adding too little data hinders the network from accurately learning the value function without overfitting, whereas generating too much data becomes computationally expensive. Hence, we require some guideline making sure that we are taking reliable parameter updates while keeping the costs for data generation low.

As a starting point for establishing a suitable criterion, we consider the dynamic sample size technique developed in [9] for large-scale machine learning. In the original context, large amounts of data are available from the start but the size of the batches used in a stochastic optimization algorithm is gradually increased throughout the progression of the algorithm. In order to determine reasonable batch sizes, a heuristical test based on sample variance estimates of cost gradients is designed.

Although the setting differs from ours, we can still follow a similar approach when we replace the batch of the current iteration with the training set D^j_{train} used in the j-th training round. In a gradient-based optimization method, both are used for the evaluation of the cost function and its gradient, and are therefore closely related. Instead of increasing the batch size by taking larger samples from a pre-existing vast data set, we enlarge the training set size by generating new data. While the original setting allows us to select a completely different sample at each iteration, we have to reuse the same data points over and over again in our case. Besides that, we modify the cost function to allow for gradient regularization and use Adam as optimizer in order to take advantage of adaptive learning rates instead of performing line search. Furthermore, we need to be even more careful to prevent the sample size from growing too fast because generating large data sets is prohibitively expensive. Therefore, we slightly change the sample size selection scheme by using a different norm and introducing an upper bound as explained later in this section.

The resulting heuristic has already been successfully applied in [49]. We provide a derivation of the heuristical tests in full detail before refining them even more by linking them with an early stopping criterion.

Preliminaries

We begin with some definitions and notation needed for the derivation of the data set size selection scheme which we specifically adapt to our physics-informed training procedure described in Section 3.2.3. Hence, data sets are given in the form (3.5), but for simplicity, we only denote the input $(t, x) \in [t_0, t_f] \times \mathbb{X}$ of each data point and omit the label, i. e., the corresponding function values and costate data $(V(t, x), \mathbf{p}(t)) \in \mathbb{R} \times \mathbb{R}^n$.

First, we introduce the infinite data set

$$D_\infty := [t_0, t_f] \times \mathbb{X}$$

containing the continuous domain composed of the full time interval and the semi-global domain of interest. Furthermore, let $D^j_{\text{train}} \subset \mathcal{D}_\infty$ be the training set at some training round $j \in \mathbb{N}_0$ of size $K_j := |D^j_{\text{train}}| < \infty$. Thus, we write

$$D^j_{\text{train}} = \{(t^{(k)}, x^{(k)})\}_{k=1}^{K_j}.$$

In statistics, D_∞ is referred to as the population, whereas D^j_{train} is called a sample drawn from it. The following derivation is based on one simplifying assumption.

Assumption 3.1. All data points $(t^{(k)}, x^{(k)}) \in D^j_{\text{train}}$ are independent and identically distributed (i. i. d.).

We remark that this condition is indeed not fully satisfied in our setting. Although uniform sampling from $\mathbb{X}$ ensures that the initial conditions are i. i. d. in space, data points at times $t > t_0$ are sampled along the resulting optimal trajectories and are therefore spatially correlated. Moreover, the assumption gets further violated by applying the adaptive selection technique presented in Section 3.3.6. Presumably, this results in a smaller sample variance as compared to i. i. d. data. Despite neglecting this inaccuracy, we find that the heuristical tests derived under Assumption 3.1 still serve our purpose.

Next, we consider the objective function of the learning problem in (3.7). We note that setting $\lambda = 0$ yields a conventional MSE cost so that the derived conditions also apply to the case without gradient regularization. Since the data sets are now denoted without labels, they have to be implicitly represented in the cost function. Besides that, we merge the network's weights and biases in (3.3) into one parameter θ of dimension $R = \sum_{l=0}^{L} d_{l+1} \cdot (d_l + 1)$. Furthermore, the cost function naturally generalizes from finite to infinite data sets since the MSE of a predictor is generally defined as the expected value of the squared error loss.

Definition 3.2. The cost function for the learning problem $\min_\theta \text{Cost}(\theta; D)$ is defined as

$$\text{Cost}(\theta; D) := \mathbb{E}_{(t,x)\in D}\left[\left|V^{\text{NN}}(\cdot,\cdot;\theta) - V(\cdot,\cdot)\right|^2 + \lambda \left\|V_x^{\text{NN}}(\cdot,\cdot;\theta) - \mathbf{p}(\cdot)\right\|_2^2\right].$$

for all $\theta \in \mathbb{R}^R$ and $D \subset D_\infty$.

When applied to a finite data set such as D^j_{train}, the expectation simplifies to the arithmetic mean and we obtain the cost as defined before. As the size of the data set approaches infinity, the sums become integrals yielding

$$\text{Cost}(\theta; D_\infty) = \int_{[t_0,t_f]\times\mathbb{X}} \left[\left|V^{\text{NN}}(t,x;\theta) - V(t,x)\right|^2 + \lambda \left\|V_x^{\text{NN}}(t,x;\theta) - \mathbf{p}(x)\right\|_2^2\right] \mathrm{d}(t,x).$$

For first-order optimization, we require the gradient of the cost w. r. t. the model parameters θ denoted by $\nabla \text{Cost} := \left(\frac{\partial \text{Cost}}{\partial \theta_1}, \ldots, \frac{\partial \text{Cost}}{\partial \theta_R}\right)^\top$.

We also highlight a property of the cost function that we will exploit several times.

Proposition 3.3. *For all $\theta \in \mathbb{R}^R$ and $D \subset D_\infty$ holds*

$$\operatorname{Cost}(\theta; D) = \mathbb{E}_{(t,x)\in D}\left[\operatorname{Cost}(\theta; (t,x))\right],$$

and also

$$\nabla \operatorname{Cost}(\theta; D) = \mathbb{E}_{(t,x)\in D}\left[\nabla \operatorname{Cost}(\theta; (t,x))\right].$$

The first statement follows directly from Definition 3.2. Using the dominated convergence theorem, we can interchange expectation and differentiation and thus obtain the second statement for the gradient of the cost.

Derivation

We wish to find an NN parameter $\theta^* \in \mathbb{R}^R$ that minimizes the population cost function $\operatorname{Cost}(\cdot; D_\infty)$. Ideally, we choose an optimization method that leads to some θ such that we satisfy the first-order necessary condition for optimality closely enough, that is

$$\|\nabla \operatorname{Cost}(\theta; D_\infty)\|_2 < \epsilon$$

for some small $\epsilon > 0$. However, the cost gradient corresponding to the entire population is not known since we have only a small sample $D^j_{\text{train}} \subset D_\infty$ available. Hence, we can neither use the population gradient to compute the new iterate nor its norm as convergence criterion.

Employing an algorithm based on batch gradient descent, namely Adam, we use an approximation of $-\nabla \operatorname{Cost}(\theta; D^j_{\text{train}})$ as descent direction for each step. Thus, we need to verify that this direction also yields sufficient progress on the target objective $\operatorname{Cost}(\theta; D_\infty)$.

Because of Assumption 3.1 we know that the sample gradient $\nabla \operatorname{Cost}(\theta; D^j_{\text{train}})$ is an unbiased estimator of the population gradient $\nabla \operatorname{Cost}(\theta; D_\infty)$, i. e.,

$$\mathbb{E}_{D^j_{\text{train}}}\left[\nabla \operatorname{Cost}(\theta; D^j_{\text{train}})\right] = \nabla \operatorname{Cost}(\theta; D_\infty), \tag{3.8}$$

where we used the short-hand notation $\mathbb{E}_{D^j_{\text{train}}}[\cdot] := \mathbb{E}_{D^j_{\text{train}}\in\{D\subset D_\infty \,||D|=K_j\}}[\cdot]$. The quality of an estimator is commonly assessed by the mean squared error, which is the second moment of the error. In our case, we have

$$\operatorname{MSE}\left(\nabla \operatorname{Cost}(\theta; D^j_{\text{train}})\right) := \mathbb{E}_{D^j_{\text{train}}}\left[\left\|\nabla \operatorname{Cost}(\theta; D^j_{\text{train}}) - \nabla \operatorname{Cost}(\theta; D_\infty)\right\|_2^2\right]$$

and we aim to achieve

$$\sqrt{\operatorname{MSE}\left(\nabla \operatorname{Cost}(\theta; D^j_{\text{train}})\right)} \leq C \left\|\mathbb{E}_{D^j_{\text{train}}}\left[\nabla \operatorname{Cost}(\theta; D^j_{\text{train}})\right]\right\|_1 \tag{3.9}$$

for some small tolerance $C > 0$. Hence, we want to control the root mean squared error (RMSE) by the norm of the expected sample gradient as proposed in [9, Sec. 3; 49, Sec. 4.1].

In the following, we analyze (3.9) in order to derive a condition that evaluates the suitability of the current data set size and a rule for choosing the number of additional data points that need to be generated. We start with considering the MSE on the left-hand side of the inequality. Using (3.8) and then the linearity of the expectation along with the definition of the L^2 norm, we obtain

$$\begin{aligned}
\mathrm{MSE}\left(\nabla \mathrm{Cost}(\theta; D^j_{\text{train}})\right) &= \mathbb{E}_{D^j_{\text{train}}}\left[\left\|\nabla \mathrm{Cost}(\theta; D^j_{\text{train}}) - \mathbb{E}_{D^j_{\text{train}}}\left[\nabla \mathrm{Cost}(\theta; D^j_{\text{train}})\right]\right\|_2^2\right] \\
&= \sum_{r=1}^{R} \mathbb{E}_{D^j_{\text{train}}}\left[\left|\frac{\partial \mathrm{Cost}}{\partial \theta_r}(\theta; D^j_{\text{train}}) - \mathbb{E}_{D^j_{\text{train}}}\left[\frac{\partial \mathrm{Cost}}{\partial \theta_r}(\theta; D^j_{\text{train}})\right]\right|^2\right] \\
&= \sum_{r=1}^{R} \mathrm{Var}_{D^j_{\text{train}}}\left(\frac{\partial \mathrm{Cost}}{\partial \theta_r}(\theta; D^j_{\text{train}})\right) \\
&= \left\|\mathrm{Var}_{D^j_{\text{train}}}\left(\nabla \mathrm{Cost}(\theta; D^j_{\text{train}})\right)\right\|_1 .
\end{aligned}$$

Here, the short-hand notation $\mathrm{Var}_{D^j_{\text{train}}}(\cdot)$ is defined analogously to the previous one for the expectation. This result corresponds to the well-known bias-variance decomposition, but since the bias is zero, we are only left with the variance term.

Now, we use Proposition 3.3 for the gradient of the cost corresponding to the finite data set D^j_{train} so that the expected value becomes the arithmetic mean over all data points. Furthermore, we apply some standard calculation rules for the variance, namely multiplication with a constant and sum of uncorrelated variables. Indeed, the covariance of distinct data points vanishes since they are independent according to Assumption 3.1. This yields

$$\begin{aligned}
\mathrm{Var}_{D^j_{\text{train}}}\left(\nabla \mathrm{Cost}(\theta; D^j_{\text{train}})\right) &= \mathrm{Var}_{D^j_{\text{train}}}\left(\frac{1}{K_j}\sum_{k=1}^{K_j} \nabla \mathrm{Cost}(\theta; (t^{(k)}, x^{(k)}))\right) \\
&= \frac{1}{K_j^2}\sum_{k=1}^{K_j} \mathrm{Var}_{D^j_{\text{train}}}\left(\nabla \mathrm{Cost}(\theta; (t^{(k)}, x^{(k)}))\right) \\
&= \frac{1}{K_j}\,\mathrm{Var}_{(t,x)\in D_\infty}\left(\nabla \mathrm{Cost}(\theta; (t, x))\right).
\end{aligned}$$

In the last step, we leverage Assumption 3.1 once again, particularly that the data points are identically distributed.

Since the population variance cannot be computed, we approximate it with the sample variance, more precisely

$$\left\|\mathrm{Var}_{(t,x)\in D_\infty}\left(\nabla \mathrm{Cost}(\theta; (t, x))\right)\right\|_1 \approx \left\|\mathrm{Var}_{(t,x)\in D^j_{\text{train}}}\left(\nabla \mathrm{Cost}(\theta; (t, x))\right)\right\|_1 .$$

In practice, we may evaluate the sample variance based on an even smaller subset of D^j_{train} because the computational effort of calculating gradients corresponding to single data points mounts up very quickly. Putting everything together, we get the following approximation for the mean squared error:

$$\mathrm{MSE}\left(\nabla \mathrm{Cost}(\theta; D^j_{\text{train}})\right) \approx \frac{1}{K_j}\left\|\mathrm{Var}_{(t,x)\in D^j_{\text{train}}}\left(\nabla \mathrm{Cost}(\theta; (t, x))\right)\right\|_1 . \tag{3.10}$$

On the right-hand side of (3.9), we have another unknown term, namely the expected gradient. By (3.8) and Proposition 3.3, we deduce

$$\mathbb{E}_{D^j_{\text{train}}}\left[\nabla \text{Cost}(\theta; D^j_{\text{train}})\right] = \mathbb{E}_{(t,x)\in D_\infty}\left[\nabla \text{Cost}(\theta; (t,x))\right].$$

Similarly to our above approximation, we replace the population mean by the sample mean, i. e.,

$$\left\|\mathbb{E}_{(t,x)\in D_\infty}\left[\nabla \text{Cost}(\theta; (t,x))\right]\right\|_1 \approx \left\|\mathbb{E}_{(t,x)\in D^j_{\text{train}}}\left[\nabla \text{Cost}(\theta; (t,x))\right]\right\|_1,$$

and by using Proposition 3.3 once again, we finally obtain the sample gradient as approximation for the expected gradient:

$$\left\|\mathbb{E}_{D^j_{\text{train}}}\left[\nabla \text{Cost}(\theta; D^j_{\text{train}})\right]\right\|_1 \approx \left\|\nabla \text{Cost}(\theta; D^j_{\text{train}})\right\|_1. \tag{3.11}$$

Plugging both approximations, (3.10) and (3.11), into the original RMSE criterion (3.9), we obtain the condition

$$\frac{\sqrt{\left\|\text{Var}_{(t,x)\in D^j_{\text{train}}}\left(\nabla \text{Cost}(\theta; (t,x))\right)\right\|_1}}{\sqrt{K_j}\left\|\nabla \text{Cost}(\theta; D^j_{\text{train}})\right\|_1} \leq C. \tag{3.12}$$

which can be checked at any time during the training phase. If it is not met, we cannot expect that further training with the same amount of data yields a significant decrease of $\text{Cost}(\theta; D_\infty)$. Hence, we decide to continue training with a larger set D^{j+1}_{train} by generating more data.

The criterion 3.12 also guides us in how to select the new sample size $K_{j+1} = |D^{j+1}_{\text{train}}|$ by considering how much data would be necessary in order to satisfy it in the next training round. Based on the assumption that the increase in the data set size is small enough such that the sample variance and the sample gradient do not change much, i. e.,

$$\left\|\text{Var}_{(t,x)\in D^{j+1}_{\text{train}}}\left(\nabla \text{Cost}(\theta; (t,x))\right)\right\|_1 \approx \left\|\text{Var}_{(t,x)\in D^j_{\text{train}}}\left(\nabla \text{Cost}(\theta; (t,x))\right)\right\|_1 \text{ and}$$
$$\left\|\nabla \text{Cost}(\theta; D^{j+1}_{\text{train}})\right\|_1 \approx \left\|\nabla \text{Cost}(\theta; D^j_{\text{train}})\right\|_1,$$

the above criterion suggests

$$K_{j+1} \geq \frac{\left\|\text{Var}_{(t,x)\in D^j_{\text{train}}}\left(\nabla \text{Cost}(\theta; (t,x))\right)\right\|_1}{C^2\left\|\nabla \text{Cost}(\theta; D^j_{\text{train}})\right\|_1^2}.$$

We have to pay attention not to demand an unreasonable amount of new data points, so that the computational costs for data generation do not explode. For this reason, we adopt the suggestions made in [49, Sec. 4.1]. First of all, we have already used the L^1 instead of the L^2 norm for the expected gradient in (3.9) because it is less sensitive to outliers. Additionally, we introduce an upper bound

$$K_{j+1} \leq MK_j$$

with $M > 1$ and choose the data set size for D_{train}^{j+1} according to the rule

$$K_{j+1} := \min \left\{ \left\lceil \frac{\left\| \operatorname{Var}_{(t,x)\in D_{\text{train}}^{j}} \left(\nabla \operatorname{Cost}(\theta; (t,x)) \right) \right\|_1}{C^2 \left\| \nabla \operatorname{Cost}(\theta; D_{\text{train}}^{j}) \right\|_1^2} \right\rceil , \lfloor MK_j \rfloor \right\}. \tag{3.13}$$

Application

Starting out from the RMSE as a measure for how reliable the sample cost gradient estimates the population cost gradient, we found that the sample variance in the gradient plays a central role in assessing the effectiveness of training with a data set of a certain size. However, we still lack a condition for when the derived criterion (3.12) should be checked. Evaluating the criterion after each training epoch becomes expensive since the sample variance has to be constantly recomputed. Hence, we complement this approach with another heuristic, namely early stopping. This technique is a form of regularization commonly used in deep learning and basically indicates when a neural network starts to overfit the training data, see, e. g., [52, Sec. 1.1; 27, Sec. 7.8]. Usually, training would be terminated as soon as some generalization condition is not longer satisfied. In our case, however, early stopping triggers the evaluation of criterion (3.12). If this is not met, we generate data points and add them to the training set until we reach the size determined by (3.13). Then, we will continue training with the enlarged training set and repeat the procedure as soon as the early stopping criterion is reached again. When, at some point, the test (3.12) is satisfied, we guess that providing more data would not improve the NN model significantly. Training for longer would only increase the generalization loss, and thus we stop the optimization process. In this framework, (3.12) thus provides a convergence criterion for our Adam optimizer. In practice, we also terminate training when a given time limit or maximum number of epochs is exceeded.

3.3.5 Early Stopping

As already mentioned, the sample size heuristic of the previous section is combined with an early stopping criterion. This does not only delay the computation of the more expensive sample size criterion (3.12), but also the data generation until it is needed most urgently to prevent overfitting. Validation-based early stopping monitors some performance measure computed on the validation set D_{val}, and kicks in when a given condition indicating a degrading generalization is fulfilled, see [52, Sec. 1.2]. Hence, we require a suitable metric for evaluating the model performance, as well as a trigger that interrupts the training. In the following, we discuss good choices for both components, performance measure and stopping criterion, to find a heuristic fitting our purposes best.

In order to define a performance measure $E(\theta; D)$, we make use of the error metrics introduced in Section 3.2.4, i. e., RMAE_V, $\text{RML}^1{}_p$ and RMAE_u. There are several options such as selecting only one of them or using a weighted sum composed of several metrics, and the choice depends on our main goal. If the focus lies merely on the resulting control, we propose to use $E := \text{RMAE}_u$, but if an accurate value function approximation is desired, we recommend using $E := \text{RMAE}_V + \lambda \cdot \text{RML}^1{}_p$ where λ is the gradient regularization weight from Section 3.2.3.

The selected performance measure can be computed on all data sets, D_{train}, D_{val} and D_{test}. For a more concise notation, we define the training and validation errors

$E_{\text{train}}, E_{\text{val}} : \mathbb{N} \to \mathbb{R}$ at some epoch $i \in \mathbb{N}$ belonging to training round $j \in \mathbb{N}_0$ by

$$E_{\text{train}}(i) \coloneqq E(\theta^i; D^j_{\text{train}}) \quad \text{and} \quad E_{\text{val}}(i) \coloneqq E(\theta^i; D_{\text{val}})$$

where θ^i denotes the updated model parameter after epoch i is completed. We recall that the test set is kept out of the training process. Therefore, the test error cannot be used in the early stopping criterion and only serves as the final assessment of the model performance, that is

$$E_{\text{test}} \coloneqq E(\theta^*; D_{\text{test}}),$$

with θ^* being the parameter of the fully trained model.

Considering the stopping criteria, we first note that there is still little mathematical analysis available so that they are mostly chosen based on experimental investigations. We rely on the criteria presented in [52, Sec. 2.1] and extensively tested in [52, Sec. 3].

To begin with, we define the generalization loss (GL) at the i-th epoch as the relative increase of the validation error with respect to the lowest validation error so far, that is

$$\text{GL}(i) \coloneqq \frac{E_{\text{val}}(i)}{\min_{1 \le i' \le i} E_{\text{val}}(i')} - 1, \quad i \in \mathbb{N}.$$

A simple criterion would be to interrupt training when the generalization loss exceeds a given threshold $T > 0$, i. e.,

$$\text{stop at epoch } i \in \mathbb{N} \text{ if:} \quad \text{GL}(i) > T. \tag{3.14}$$

However, the optimization of NN models is stochastic and can be very noisy. This results in validation error curves with several local minima. Hence, we want to avoid getting stuck in the first shallow minimum by stopping training too early. As an example, we consider the error plot in Figure 3.3 where stopping at epoch i_1 keeps us from finding a more promising minimum. Therefore, we prefer a more elaborate condition that takes the training error into account. The following criterion is based on the observation that overfitting usually does not occur when the training error still decreases rapidly. To make this idea more rigorous we define the training progress (TP) corresponding to a training strip of length $\iota \in \mathbb{N}$ at the i-th epoch as

$$\text{TP}_\iota(i) \coloneqq \frac{\sum_{i'=i-\iota+1}^{i} E_{\text{train}}(i')}{\iota \cdot \min_{i-\iota+1 \le i' \le i} E_{\text{train}}(i')} - 1, \quad i \ge \iota.$$

This measure relates the average training error to the minimum training error during the strip. We note that the training progress is not only high when the training error drops down steeply but also in unstable phases of training in which the training error increases again. By suppressing early stopping during those training strips we overcome instabilities and potentially reach a deeper local minimum. This is also illustrated in Figure 3.3 by delaying stopping until epoch i_2. This approach results in the criterion

$$\text{stop at epoch } i \ge \iota \text{ if:} \quad \text{TP}_\iota(i) = 0 \quad \text{or} \quad \frac{\text{GL}(i)}{\text{TP}_\iota(i)} > T. \tag{3.15}$$

Alternatively, we can establish stopping criteria merely based on qualitative

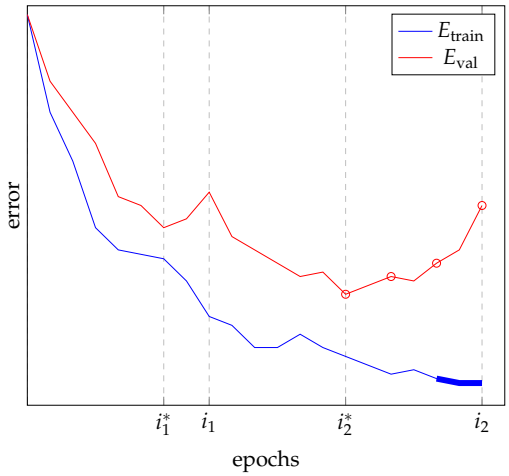

FIGURE 3.3: Exemplary training and validation error curve. Criterion (3.14) is likely to trigger early stopping at epoch i_1, whereas criteria (3.15) and (3.16) typically manage to overcome less promising local minima by stopping later at epoch i_2. Illustrating the strips used by the latter two heuristics, we thickened the section of the training curve relevant for (3.15) and marked the points on the validation curve checked by (3.16). The so far optimal epochs in both scenarios are denoted by i_1^* and i_2^*, respectively.

changes ignoring how large the increases of the validation error are. For instance, one could check if the validation error has grown during $S \in \mathbb{N}$ consecutive strips, yielding

$$\text{stop at epoch } i > S \cdot \iota \text{ if: } \quad E_{\text{val}}(i - s \cdot \iota) > E_{\text{val}}(i - (s+1) \cdot \iota) \quad \text{for all } s = 0, \dots, S-1. \tag{3.16}$$

The more training strips we choose to include, i. e., the larger S, the more likely it is to jump over local minima.

The selection of a stopping criterion among (3.14), (3.15) and (3.16), including the choice of their hyperparameters T, ι and S, depends strongly on the properties of the problem at hand. Since theoretical results are still scarce, one has to determine experimentally which variant yields an appropriate tradeoff between minimizing training time and maximizing generalization performance. What all of the criteria have in common, though, is that they may never be met during the whole training phase. For this reason, we also stop training when the training progress falls below some small threshold or, at the latest, after some maximum number of epoches is reached.

When the training has been interrupted by one of the above criteria, the NN model is likely to generalize worse than at some prior epoch. Rather than using the latest network parameters θ^i, we thus return the parameters $\theta^* := \theta^{i^*}$ of the epoch exhibiting the lowest validation error, i. e., for $i^* = \operatorname{argmin}_{1 \le i' \le i} E_{\text{val}}(i')$, cf. [27, Sec. 7.8]. Consequently, we have to save the model parameters whenever the performance on the validation set improves.

Employing early stopping during training adds further computational cost because we have to generate a validation set beforehand, and then evaluate the respective performance measure and stopping criterion at the end of each epoch. However, this cost is negligible, particularly when using a small validation set, in comparison

to the repeated computation of the variance estimation based criterion (3.12) on constantly growing training sets. Furthermore, tracking the validation error in combination with storing a copy of the best parameter θ^* allows us to go back to the most favorable model.

3.3.6 Adaptive Sampling

When criterion (3.12) is not satisfied, the training set has to be augmented until the size determined by (3.13) is achieved. Since some information about the value function is already encoded in the neural network, we can generate data in complicated regions where they seem to be most useful instead of sampling blindly from the semi-global domain. This adaptive sampling approach proposed in [49, Sec. 4.2] leads to a more data-efficient optimization which, considering the costs of data generation, may speed up training significantly.

More specifically, we certainly benefit from data that is placed where the gradient of the predicted value function V_x^{NN} is large because of two reasons: On the one hand, the value function tends to be more difficult to learn when it is steep. Having more data available of such complicated features of V facilitates learning them appropriately and, thus, improves the overall value function approximation V^{NN}. On the other hand, the control effort is likely to be large since the feedback control law depends explicitly on V_x^{NN} as apparent from (3.2). Therefore, we desire the NN controller to be particularly accurate there, which in turn requires a good prediction of V_x^{NN} near those regions.

However, we note that not all data can be sampled deliberately in the specified regions. In fact, only the initial conditions can be chosen directly, whereas all data points at later times have to be sampled along the trajectories starting from these. With this limitation in mind, we proceed in the following way: First, we sample several initial conditions

$$\{x_0^{(k)}\}_{k=1}^{K_{\mathrm{init}}} \subset \mathbb{X}_0$$

uniformly at random that will serve us as candidate points. Feeding them into the partially-trained neural network, we obtain the predicted value function and by AD its gradient $V_x^{\mathrm{NN}}(t_0, x_0^{(k)})$ for all $k = 1, \dots, K_{\mathrm{init}}$. This reveals the regions we are most interested in, namely where $\left\| V_x^{\mathrm{NN}}(t_0, \cdot) \right\|_2$ is large. Hence, we select $x_0^{(k^*)}$ where

$$k^* := \operatorname*{argmax}_{1 \le k \le K_{\mathrm{init}}} \left\| V_x^{\mathrm{NN}}(t_0, x_0^{(k)}) \right\|_2 .$$

Depending on the amount of new data that has to be generated, we may not only choose one but the top few initial conditions with the largest predicted gradient norms. Then, we proceed as usual by solving the BVP (2.20), yielding numerous data points along the optimal trajectories which are added to the data set.

3.4 Algorithmic Realization and Key Strengths

Summarizing all the techniques explored in this chapter, we establish Algorithm 1 which represents the adaptive deep learning approach for general optimal control problems. Steps 1, 8 and 10 contain the generation of all data sets for training D_{train}^j, validation D_{val} and testing D_{test} with the respective methods for sampling initial

conditions and for initialization of the BVP solver. As a central element of the algorithm, the supervised deep learning problem is solved in the 3rd step. This means that we optimize the parameters θ of a neural network function F^{NN} such that the corresponding model V^{NN} fits the value function best. Employing a stochastic optimizer operating on batches, we train the NN model in epochs indexed by $i \in \mathbb{N}_0$. Additionally, the training phase is divided in several training rounds $j \in \mathbb{N}_0$ according to the training set that is available during the current epoch. Furthermore, the progressive data generation is guided by several heuristical tests including the early stopping criterion in step 4, the variance estimation based convergence criterion in step 6, as well as the data set size selection rule in step 7. Besides that, the heuristic for the adaptive selection of initial conditions is included in the 8th step. Throughout the optimization process, the model parameter θ^* with the smallest validation error $E_{\mathrm{val}}(i^*)$ is stored in the 5th step, so that it can be retrieved in step 9 when training is completed. Finally, in the last step, the model accuracy is evaluated on unseen data contained in the test set yielding the test error E_{test}. This enables us to measure the generalization performance of the final model.

Algorithm 1 Adaptive Deep Learning

1. Generate initial training set D^0_{train} and validation set D_{val} by solving BVP (2.20) for random initial conditions using time-marching
2. Initialize NN with model parameter $\theta \in \mathbb{R}^R$; set epoch $i \leftarrow 0$, training round $j \leftarrow 0$, as well as optimal epoch $i^* \leftarrow 0$ and corresponding parameter $\theta^* \leftarrow \theta$

while not converged **do**
  while not early stopped **do**
    3. Update θ by training NN on D^j_{train} for one epoch, i. e., by solving the learning problem (3.7); set $i \leftarrow i + 1$
    4. Track training error $E_{\mathrm{train}}(i)$ and validation error $E_{\mathrm{val}}(i)$; evaluate early stopping criterion, e. g., (3.15)
    if $E_{\mathrm{val}}(i) < E_{\mathrm{val}}(i^*)$ **then**
      5. Set $i^* \leftarrow i$ and $\theta^* \leftarrow \theta$
    end if
  end while
  6. Evaluate convergence criterion (3.12)
  if not converged **then**
    7. Determine data set size $|D^{j+1}_{\mathrm{train}}|$ by (3.13)
    8. Augment training set D^j_{train} by solving BVP (2.20) for adaptively sampled initial conditions using NN warm start; set $j \leftarrow j + 1$
  end if
end while
9. Load NN with optimal parameter θ^*
10. Generate test set D_{test} by solving BVP (2.20) for random initial conditions using NN warm start
11. Compute test error E_{test}; evaluate generalization performance of NN model

In practical application, step 1 to 9 of Algorithm 1 are carried out entirely offline. Hence, the largest part of the computational cost coming from the data generation and optimization process is incurred before runtime. Since the fully trained model V^{NN} is now available, we can pass any given point in time $t \in [t_0, t_f]$ and measurement of the state $x \in \mathbb{X}$ through the neural network and perform AD to obtain the gradient $V^{\mathrm{NN}}_x(t, x)$ which is required for calculating the feedback law (3.2).

This online computation is inexpensive and yields the closed-loop NN controller $\mathbf{u}^{\mathrm{NN}}(t, x)$ in real-time. Notably, the speed of online control computation remains unimpaired when the OCP is made more difficult, e. g., by increasing the dimensionality or adding non-linearity to the system dynamics, because the additional computational load is tackled mainly offline. Furthermore, the causality-free BVP solver [38] keeps the costs for data generation tractable since it operates on single points without the need for a mesh discretizing the initial condition domain. By leveraging the special ability of deep neural networks to learn from complex and high-dimensional data, the optimization of the model does not form a major obstacle either. Hence, the offline computation remains feasible, suggesting that our approach has great potential to provide closed-loop solutions to problems that are both highly scalable and complex.

Steps 10 and 11 allow us to empirically evaluate the performance of a model on independently generated data by exploiting the causality-free property of the BVP solver once again. The main benefit of this data-driven verification is that we do not require an analytical closed-loop solution, which may be unavailable for more complicated problems. Hence, we are able to compare models arising from separate training runs based on different hyperparameter choices for a wide range of problems. This is especially useful for selecting the best model from among various candidates which is a common practice in machine learning.

Our intention is to provide a framework which is flexible enough, so that it can be adapted to various settings and specific properties of the problem at hand. Therefore, Algorithm 1 does not specify all implementation details. In this way, we leave sufficient freedom of choice with respect to certain key aspects, while suggesting promising state-of-the-art techniques to fill in these gaps. To name the most important choices, we propose to use

- the value function approximation model V^{NN} defined by (3.1) which ensures $V^{\mathrm{NN}}(t_f, \cdot) = V(t_f, \cdot)$ by construction,
- the residual neural network architecture F^{ResNet} defined by (3.4) as central component of the above approximator,
- the Adam algorithm [39, Alg. 1] as batch optimization method for training the neural network,
- the gradient cost (3.6) as penalty term for physics-informed regularization and simultaneously L^2 regularization, as well as
- the early stopping criterion (3.15) incorporating generalization loss and training progress.

We also note that the algorithm's performance depends to some extent on an adequate tuning of the hyperparameters. These include the network's depth, width and activation function, as well as the batch size, the optimizer's learning rate and the weights for regularization. Even the heuristical tests require a sensitive selection of the respective tolerance or threshold parameters. Besides that, one has to choose a suitable performance measure for calculating the training, validation and test errors. Furthermore, we need to decide on the size of the initial training set, the validation and test set. The elaborate guidelines for progressive data generation determine the sizes of the later training sets but the number of candidate initial conditions for adaptive sampling still has to be chosen beforehand.

The algorithm presented here is strongly inspired by the approach developed in [49]. However, we propose some significant modifications and refinements, particularly with respect to the choices listed above, as already mentioned in Section 1.3. In the following, we summarize how these suggestions support our endeavor to learn a robust and accurate value function approximation while keeping computational costs low.

Exploiting prior information: We pursue several strategies to leverage as much prior information provided by the problem as possible. The first one is the above choice of the approximator V^{NN} which guarantees that the final condition of the value function is exactly met by the model independently of its network parameters θ. Besides that, the NN warm start as initialization technique for the BVP solver utilizes information about the problem which has already been gathered up to this point. The last and certainly most important use of structured prior information happens through the physics-informed learning which is guided by the known underlying relation of the value function to the costate.

Using data efficiently: Furthermore, we try to be as data-efficient as possible since generating large data sets becomes expensive. One technique contributing to this aspiration is again the physics-informed DL approach because it incorporates the costate data into the optimization process which is generated as a byproduct of solving the BVP anyway. In this sense, the data which the model can learn from is doubled without any additional computational cost. Another way to make efficient use of small data is to sample it adaptively in regions which are more complicated to learn or where we desire a higher level of accuracy. In our case, we select initial conditions for which the gradient of the value function is large. Besides that, we delay the data generation by employing an early stopping technique so that data is only added when the NN model starts to overfit and, thus, benefits most from seeing new data.

Promoting generalization: A general challenge in deep learning is to avoid overfitting, i. e., to ensure that the optimized NN model generalizes to unknown data on $\mathbb{X} \subset \mathbb{R}^n$ and perhaps even beyond this semi-global domain. As already mentioned, an early stopping criterion indicates when generalization loss is occurring so that training can be interrupted in order to provide more data. Keeping track of the validation error also allows us to determine the model parameter θ^* that generalizes best thus far. Furthermore, we use two regularization techniques based on penalty terms which are added to the cost function: conventional L^2 regularization also known as weight decay and gradient regularization as part of the physics-informed training. The former encourages small weight matrices and, thus, leads to simpler model, while the latter promotes an accurate approximation of the value function gradient V_x^{NN}. Last but not least, by applying a stochastic optimizer operating on mini-batches such as Adam, we introduce noise into the training. This facilitates finding a flat minimizer in the cost function's landscape and successfully overcome the so-called generalization gap.

Reducing the sensitivity to hyperparameter tuning: Choosing Adam as optimization algorithm also comes with the benefit of individual adaptive learning rates which makes it more robust to hyperparameter choices for training. Concerning the network's hyperparameters, we note that using a residual architecture generally

increases the stability of the model by preventing vanishing gradients and representational bottlenecks. This particularly enables us to choose substantially deeper neural networks which have a greater capacity to learn complex patterns. Besides that, heuristics like the data set size selection rule help us to guide our choices with respect to data generation.

Algorithm 1 together with the additionally listed proposals is implemented for the exemplary optimal control problem presented in Chapter 4 and numerically tested in Chapter 5. Since we are only able to show results for a few problem, model and training configurations, we encourage to continue the exploration of further variants of the adaptive deep learning algorithm by testing alternative options for the approximation model, the neural network design and its training, as well as for the dynamic data generation.

Chapter 4

Application to Reaction-Diffusion System

4.1 Optimal Control of Reaction-Diffusion System

The approach presented in Chapter 3 will now be applied to specific example problems arising from the heat or diffusion equation given by

$$X_t(t,\omega) = a\Delta X(t,\omega) \quad \text{for a. a. } (t,\omega) \in [t_0,t_f] \times \Omega, \tag{4.1}$$

see, e. g., [8, Sec. 10.1; 18, Sec. 6.1.3; 54, Sec. 12.2]. Here, the scalar-valued state $X : [t_0,t_f] \times \Omega \to \mathbb{R}$ evolves on a spatial domain $\Omega \subseteq \mathbb{R}^d$ with boundary $\partial\Omega$ according to a constant diffusion coefficient $a > 0$ over a finite time interval $[t_0,t_f] \subset \mathbb{R}$. We denote $X_t := \frac{\partial X}{\partial t}$ and the Laplace operator Δ is defined as the sum of second-order partial derivatives w. r. t. the spatial variable ω, i. e.,

$$\Delta X = \sum_{k=1}^{d} \frac{\partial^2 X}{\partial \omega_k^2}. \tag{4.2}$$

Equation (4.1) is a prototypical parabolic partial differential equation (PDE) which can be transformed into a system of ordinary differential equations (ODEs) via discretization in space.

In order to introduce some non-linearity into the system, we add a reaction term $R : \mathbb{R} \to \mathbb{R}$ applied to the state to the right-hand side of the equation. Besides that, we include a scalar-valued control $\mathbf{u} : [t_0,t_f] \to \mathbb{R}$ entering the system via a source or sink term $S : \Omega \times \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. Both terms, R and S, are weighted by some constants $b, c > 0$, respectively. This yields

$$\begin{aligned} X_t(t,\omega) = a\Delta X(t,\omega) + bR(X(t,\omega)) + cS(\omega,\mathbf{u}(t),X(t,\omega)) \\ \text{for a. a. } (t,\omega) \in [t_0,t_f] \times \Omega, \end{aligned} \tag{4.3}$$

resulting in a controlled reaction-diffusion system, see generally [42, Sec. 1.1; 20, Preface and General Introduction; 8, Comments on Ch. 10 4.(ii)]. We note that although the control is a function of time only, its influence on the system might still be dependent on the space and the current state since we allow S to contain those variables as arguments.

Furthermore, the state has to satisfy homogeneous Dirichlet boundary conditions given by

$$\begin{cases} X(t,\omega) = 0 & \text{on } [t_0,t_f] \times \partial\Omega \\ X(t_0,\omega) = X_0(\omega) & \text{on } \Omega, \end{cases} \tag{4.4}$$

where $X_0 : \Omega \to \mathbb{R}$ is the initial or Cauchy data, see [8, Sec. 10.1; 18, Sec. 6.1.3] and also [54, Sec. 12.2].

Our aim is to steer the state X to a desired target $Y : \Omega \to \mathbb{R}$ expending minimum control effort. We consider a quadratic cost function of the form

$$\begin{aligned} J(\mathbf{u}, X) = \int_{t_0}^{t_f} \left[\int_\Omega \alpha |X(t,\omega) - Y(\omega)|^2 \, \mathrm{d}\omega + \beta |\mathbf{u}(t)|^2 \right] \mathrm{d}t \\ + \int_\Omega \gamma |X(t_f,\omega) - Y(\omega)|^2 \, \mathrm{d}\omega \end{aligned} \tag{4.5}$$

inspired by [56, Sec. 5; 40, Sec. 11.5]. The weights $\alpha, \beta, \gamma > 0$ determine how much the running state cost, the running control cost and the final cost should be penalized. If we interpret X as a function of t with values in the space whose elements are functions mapping from Ω to $\mathbb{R}$, see [8, Sec. 10.1; 18, Sec. 6.1.1], we can reformulate (4.5) as

$$J(\mathbf{u}, X) = \int_{t_0}^{t_f} \left[\alpha \left\| X(t,\cdot) - Y(\cdot) \right\|_{L^2(\Omega)}^2 + \beta |\mathbf{u}(t)|^2 \right] \mathrm{d}t + \gamma \left\| X(t_f,\cdot) - Y(\cdot) \right\|_{L^2(\Omega)}^2 . \tag{4.6}$$

For the cost to be finite, we clearly require $\mathbf{u} \in L^2([t_0, t_f])$, as well as $Y \in L^2(\Omega)$ and $X \in H^1(t_0, t_f; L^2(\Omega)) \coloneqq W^{1,2}(t_0, t_f; L^2(\Omega))$. However, we even assume $\mathbf{u} \in L^\infty$ and $X \in W^{1,\infty}$ so that after discretization the setting of the theoretical considerations in Chapter 2 is attained.

In our numerical experiments, we will proceed in three steps advancing from the easier to the more complicated problem:

1. considering the linear-quadratic case emerging from an additive control,
2. adding a destabilizing non-linear reaction term, and
3. introducing a state-dependent, e. g. bilinear, control.

The first example serves mainly as a starting point for testing the functionality of our adaptive deep learning algorithm since an analytical solution via the Riccati differential equation is available, see Section 4.5. In the latter two cases, the performance of the algorithm for problems of increasing difficulty, in which traditional methods fail, is evaluated.

4.1.1 Additive Control

First, we ignore the reaction term setting $R \equiv 0$ and consider an additive state-independent control acting on some subset $\Omega_S \subseteq \Omega$. Hence, the source or sink term is given by

$$S_1(\omega, \mathbf{u}(t)) \coloneqq \mathbf{1}_{\Omega_S}(\omega) \mathbf{u}(t), \tag{4.7}$$

where $\mathbf{1}_{\Omega_S} : \Omega \to \{0, 1\}$ is the indicator or characteristic function defined by

$$\mathbf{1}_{\Omega_S}(\omega) = \begin{cases} 1, & \omega \in \Omega_S \\ 0, & \text{else.} \end{cases}$$

Such a control term is also used in [49, Sec. 6; 41, Sec. 8; 34, Sec. 2.1] and describes the effect of an externally added and locally restricted source or sink while the principal

intrinsic properties of the system remain unchanged. By discretization (4.3) becomes a linear ODE coupled with a quadratic cost, leading to the extensively studied linear-quadratic case.

4.1.2 Non-linear Reaction

Now, we introduce a destabilizing non-linear reaction term into the heat equation while keeping the term S_1 unchanged. We choose the hyperbolic and logistic growth rates as reaction terms given by

$$R_1(X(t,\omega)) \coloneqq X(t,\omega)^2 \quad \text{and} \quad R_2(X(t,\omega)) \coloneqq X(t,\omega)(1 - X(t,\omega)), \tag{4.8}$$

respectively, see, e. g., [7, Sec. 2.1; 42, Sec. 1.2.1]. The system (4.3) with $S \equiv 0$ and $R = R_2$ in a one-dimensional space is also famously known as Fisher's equation, introduced and investigated by Fisher as well as Kolmogorov, Petrovsky and Piskunov, see [20, Sec. 2.1; 42, Sec. 5.2].

4.1.3 Bilinear Control

Last but not least, we replace the additive control with localized support in (4.7) by a multiplicative also known as bilinear control

$$S_2(\mathbf{u}(t), X(t,\omega)) \coloneqq \mathbf{u}(t)X(t,\omega) \tag{4.9}$$

thoroughly studied in [26, Sec. 1.1]. Thus, the control enters the differential equation as a coefficient of the state and therefore changes the reaction rate.

4.2 Examples for Real-World Problems

Reaction-diffusion systems of the form (4.3) are used to describe phenomena occurring in a wide range of fields.

In physics, it typically models the heat distribution in some material, where the state X represents the temperature and the coefficient a incorporates the thermal conductivity, the specific heat capacity and the density of the material, see, e. g., [8, Sec. 10.1; 18, Sec. 6.1.3; 54, Sec. 12.2]. In this context, the additive control term (4.7) introduces the effect of an external local heat source which serves the purpose of guiding the temperature towards a target profile, see [56, Sec. 5].

Another diffusive process is studied in ecology, namely the spreading of biological populations with X describing the population density, see, for instance, [7, Sec. 2; 42, Sec. 1.2.1]. In this application, the diffusion equation is often complemented with a reaction term modelling the population growth according to a specified birth rate such as the hyperbolic or logistic one defined in (4.8). Furthermore, the population density might also be regulated externally via a control term. New population members could simply be added or removed at a certain time and location resulting again in an additive control. If the population density is reduced by e. g. harvesting or pest control, however, a bilinear control (4.9) is more suitable in order to capture the dependence of the effectiveness on the number of population members.

A last example, taken from chemistry, are reactions of one or more chemical substances whose concentration constitutes the state of the system, see [42, Sec. 1.2.5]. Diffusion causes the substances to spread out while the chemical reactions transform the substances into each other. Adding a bilinear control takes the effect of catalysts

into account that can accelerate or slow down the chemical reactions as described in [26, Sec. 1.1].

4.3 Spatial Discretization

So far, we established a PDE-constrained optimal control problem (OCP) of the form

$$\min_{(\mathbf{u},X)} (4.6) \quad \text{s. t. } (4.3) \text{ and } (4.4). \tag{4.10}$$

In order to fit this into the abstract framework developed in Chapter 2, we approximate the state by discretizing it in space yielding an ODE-constrained problem.

For simplicity, we will only treat the one-dimensional case, i. e., $d = 1$, in which Ω is an open and bounded interval $(\omega_{lb}, \omega_{ub})$. Using an equidistant mesh with $n+2$ nodes, that is

$$\omega_j = \omega_{lb} + jh(\omega_{ub} - \omega_{lb}) \quad \text{where } j = 0, \dots, n+1 \quad \text{and} \quad h = \frac{1}{n+1}, \tag{4.11}$$

and denoting the approximation of the state at these points by $\mathbf{x}_j(t) \approx X(t, \omega_j)$, the Laplace operator (4.2) can be readily approximated by the second-order central difference scheme, i. e.,

$$\Delta X(t, \omega_j) = \frac{\partial^2 X}{\partial \omega^2}(t, \omega_j) \approx \frac{\mathbf{x}_{j-1}(t) - 2\mathbf{x}_j(t) + \mathbf{x}_{j+1}(t)}{h^2},$$

see [54, Sec. 12.2.1] and also [56, Sec. 5]. The controlled PDE (4.3) then becomes a n-dimensional system of ODEs given by

$$\dot{\mathbf{x}}_j(t) = \frac{a}{h^2}(\mathbf{x}_{j-1}(t) - 2\mathbf{x}_j(t) + \mathbf{x}_{j+1}(t)) + bR(\mathbf{x}_j(t)) + cS(\omega_j, \mathbf{u}(t), \mathbf{x}_j(t))$$
$$\text{for a. a. } t \in [t_0, t_f] \text{ and all } j = 1, \dots, n$$

with boundary conditions

$$\begin{cases} \mathbf{x}_0(t) = \mathbf{x}_{n+1}(t) = 0 & \text{for a. a. } t \in [t_0, t_f] \\ \mathbf{x}_j(t_0) = X_0(\omega_j) & \text{for all } j = 1, \dots, n \end{cases}$$

resulting from discretizing (4.4).

In the following, we are ignoring the error arising from spatial discretization of the PDE, and just consider $\mathbf{x} := (\mathbf{x}_1, \dots, \mathbf{x}_n)^\top$ as the new state. With the above assumptions, we have the same function spaces (2.2) and (2.3) of our previous considerations, that is $\mathbf{u} \in L^\infty(t_0, t_f; \mathbb{R}) = \mathcal{U}$ with $m = 1$ and $\mathbf{x} \in W^{1,\infty}(t_0, t_f; \mathbb{R}^n) = \mathcal{X}$. We note that the dimension of the state space corresponds to the number of mesh nodes. Thus, mesh refinement quickly leads to high-dimensional HJB equations suffering from the curse of dimensionality.

Now the original OCP (4.10) can be brought into the form of Problem 2.2. For a more condensed notation, we define the discretized target $y := (Y(\omega_1), \dots, Y(\omega_n))^\top$ and initial condition $x_0 := (X_0(\omega_1), \dots, X_0(\omega_n))^\top$, as well as the $n \times n$ matrix Q for

the discretized diffusion term by

$$Q = \frac{1}{h^2}\begin{pmatrix} -2 & 1 & & & 0 \\ 1 & -2 & & & \\ & \ddots & \ddots & \ddots & \\ & & & -2 & 1 \\ 0 & & & 1 & -2 \end{pmatrix}.$$

Approximating the spatial integrals in the cost function (4.6) by sums over the discretized variables via quadrature, the L^2-norm is replaced by the Euclidean norm denoted by $\|\cdot\|_2$. Ultimately, this leads to the ODE-constrained problem

$$\begin{aligned} \min_{(\mathbf{u},\mathbf{x})\in\mathcal{U}\times\mathcal{X}} & \int_{t_0}^{t_f} \left[\alpha h \left\|\mathbf{x}(t) - y\right\|_2^2 + \beta|\mathbf{u}(t)|^2\right] \mathrm{d}t + \gamma h \left\|\mathbf{x}(t_f) - y\right\|_2^2 \\ \text{subject to} \quad & \dot{\mathbf{x}}(t) = aQ\mathbf{x}(t) + bR(\mathbf{x}(t)) + cS(\mathbf{u}(t),\mathbf{x}(t)) \quad \text{for a. a. } t \in [t_0, t_f]; \\ & \mathbf{x}(t_0) = x_0. \end{aligned} \tag{4.12}$$

For notational convenience, we still use the variables R and S, and essentially refer to their componentwise application when given a multi-dimensional argument. More precisely, we have the reaction terms

$$R_1(\mathbf{x}(t)) = \mathbf{x}(t) \odot \mathbf{x}(t) \quad \text{and} \quad R_2(\mathbf{x}(t)) = \mathbf{x}(t) \odot (\mathbb{1} - \mathbf{x}(t)), \tag{4.13}$$

where $\odot$ denotes the Hadamard product and $\mathbb{1}$ is the vector of all ones, combined with the source and sink terms

$$S_1(\mathbf{u}(t)) = \mathbf{u}(t)\mathbb{1}_{\Omega_S} \quad \text{and} \quad S_2(\mathbf{u}(t),\mathbf{x}(t)) = \mathbf{u}(t)\mathbf{x}(t), \tag{4.14}$$

where $\mathbb{1}_{\Omega_S} = (\mathbf{1}_{\Omega_S}(\omega_1), \ldots, \mathbf{1}_{\Omega_S}(\omega_n))^\top$ is the discretized indicator function.

This transformation of the OCP can be achieved similarly in higher dimensional spaces, i. e., for $d \geq 2$, when Ω is an open, bounded, connected set with sufficiently smooth boundary $\partial\Omega$, as defined in [18, Def. 1.46]. Then, one could use the finite difference method or the finite element method as applied in [18, Sec. 6.1.3] for discretization in space. But since the example problem formulation in (4.12) already provides a scalable state, allowing us to test our approach for solving high-dimensional HJB equations, we omit further investigation concerning more general spatial domains.

4.4 Boundary Value Problem and Neural Network Control

Solving the discretized OCP (4.12) with the adaptive deep learning algorithm developed in Section 3.4, we need to be able to generate data before and during the network training. This is accomplished by computing open-loop solutions via the boundary value problem (BVP) (2.20) consisting of three ODEs with boundary conditions. In the following, these will be specified for our example of controlled reaction-diffusion systems.

The first ODE is simply the discretized state equation with initial condition, appearing as the constraint in (4.12). The second one is the adjoint equation with boundary condition at final time. Differentiating the Hamiltonian, see Definition 2.3,

w. r. t. the state variable, we obtain

$$\begin{aligned}\dot{\mathbf{p}}(t) = -2\alpha h(\mathbf{x}(t) - y) - aQ^\top \mathbf{p}(t) - bR_x^\top(\mathbf{x}(t))\mathbf{p}(t) - cS_x^\top(\mathbf{x}(t))\mathbf{p}(t) \\ \text{for a. a. } t \in [t_0, t_f]; \quad \mathbf{p}(t_f) = 2\gamma h(\mathbf{x}(t_f) - y),\end{aligned} \tag{4.15}$$

where $R_x := \left(\frac{\partial R_j}{\partial x_k}\right)_{j,k=1,\dots,n}$ is the Jacobian matrix of the discretized version of the reaction term. For our specific choices stated in (4.13), these Jacobians are diagonal matrices leading to a componentwise multiplication with the costate, i. e.,

$$R_{1x}^\top(\mathbf{x}(t))\mathbf{p}(t) = 2\mathbf{x}(t) \odot \mathbf{p}(t) \quad \text{and} \quad R_{2x}^\top(\mathbf{x}(t))\mathbf{p}(t) = (\mathbb{1} - 2\mathbf{x}(t)) \odot \mathbf{p}(t).$$

Regarding the source and sink term, S_x is defined analogously yielding

$$S_{1x}^\top(\mathbf{x}(t))\mathbf{p}(t) = 0 \quad \text{and} \quad S_{2x}^\top(\mathbf{u}(t), \mathbf{x}(t))\mathbf{p}(t) = \mathbf{u}(t)\mathbf{p}(t)$$

for the additive and bilinear control in (4.14).

The state and adjoint equation are complemented by the characteristic ODE for the value function along the optimal trajectory given by

$$\dot{\mathbf{v}}(t) = -\alpha h \left\|\mathbf{x}(t) - y\right\|_2^2 - \beta|\mathbf{u}(t)|^2 \quad \text{for a. a. } t \in [t_0, t_f]; \quad \mathbf{v}(t_f) = \gamma h \left\|\mathbf{x}(t_f) - y\right\|_2^2. \tag{4.16}$$

This system of differential equations is evaluated along the candidate optimal control provided by the Hamiltonian minimization condition (2.10) from Pontryagin's Minimum Principle (PMP). For our example, this condition reduces to

$$\mathbf{u}^*(t; x_0) \in \operatorname*{argmin}_{u \in \mathbb{R}} \{\beta u^2 + c\mathbf{p}^*(t) \cdot S(u, \mathbf{x}^*(t))\} \quad \text{for a. a. } t \in [t_0, t_f].$$

Depending on the chosen source and sink term, we therefore get the open-loop solution

$$\mathbf{u}^*(t; x_0) = \begin{cases} -\frac{c}{2\beta}\mathbf{p}^*(t) \cdot \mathbb{1}_{\Omega_S}, & \text{if } S = S_1 \\ -\frac{c}{2\beta}\mathbf{p}^*(t) \cdot \mathbf{x}^*(t), & \text{if } S = S_2. \end{cases}$$

By solving the above BVP for several initial conditions x_0 with a causality-free algorithm, we are able to collect data for the value function and the costate. Following the approach of adaptive deep learning, we train a neural network (NN) yielding a value function approximation V^{NN}. The calculation of the NN feedback controller is then given by the formula (3.2) which is based on the Hamiltonian minimization condition arising in the context of Hamilton-Jacobi-Bellman (HJB) equations. Analogously to the previously considered minimization condition, we obtain

$$\mathbf{u}^{\mathrm{NN}}(t, x) = \begin{cases} -\frac{c}{2\beta}V_x^{\mathrm{NN}}(t, x) \cdot \mathbb{1}_{\Omega_S}, & \text{if } S = S_1 \\ -\frac{c}{2\beta}V_x^{\mathrm{NN}}(t, x) \cdot x, & \text{if } S = S_2 \end{cases} \tag{4.17}$$

for all $t \in [t_0, t_f]$ and all x in some semi-global domain $\mathbb{X} \subset \mathbb{R}^n$.

4.5 Linear-Quadratic Regulator

An alternative method of computing closed-loop solutions for a linear-quadratic problem relies on the Riccati differential equation (RDE), an ordinary matrix differential equation. The resulting feedback controller is called a linear quadratic regulator (LQR) and has received a lot of attention in the literature such as [44, Sec. 6.1; 55, Sec. 2.1; 57, Sec. 8.2; 43, Ch. 1]. Besides approximation errors due to numerical integration, this approach yields an analytical solution. Particularly in the presence of noise, which is not taken into account by the open-loop control analyzed in Section 2.2, the LQR controller serves as a baseline to evaluate the performance of the NN controller.

In the linear-quadratic case presented in Section 4.1.1 with $R \equiv 0$ and $S = S_1$, and for a vanishing target, i. e., $Y \equiv 0$, the discretized OCP (4.12) can be reformulated as a standard LQR problem

$$\begin{aligned} &\min_{(\mathbf{u},\mathbf{x})\in\mathcal{U}\times\mathcal{X}} \int_{t_0}^{t_f} \left[\mathbf{x}^\top(t)A\mathbf{x}(t) + \mathbf{u}^\top(t)B\mathbf{u}(t)\right] \mathrm{d}t + \mathbf{x}^\top(t_f)C\mathbf{x}(t_f) \\ &\text{subject to} \quad \dot{\mathbf{x}}(t) = F\mathbf{x}(t) + G\mathbf{u}(t) \quad \text{for a. a. } t \in [t_0, t_f]; \quad \mathbf{x}(t_0) = x_0, \end{aligned} \tag{4.18}$$

where A and C are $n \times n$ diagonal matrices containing αh and γh, respectively, as their entries. Together with $B = \beta$ interpreted as a one-dimensional matrix, these matrices specify the quadratic cost. In addition, we have $F = aQ$ and $G = c\mathbb{1}_{\Omega_S}$ forming the linear ODE.

Clearly, A and C fulfill the conditions of being symmetric positive semidefinite matrices and B being a non-zero scalar is trivially symmetric positive definite and therefore invertible. Hence, we know from [55, Thm. 2.1.1; 57, Thm. 37] that the solution to the RDE

$$\dot{P}(t) = -P(t)F - F^\top P(t) - A + P(t)GB^{-1}G^\top P(t), \quad P(t_f) = C$$

exists on the entire interval $[t_0, t_f]$ and is symmetric positive semidefinite. Furthermore, the linear feedback law is given by

$$\mathbf{u}^*(t) = -B^{-1}G^\top P(t)\mathbf{x}^*(t). \tag{4.19}$$

The corresponding costate and the value function can also be expressed in terms of $P(t)$, that is

$$\mathbf{p}^*(t) = -2P(t)\mathbf{x}^*(t) \quad \text{and} \quad V(t,x) = x^\top P(t)x,$$

see also [44, Secs. 6.1.1 and 6.1.3; 55, Ex. 5.1.1].

For a non-zero target, we first have to transform the state

$$\tilde{\mathbf{x}}(t) = \begin{pmatrix} \mathbf{x}(t) - y \\ y \end{pmatrix} \in \mathbb{R}^{2n}$$

and accordingly the matrices

$$\tilde{A} = \begin{pmatrix} A & 0 \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{2n\times 2n}, \quad \tilde{C} = \begin{pmatrix} C & 0 \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{2n\times 2n}$$
$$\tilde{F} = \begin{pmatrix} F & F \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{2n\times 2n}, \quad \tilde{G} = \begin{pmatrix} G \\ 0 \end{pmatrix} \in \mathbb{R}^{2n\times n}.$$

Then, problem (4.12) can also be written in the form (4.18) by replacing the state and matrices by their respective transformations. Since $\tilde{A}$ and $\tilde{C}$ are again symmetric positive semidefinite, the above results can be applied analogously.

If a non-linear reaction term is added as described in Section 4.1.2, the closed-loop solution can still be approximated by computing the LQR for the linearized system as proposed in [41, Sec. 8]. Considering the linear approximation at the origin, that is the first-order Taylor polynomial in the vector of all zeros

$$R(\mathbf{x}(t)) \approx R(0) + R_x(0)\mathbf{x}(t),$$

we find that the discretized dynamics remain unchanged for the hyperbolic growth rate since $R_1(\mathbf{x}(t)) \approx 0$. However, the linearized logistic growth rate $R_2(\mathbf{x}(t)) \approx \mathbf{x}(t)$ modifies the matrix F of the LQR problem (4.18) by

$$F = aQ + bI,$$

where $I \in \mathbb{R}^{n\times n}$ denotes the unit matrix. Applying the Riccati theory from above, the linearization based feedback law is obtained by (4.19).

To sum up, the RDE approach yields LQR controllers which are optimal for linear-quadratic OCPs and deliver at least an approximate solution for non-linear problems via the linearization of their state dynamics.

4.6 Implementation

To demonstrate how an exemplary problem can be solved numerically within the adaptive deep learning framework, we provide an implementation of Algorithm 1, which is specifically tailored to the controlled reaction-diffusion system presented in this chapter. More precisely, the program addresses the ODE-constrained OCP (4.12) resulting from spatial discretization and generates data based on the equations (4.15) and (4.16) for the costate and value function, respectively. Furthermore, we implement and visualize the network training and the computation of the NN controller according to the feedback law (4.17). Besides an empirical evaluation of the model accuracy, we compare the closed-loop solution obtained by the NN model with the one resulting from the LQR approach calculated via (4.19). In simulations, we can then observe the resulting feedback controls, as well as state and costate trajectories for a particular initial condition. Additionally, we consider the open-loop solution provided by the BVP solver and the evolution of the uncontrolled system for the same initial condition and draw a comparison of the running, final and overall costs across all solutions. By adding noise to the dynamical system, we can further examine how well the controllers respond to disturbances. Lastly, we also pay special attention to the initialization for solving the boundary value problem in order to guarantee an efficient data generation in the pre-training, training and post-training phases.

All the mentioned functionalities are included in the program available on GitHub at the following link:

`https://github.com/ElisaGiesecke/AdaDL-for-HJB-Equations`.

The code is written in Python (version 3.9.13) following the modular programming approach. The modular structure allows us to modify the choices with respect to, e. g., the network architecture and the training procedure, as well as for performance tests and simulations. Moreover, new pieces of code can be added easily, so that

the program can be extended to other optimal control problems and novel solution techniques.

We make use of the SciPy library (version 1.9.1), particularly of the `scipy.integrate` package containing the functions `solve_bvp` and `solve_ivp`. The former solves the boundary value problem in the context of data generation as already mentioned in Section 3.3.1, whereas the latter integrates the state equation for given controllers in order to simulate the corresponding trajectories.

Besides that, we import the open-source machine learning library PyTorch (version 1.13.0) which offers automatic differentiation (AD) capabilities and supports GPU acceleration among other high-level features. For an efficient implementation of neural networks we mainly rely on the building blocks provided in the package `torch.nn` including the base class `Module` for all NN models as well as pre-implemented layer, activation and loss functions. In addition, we subclass `Dataset` and use the respective `DataLoader` from `torch.utils.data` to automatically collate previously generated data points into batches which can then be fed to the NN model. Lastly, we also apply popular optimization algorithms implemented in the `torch.optim` package.

Our functions and classes which are grouped into separate `.py` files are executed by the enclosed Jupyter Notebooks. This interactive interface allows the user to gain an insight into the main steps of the algorithm: how the control problem is set up and how data is generated, but also in which way the networks are constructed, trained and evaluated. Besides that, the notebook enables us to flexibly choose the experiment setting, adjust hyperparameters and create simulations on the fly.

In the notebook `.ipynb` files, we set the configurations for three central components of the program, namely the problem, model and training. Additionally, we can make key choices for the progressive and adaptive data generation, as well as for the evaluation of the final model. During an experimental run, the results are not only displayed within the notebook, but also saved externally along with the generated data. Moreover, the problem, model and training configurations and their respective class instances are stored offline, so that they can be reloaded for later usage. Figure 4.1 portrays the building blocks of the program with the most important experiment choices and output files. In the following, we shortly comment on each block.

The basis for the experiment is formed by the chosen problem type. Although the program is so far designed for only one-dimensional domains $\Omega = (\omega_{lb}, \omega_{ub})$, the problem is still scalable since the spatial discretization given by (4.11) can be refined as desired. Hence, we are able to numerically test the algorithm's performance also for high-dimensional systems. Furthermore, the program allows to arbitrarily combine various control and reaction terms. More specifically, we provide an implementation of the additive and bilinear control, S_1 and S_2 in (4.14), and offer the option to add hyperbolic and logistic growth rates, R_1 and R_2 in (4.13), as non-linear reactions.

Concerning the data generation, we note that the data set size cannot be chosen directly because it depends on the number of simulated trajectories from which the data points are sampled. Besides that, the initialization and the tolerance level of the BVP solver influence how much data results from each trajectory. Hence, the user modifies the amount of data used for the initial training set, validation and test set primarily by choosing the number of initial conditions which are typically sampled from a domain of the form $\mathbb{X}_0 = [x_{lb}, x_{ub}]^n$. We further note that the data generation is closely linked to the value function approximation model V^{NN} via the NN warm start initialization, as well as to the heuristical criteria applied during

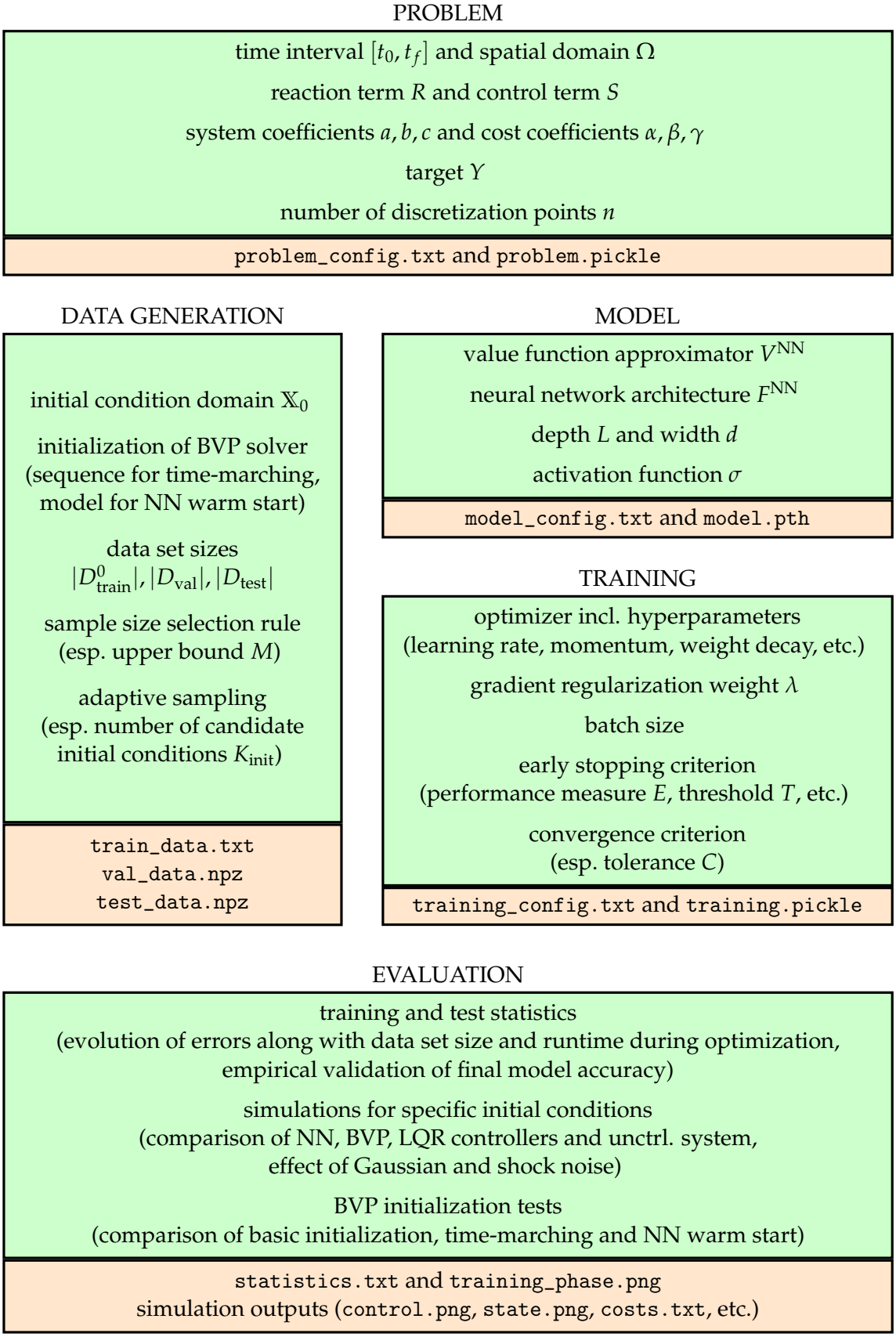

FIGURE 4.1: Structure of program with experiment settings and parameter choices determined by the user within the notebook (colored in green) and output files listed below containing configurations, class instances, data or evaluation results (colored in orange).

the training phase. Whenever new data points are computed, the corresponding `.npz` file is updated, ensuring that the data remains available and can potentially be reused.

The models contained in our program are based on feedforward or residual neural networks, F^{MLP} or F^{ResNet} in (3.4), of a given depth, a constant width across all layers and tanh, ReLU or softplus activation. By implementing (3.1), we are able to enforce the final condition of the value function and its gradient. In order to recover the best parameters after the training is terminated, the model is saved in the `.pth` file whenever the validation error reaches a new minimum. This makes it also possible to load the partially trained model for further training, in-the-loop simulations and empirical performance tests.

The model can be trained with a wide range of optimization methods. On the one hand, the program offers first-order stochastic optimizers like SGD and Adam operating on mini-batches, and on the other hand, it contains L-BFGS as a second-order deterministic optimizer requiring the full training set for each iteration. Further choices concern the employed regularization techniques and the criteria guiding the training and determining convergence. For validation based heuristics, the user can freely choose any linear combination of the error metrics defined in Section 3.2.4 as performance measure and one of the triggers (3.14), (3.15) or (3.16) for early stopping. Furthermore, the subset size for approximating the population variance and the cost gradient in the criteria (3.12) and (3.13) can be adjusted to trade off computational effort and accuracy adequately.

Throughout the training phase, careful monitoring is employed to keep the user informed about the ongoing optimization progress and data augmentation. Subsequently, an extensive evaluation of the deep learning based approach is conducted, complemented by a variety of plots and tables for visualization. In the following chapter a thorough interpretation is presented for the results obtained through multiple numerical experiments.

Chapter 5

Numerical Experiments

5.1 Experimental Set-up

In order to test our adaptive deep learning (DL) approach developed in Chapter 3, we perform multiple experiments for the optimal control problem (OCP) of reaction-diffusion systems presented in Chapter 4. Our numerical analysis is based on the implementation described in Section 4.6 and the provided code includes all the necessary components to reproduce the results of this chapter. In the following, we outline the experimental set-up by specifying the choices made with regard to the problem, the model and its training, as well as the data generation, all of which are summarized in Figure 4.1.

Certain problem configurations remain consistent throughout the experimental trials. First of all, the initial and final times are given by $t_0 = 0$ and $t_f = 5$, respectively, and the spatial domain is the interval $\Omega = [0, 1]$ which is discretized equidistantly. Our default set-up involves $n = 20$ internal mesh nodes, but we alter the discretization level later to investigate how the solution scales. The coefficients for the cost function are $\alpha = 1, \beta = 0.1$ and $\gamma = 2$ which makes the system rather cheap to control and puts an emphasis on the state reaching a specified target, such as

$$\begin{aligned} Y^{\text{zero}}(\omega) &= 0 \quad \text{or} \\ Y^{\text{quad}}(\omega) &= -(2\omega - 1)^2 + 1, \end{aligned}$$

at final time. In simulations, we consider two initial conditions, namely

$$\begin{aligned} X_0^{\sin}(\omega) &= \sin(2\pi\omega) \quad \text{and} \\ X_0^{\text{quadcos}}(\omega) &= -0.5\cos(4\pi(\omega - 0.5))\cos(2\pi(\omega - 0.5)) - 0.5. \end{aligned}$$

Besides that, we investigate various control and reaction terms leading to dynamical systems of increasing difficulty.

At the core of our model lies a residual network F^{ResNet} consisting of $L = 5$ hidden layers with a width of $d = 100$ neurons and softplus activation. This neural network (NN) is used within the proposed approximator V^{NN} which guarantees that the final condition of the value function is met. The model parameters θ are optimized with the goal of achieving a low control error which is closely associated with a precise approximation of the value function gradient V_x. Therefore, the training objective contains a large gradient regularization weight $\lambda = 100$ serving as a strong penalty mechanism. As optimization method, we adopt Adam with a learning rate of 10^{-3} and weight decay of 10^{-2} for handling batches of 100 data points. We remark that the training set is shuffled after each epoch, thus introducing stochasticity in the optimization process. Since we focus on an accurate control computation, we

use $E = \mathrm{RMAE}_u$ as performance measure and the quotient of generalization loss GL and training progress TP_ι over a strip of length $\iota = 5$ with threshold $T = 0.1$ as early stopping criterion. This technique helps to detect overfitting while also overcoming instabilities particularly during the early stages of training when the amount of available data is still limited. For the relatively simple problems considered first, we set the convergence tolerance to $C = 10^{-4}$ which will be adjusted to $C = 4 \times 10^{-4}$ in later experiments to ensure training completion within a reasonable time frame. Additionally, in some experimental series, we also terminate training before convergence after a certain time limit in order to compare model performance using the same computational resources.

For the data generation, we choose the domain $\mathbb{X}_0 = [-1.5, 1.5]^n$ from which we will sample initial conditions uniformly at random or adaptively where the predicted gradient norm $\left\| V_x^{\mathrm{NN}}(t_0, \cdot) \right\|_2$ is large. We start with a very small number of five initial conditions for the training set D^0_{train} and validation set D_{val} each in order to avoid solving the boundary value problem (BVP) multiple times without warm starting with a pre-trained NN model. For initial data generation, we initialize the solver using a straightforward approach if the problem is easy enough. However, when failed attempts occur more frequently, we switch to the time-marching method. When data is added adaptively during training, we repeatedly select the most suitable initial condition out of $K_{\mathrm{init}} = 10$ candidate points until enough data is generated. The test set D_{test} is created based on ten initial conditions after the model has been optimized. It should be noted that the number of data points along one trajectory depends on several factors such as the complexity of the problem, the initial guess and the desired tolerance of the BVP solver. The data set size selection rule is complemented by an upper bound of $M = 1.25$, meaning that a maximum of one quarter of the existing data can be added per training round.

An experimental run comprises several sources of randomness, such as uniform sampling from the initial condition domain and the neural network initialization, in addition to shuffling the training data before dividing it into mini-batches for the stochastic optimizer. For this reason, we base our experimental evaluation on multiple runs or select a representative run which reflects the typical behavior. To ensure reproducibility of the results, we set a seed to manage the sources of randomness. The only aspect that may differ is the computation time measured in wall time, encompassing CPU as well as input/output operations, which depends on the device. However, that does not impact the qualitative statements about the computational costs of the tested techniques.

5.2 Numerical Results and Interpretation

The experiments are conducted in various problem settings. Starting out from the linear-quadratic problem, we move forward by including non-linear states and finally bilinear controls. We also examine how well the neural network closed-loop method scales, how it reacts in the presence of noise, and how the heuristics guide the training. Furthermore, we empirically validate the accuracy of the NN model and compare its performance with the conventional linear-quadratic regulator (LQR) approach by simulating specific initial conditions.

5.2.1 Sensitivity to Noise in the Linear-Quadratic Problem

In our initial experiment, we focus on a linear-quadratic OCP as presented in Section 4.1.1. This allows us to compute the linear-quadratic regulator as a baseline to assess the performance of the neural network controller for certain initial conditions. Both closed-loop solutions, i. e., the ones provided by the NN and the LQR, are compared to the open-loop BVP solution in simulations under the presence of noise.

To specify the experimental setting, we choose a diffusion coefficient of $a = 0.02$. Furthermore, we use the additive control $S = S_1$ where $\Omega_S = (0.25, 0.5)$ is the support of the indicator function and $c = 1$ is the corresponding coefficient. For now, there is no reaction term, i. e., $R \equiv 0$. We set the quadratic function Y^{quad} as target and in the simulations, we examine both initial conditions X_0^{sin} and X_0^{quadcos}. Since we do not encounter any failed attempts solving the BVP directly over the whole time interval $[t_0, t_f]$ with a basic initialization, we do not employ the time-marching technique yet.

To perturb the system, we introduce a noise term into the right-hand side of the ordinary differential equation (ODE). Specifically, we consider two types of noise: On the one hand, we add Gaussian noise with some mean μ and variance σ^2. We remark that the diffusion process tends to smooth out small variance in the system dynamics, so that only biased Gaussian noise creates a significant offset which needs to be taken into account by the controller. In practical applications, the mean could correspond to systematic errors and the variance to random errors caused by inaccurate measurement or observation. On the other hand, we define a shock noise of intensity A_W that occurs during a short time interval $[t_{\text{start}}, t_{\text{end}}]$ by

$$W(t) = \begin{cases} A_W(t - t_{\text{start}})^2(t - t_{\text{end}})^2, & \text{if } t_{\text{start}} \leq t \leq t_{\text{end}} \\ 0, & \text{else} \end{cases} .$$

This type of noise might occur due to a sudden external influence which is not captured by the modelled dynamics. In the following experiments, we combine both stochastic and deterministic sources of noise given by the Gaussian noise with $\mu = 0.25$ and $\sigma^2 = 4$, and the shock noise with $t_{\text{start}} = 2$, $t_{\text{end}} = 3$ and $A_W = 25$, respectively.

In the simulation, we find that both the NN and LQR controllers are able to compensate for the noise in such a way that the state is brought close to the desired quadratic target, particularly towards the end of the time interval $[t_0, t_f]$. When applying the open-loop control obtained from the BVP solution, however, the state increasingly deviates from the target as time progresses, especially after the shock appears at t_{start}. This is illustrated in Figure 5.1 showing the state trajectories of all three methods. We only display the results for X_0^{sin}, as the behavior is very similar for both tested initial conditions.

The respective controls plotted in Figure 5.2 help explain why the closed-loop approaches perform so much better. Although the open-loop BVP control is optimal for the undisturbed system, it is unable to take the noise into account. As a result, the control effort is initially large to steer the state from its initial condition to the target and then remains relatively constant in order to stabilize the state around Y^{quad} if no noise were present. In contrast, the closed-loop controllers resulting from the NN and the LQR approach react to the disturbances in the dynamical system in order to continually correct the state trajectory. Throughout the entire time interval, they adapt to the bias μ of the Gaussian noise and therefore drift apart from the open-loop

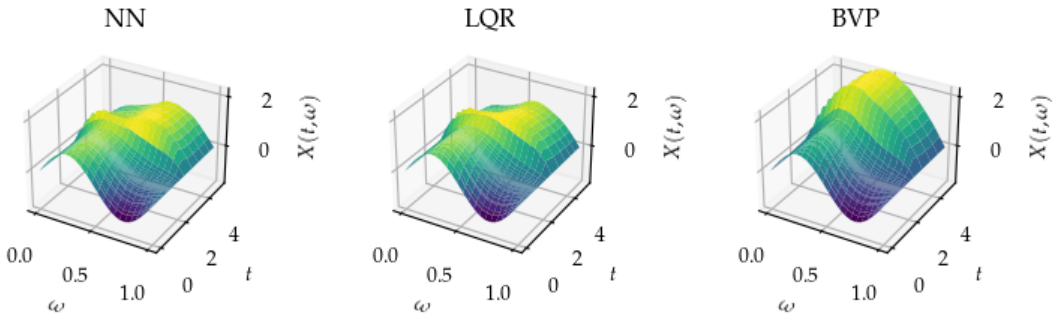

FIGURE 5.1: Simulated state trajectories for the linear-quadratic problem starting from the initial condition $X_0^{\sin}$ and aiming to reach the target Y^{quad} under the presence of Gaussian and shock noise.

control. The larger the mean of the Gaussian distribution, the bigger becomes the gap between open- and closed-loop solutions. Since the closed-loop methods also respond to the variance σ^2, their control graphs show small fluctuations. However, the shock noise has the greatest impact on the closed-loop controls, causing a drastic peak whose amplitude depends on the factor A_W. This control intervention ensures that the state maintains the desired parabolic shape despite the unexpected disturbance. We further note that the NN controller responds more rapidly and strongly compared to the one from the LQR.

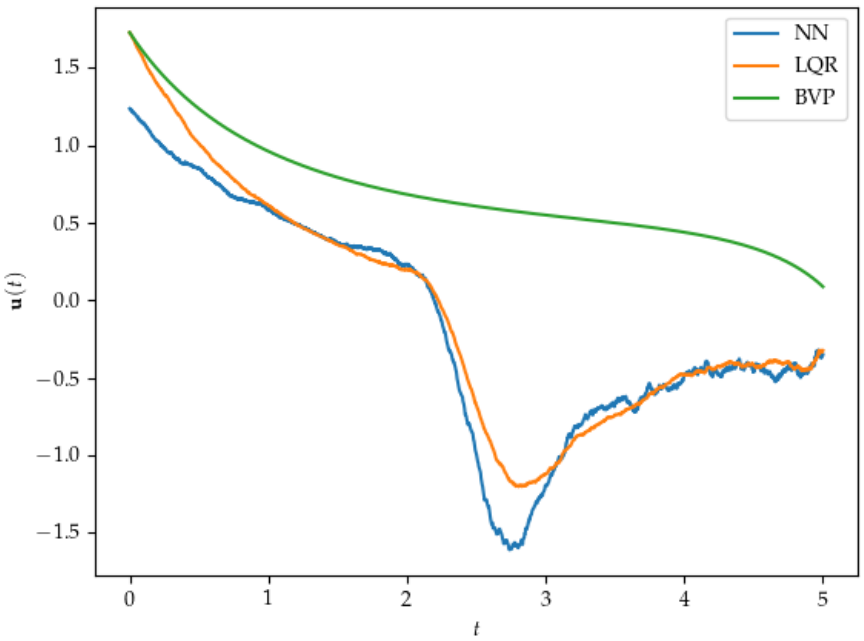

FIGURE 5.2: Simulated controls for the linear-quadratic problem with initial condition $X_0^{\sin}$ under the presence of Gaussian and shock noise.

These observations are also reflected in the costs. In the following, we do not only analyze the total cost

$$J(\mathbf{u}, \mathbf{x}) = \psi(\mathbf{u}(t), \mathbf{x}(t)) + \phi(\mathbf{x}(t_f))$$

but also its single components, namely the running state and control cost given by

$$\psi_1(\mathbf{x}(t)) := \alpha h \int_{t_0}^{t_f} \|\mathbf{x}(t) - y\|_2^2 \, \mathrm{d}t \quad \text{and} \quad \psi_2(\mathbf{u}(t)) := \beta \int_{t_0}^{t_f} |\mathbf{u}(t)|^2 \, \mathrm{d}t,$$

respectively, adding up to $\psi := \psi_1 + \psi_2$, as well as the final cost

$$\phi(\mathbf{x}(t_f)) := \gamma h \left\|\mathbf{x}(t_f) - y\right\|_2^2.$$

First, we examine the results in Table 5.1. We find that for both initial conditions, the total costs J associated with the NN and LQR solutions are three to four times lower than those of the BVP solution. This once again highlights the shortcomings of open-loop methods in noisy real-world scenarios. The significant difference in cost can be attributed to the high state costs incurring during the time interval and at final time, as the state is perturbed and not corrected accordingly. In Figure 5.3, we observe the behavior of the running cost components for the case of $X_0^{\sin}$ in even more detail. At the beginning of the time interval, the gap between the running cost ψ of the open- and closed-loop solutions increases gradually due to the influence of the biased Gaussian noise on both summands ψ_1 and ψ_2. However, the shock noise has a much larger effect on the cost development: Shortly after t_{start}, the running state cost of the BVP solution and, consequently, its total running cost jump to and stay at a high level since open-loop approaches are not able to correct the state trajectory on the fly.

It is noteworthy that the total costs resulting from the NN controller are slightly smaller than those from the LQR controller, which is somewhat unexpected. However, it should be kept in mind that the optimality of the LQR controller relies on the absence of noise in the remaining time interval. As we lack prior knowledge of future disturbances, we cannot do better than the LQR controller in this scenario. However, it is possible that the NN controller finds a better solution by chance. A closer examination of the cost components reveals that the LQR method suffers mainly from a higher running state cost ψ_1 while the control cost ψ_2 is the same or even smaller than that of the NN approach. Figure 5.3 sheds light on this phenomenon: Prior to the shock, the cost development of both closed-loop methods is similar, but after t_{start}, the NN control cost peaks higher. This enlarged control effort for regulating the shock noise pays off since it subsequently leads to lower state costs. Although the gap between the NN's and LQR's cost components narrow again as we approach t_f, the NN solution outperforms the LQR after integrating the running cost over the whole interval $[t_0, t_f]$.

initial condition	solution technique	running state cost	running control cost	final state cost	total cost
$X_0^{\sin}$	NN	1.15	0.28	0.13	1.56
	LQR	1.24	0.28	0.13	1.65
	BVP	3.72	0.30	2.19	6.21
X_0^{quadcos}	NN	1.21	0.73	0.10	2.04
	LQR	1.28	0.65	0.12	2.05
	BVP	3.75	0.77	2.26	6.78

TABLE 5.1: Costs with their components resulting from simulations for the linear-quadratic problem under the presence of Gaussian and shock noise.

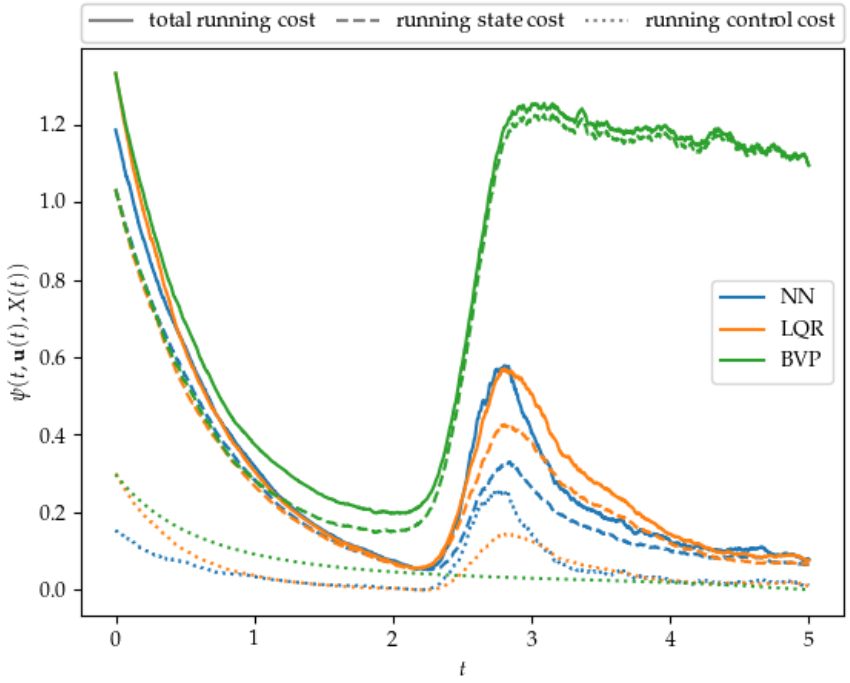

FIGURE 5.3: Simulated running costs with state and control components for the linear-quadratic problem with initial condition X_0^{sin} under the presence of Gaussian and shock noise.

This experiment demonstrates that the DL-based approach of computing optimal feedback controls performs well, even in the presence of various sources of noise. This makes the NN controller attractive for real-world applications in which open-loop solutions cannot be used appropriately. Moreover, the NN method may be applied to problems beyond the linear-quadratic realm where an analytical closed-loop LQR solution is no longer available. Such problem settings will be explored in the following numerical tests.

5.2.2 Handling Non-linearities in the System Dynamics

In the next experiment, we investigate the effects of adding a non-linear reaction term to the dynamical system as described in Section 4.1.2. More specifically, we assess the performance of the NN controller in comparison to the one obtained by applying the LQR to the linearized system. Since we run this simulation without any sources of noise, the open-loop BVP solution provides an accurate solution which acts as a reference point for evaluating the two closed-loop approaches. Besides that, we simulate the evolution of the state for the uncontrolled system, i. e., for $\mathbf{u} \equiv 0$.

To introduce non-linearity into the system, we use either the hyperbolic or the logistic growth rates as reaction term, that is $R = R_1$ or R_2, and set the corresponding coefficient to $c = 1$. The diffusion coefficient is increased to $a = 0.09$ in order to avoid an all too rapid explosion of the state caused by the growth terms. The control term S and its coefficient c remain the same as in the previous experiment. Furthermore, we reuse the quadratic target Y^{quad} and both initial conditions X_0^{sin} and X_0^{quadcos}.

For this problem setting, we observe that the BVP solver sometimes fails for uniformly at random drawn initial conditions if it is initialized straightforwardly. More precisely, the convergence rate is 81 % in case of hyperbolic growth and even 91 %

for logistic growth. By employing time-marching or warm start with even a barely trained NN model, convergence is always achieved for both reaction terms. For initial data generation, we nevertheless use a simple initialization instead of the rather slow time-marching technique, and replace the few initial conditions for which the BVP solver diverges.

Generally, we observe that the NN solution leads to a marginally higher total cost J compared to the BVP solution, and hence, outperforms the linearization based LQR approach, as presented in Table 5.2.

reaction term	initial condition	NN	LQR	BVP	unctrl. system
R_1	$X_0^{\sin}$	0.734	6.440	0.730	3.687
	X_0^{quadcos}	1.553	2.029	1.546	4.509
R_2	$X_0^{\sin}$	1.440	1.442	1.436	8.899
	X_0^{quadcos}	2.454	2.474	2.452	∞

TABLE 5.2: Total Costs resulting from simulations of the non-linear system with different reaction terms.

This becomes especially apparent in the simulation with the hyperbolic growth rate R_1, in which the LQR method yields an even worse result than the uncontrolled system when starting from the initial condition $X_0^{\sin}$. Considering the control and state trajectories depicted on the left of Figure 5.4 and in the top row of Figure 5.5, we find that the NN and BVP solutions strongly resemble each other, whereas the LQR yields a significantly different shape for both graphs. Since the non-linear term R_1 vanishes through linear approximation at the origin, the LQR controller fails to guide the state to the quadratic target. In fact, it can barely prevent the state from exploding by exerting a huge control effort when approaching the final time t_f. On the contrary, the state of the uncontrolled system converges to zero as the diffusion prevails over the hyperbolic growth rate whenever $R_1(x_0)$ is small enough.

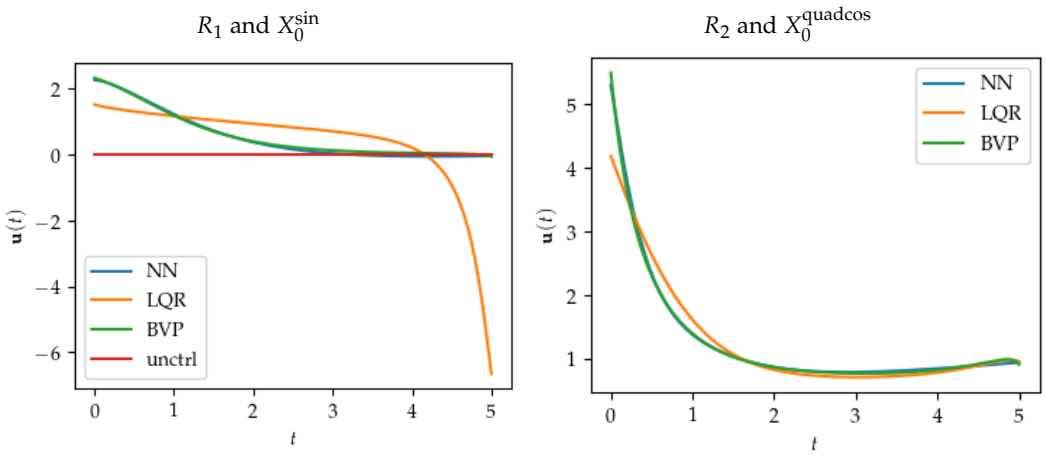

FIGURE 5.4: Simulated controls for the non-linear system with hyperbolic growth rate R_1 and initial condition $X_0^{\sin}$ (left) or logistic growth rate R_2 and initial condition X_0^{quadcos} (right).

For the logistic growth rate R_2, however, the LQR controller achieves a cost that is only slightly larger than the one of the NN controller. Moreover, it avoids a blow-up as observed for the uncontrolled dynamics starting at X_0^{quadcos}. This can be seen in

Table 5.2 but is also reflected in the control and state plots shown on the right of Figure 5.4 and in the bottom row of Figure 5.5 which are remarkably similar compared to the baseline given by the BVP solution. This indicates that in this specific case, the LQR method is able to provide a reasonable solution despite the linearization of the logistic growth term.

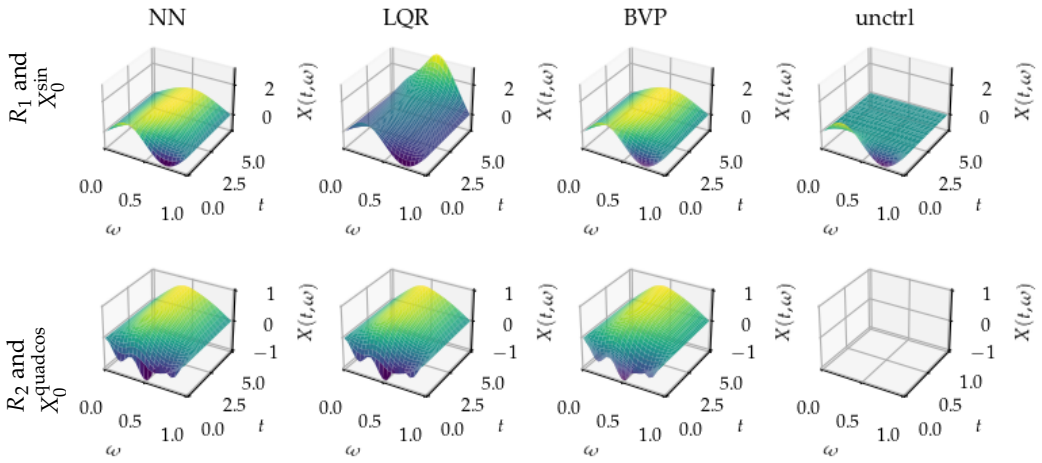

FIGURE 5.5: Simulated state trajectories for the non-linear system with hyperbolic growth rate R_1 starting from the initial condition $X_0^{\sin}$ (top row) or logistic growth rate R_2 starting from the initial condition X_0^{quadcos} (bottom row), aiming to reach the target Y^{quad}.

To sum up, the NN method reliably produces accurate solutions to problems with non-linear state dynamics and can handle cases that would otherwise lead to blow-ups if uncontrolled. The LQR controller can be effective when the non-linearity is not too strong or can be well-approximated by a linearization, but it still results in higher costs and may even fail in some cases. Hence, the NN approach is clearly the preferred tool when choosing an closed-loop solution method for non-linear systems.

5.2.3 Heuristic-Guided Training for Bilinear Control

In order to make the problem even more challenging, we replace the additive with a bilinear control as in Section 4.1.3. Our first objective is to assess the effectiveness of the heuristics used for the progressive data generation during training, including the early stopping criterion, the data set size selection rule, as well as the adaptive sampling of initial conditions. Besides that, we aim to examine the influence of physics-informed learning via gradient regularization on the optimization process. To achieve all of this, we monitor the training phase by tracking the relative errors, the amount of data and the runtime over the course of the epochs.

In addition to the diffusion term with coefficient $a = 0.02$, we have the hyperbolic growth $R = R_1$ as non-linear reaction term with $b = 1$, along with the bilinear control term $S = S_2$ with $c = 1$. For this experiment, we aim to drive the state to zero. We note that the system is stable for Y^{zero} since both the diffusion and the hyperbolic reaction term vanish for this target.

As the problem difficulty increases, the BVP solver gets more sensitive to initial guesses. As a result, a basic initialization leads to convergence for only 58 % of the initial conditions, while more sophisticated techniques like time-marching and NN warm start still achieve a 100 % convergence rate. Hence, we use time-marching as initialization whenever a NN model is not yet available for warm starting the BVP solver.

Figure 5.6 illustrates the training phase of a representative run displaying all error metrics introduced in Section 3.2.4. These are the relative mean absolute error w. r. t. the value function and the control, denoted by RMAE_V and RMAE_u, respectively, as well as the relative mean L^1 error w. r. t. the value function gradient, that is $\mathrm{RML}^1{}_p$, which are applied to both the training and the validation set. This yields three training and validation errors each, whose evolution throughout the optimizer's epochs are shown in separate plots.

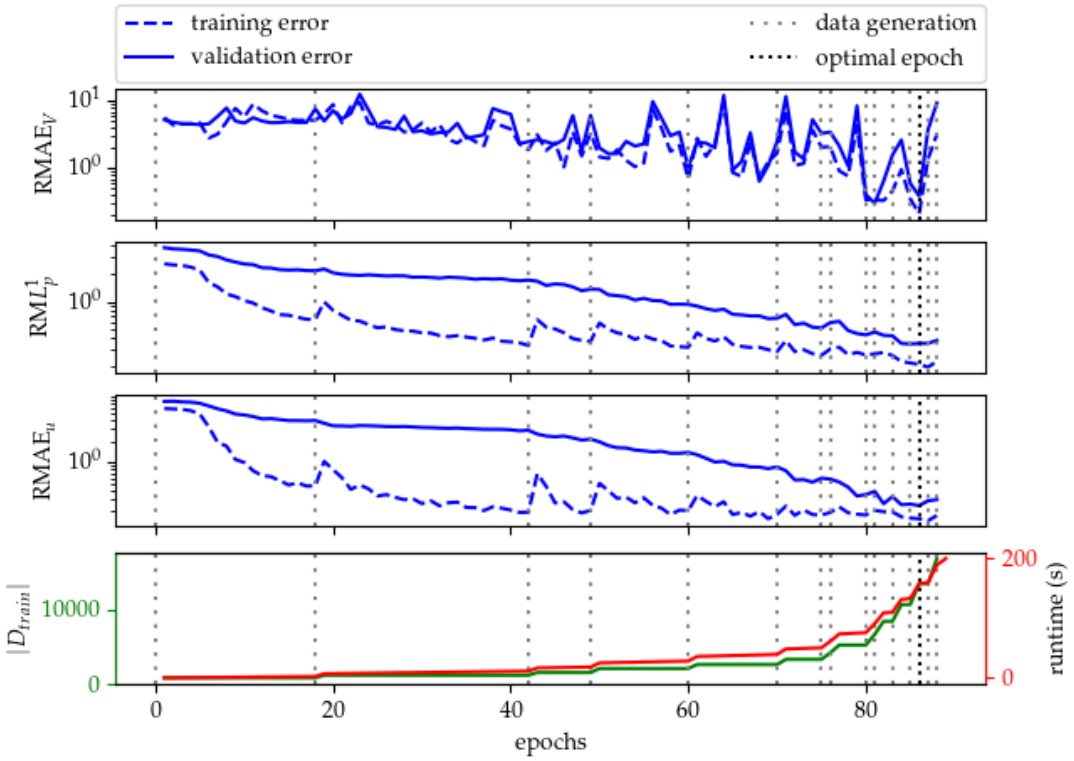

FIGURE 5.6: Training phase for the bilinear control problem: Training and validation error measured by metrics concerning (from top to bottom) the value function, its gradient and the control in the three upper plots. Number of training data points (left axis) and training time (right axis) plotted below. Sparsely dotted lines indicate the expansion of the training data set corresponding to the start of a new training round. Epoch at which best model parameters are achieved is marked by densely dotted line.

It is typical for the training errors to be consistently lower than the respective validation errors, except during early stages of training where the model is still not well fitted to the data. We remark that the gap between the two corresponding errors serves as an estimate of the generalization error. Using an extremely large gradient regularization weight of $\lambda = 100$ encourages the model to learn the gradient V_x over the value function itself. Thus, we observe that the RMAE_V fluctuates strongly, especially at later epochs when the model is being fine-tuned. Conversely, the $\mathrm{RML}^1{}_p$

and RMAE_u exhibit a more steady and smoother decrease over time.

It should be noted that the validation set D_{val} remains unchanged during the entire training phase, whereas the training set D_{train}^j is dynamically augmented. The effects of this data expansion are clearly visible by the spikes in the training error curves of the gradient and control metrics. Each time new data is generated, a new training round j begins indicated by a vertical dotted line. The current training set size is shown as a step function in the bottom plot.

Starting from random initialized network weights and biases, the model can make significant progress with only a small amount of data. However, as we continue optimization, more data is necessary in order to reach a higher model accuracy without suffering from generalization loss. The early stopping technique which is used to trigger data generation seems to capture this approach effectively by initially delaying the expansion of the training set and then, when the model needs to be refined, interrupting the optimizer more frequently for adding more data. Towards the end of the training phase, data generation occurs at almost every epoch so that the number of data points grows nearly exponentially.

To gain insight into why the heuristics lead to data generation at this specific rate, we analyze the training error E_{train} and validation error E_{val} based on the performance measure $E = \mathrm{RMAE}_u$ more closely. When E_{val} stagnates and starts to increase slightly while E_{train} remains mostly unchanged during the last $\iota = 5$ epochs, we obtain a positive generalization loss $\mathrm{GL} > 0$ and a very small training progress TP_ι. Hence, the quotient of these two values exceeds the threshold $T = 0.1$ so that the early stopping criterion (3.15) is met. This leads to either termination of the training if the convergence criterion (3.12) is satisfied, or to adaptive data generation according to the data set size selection rule (3.13). In the latter case, the model is confronted with new data points sampled from rather complicated regions, causing E_{train} to jump up abruptly. After updating the model parameters, it quickly drops down again, forming the characteristic spikes in this error curve. At the same time, E_{val} begins to decrease slightly indicating that augmenting the training set also helps to improve the model performance on unseen data. In summary, whenever the generalization error, measured by the difference between the training and validation error, becomes larger due to overfitting, it is reduced again by providing more data. Since the control error metric RMAE_u is closely tied to the one of the value function gradient via the feedback law (3.2), we observe the same behavior for $\mathrm{RM}{L^1}_p$. When the convergence criterion is finally met, we select the model parameters at the epoch where the performance measure E evaluated on D_{val} is lowest, marked by the black dotted line. That is typically not the last epoch since the training is not interrupted until the generalization loss starts to increase again.

Finally, we examine the computation time spent during the training process presented alongside the development of the data. It is evident that the runtime increases parallel to the number of data points. The reason for this is that the computational costs mainly arise from the variance estimate calculation needed for the data set size selection as well as from solving the BVPs for generating new data. To ensure that the training time does not become infeasible, several factors need to be considered: First of all, the mentioned computational load should only be incurred when most necessary which is managed well by the early stopping method. Additionally, the generation of an excessive amount of new data should be prevented. This is achieved by setting an upper bound of $M = 1.25$, which is indeed often attained. Most importantly, the training has to be stopped as soon as a larger data set does not promise any significant improvement of the model accuracy, as determined by the

tolerance value $C = 10^{-4}$. Otherwise the computation time would increase unreasonably as the data set size grows very fast at late stages of training. We also note that reducing the weight λ or not using gradient regularization altogether leads to an excessive amount of training data and time. Since valuable gradient information is excluded in the learning process, we obtain less effective optimization steps and, thus, slower convergence.

Overall, the combination of the heuristics appears to guide the training process effectively. In order to evaluate them in more detail, we would need to consider the specific criteria for adaptive and progressive data generation separately. This requires to perform a series of experiments in which certain elements of the heuristical tests are disabled or modified to assess their effect on the overall training and final model accuracy. Since this further numerical analysis goes beyond the scope of this thesis, we only provide the following considerations for future research.

Progressive training set augmentation via sample size selection: To start with, one could draw a comparison between progressively adding data throughout the training phase and optimizing with a fixed data set. The latter approach eliminates the computational costs for evaluating heuristical tests but comes with several limitations. Firstly, we cannot take advantage of the NN warm start technique for initializing the BVP solver. Instead, we have to rely on the more expensive time-marching method since all data is generated before a pre-trained model is available. Secondly, an adaptive selection of potentially more relevant data is impossible. Moreover, progressive data generation alleviates the problem of deciding how many data points are necessary for a desired generalization performance. In case of a fixed training set, we need to determine the amount of data beforehand, which makes it even more complicated to trade off accuracy and costs appropriately. Furthermore, the sample size selection method provides a convergence criterion that is absent in the fixed training set approach, raising the question of when to stop training.

Delayed data generation through early stopping: In order to assess the benefits of delaying data generation by the early stopping method, one could alternatively evaluate the test for data set size selection after each epoch. This either interrupts the training for adaptive data generation or terminates the training phase altogether. Although this simplified approach has been employed and numerically tested in [49] for a deterministic second-order optimization, a direct comparison to using early stopping is lacking. Preliminary trials indicate that this approach tends to generate more data and results in fewer training epochs, but at the cost of higher errors and occasional failures in simulations. By postponing the data set size selection until the current training set no longer yields reliable optimization steps, we not only reduce the computational burden of variance computation but also ensure adequate pre-training of the NN model.

Adaptive sampling for targeted data generation: The impact of adaptive data sampling can be explored by replacing the selection of top candidates where value function gradients are large with drawing initial conditions uniformly at random. Although adaptive selection introduces some additional computational effort, including a pass through the NN model and gradient evaluation, it promises more effective learning. This approach is likely to result in fewer iterations, requiring less data and potentially yielding lower test errors. We note that training errors might in

fact be larger due to the selection of particularly challenging data, but the generalization accuracy, as measured by test errors, is expected to improve. The effects of adaptive sampling are assumed to be especially significant in challenging problems with a steep and complicated value function. One might also develop other selection criteria specifically tailored to the application, for instance, to achieve a high level of accuracy in some targeted region.

5.2.4 Empirical Evaluation of Scalability across System Dimensions

In the context of the bilinear control problem, we also test the scalability of the DL-based approach with respect to the system dimension. While our previous experiments were conducted using a default discretization with $n = 20$ internal nodes, we now vary the dimension of the state space by adjusting the mesh (4.11) to be coarser or finer, yielding $n = 5$ or 40, respectively. We are particularly interested in the performance for high dimensional problems since these are known to be hard to solve due to the curse of dimensionality.

In the first series of experiments, we focus on the same problem set-up as before, which involves the dynamical system with hyperbolic growth R_1 and bilinear control S_2 steering the state towards the target Y^{zero}. In order to draw a comparison under similar computational resources, we limit the training phase to 100 seconds for each considered dimension n. This fixed time frame includes the time spent on testing convergence and generating additional data on the fly, but excludes the time for generating the initial training set D^0_{train}. The training configurations including the heuristical tests remain otherwise unchanged. The performance of the trained NN model is then measured empirically on an independent test set D_{test}. Each experiment is repeated 10 times with different seed values, and the mean over all runs is displayed in Table 5.3.

As n increases, the amount of data generated during the fixed training time shrinks. Since the problem becomes more difficult in higher dimensions, the training process advances more slowly and overfitting, which triggers data generation via early stopping, tends to occur less frequently. As a result, the number of data points in the final training round j, that is $|D^j_{\text{train}}|$, is lower. Moreover, the total number of trajectories from which the data points are sampled decreases even more drastically because the BVP solver produces more data points per trajectory in order to maintain the same level of accuracy for larger problems. Thus, for the coarse mesh with $n = 5$, less than half of the data points are sampled along one trajectory on average, whereas for the fine discretization with $n = 40$, more than double are sampled, compared to the standard case of $n = 20$.

Since the stochastic optimizer Adam iterates through the training set in batches of constant size, a smaller training set leads to fewer optimization steps per epoch. As we discussed earlier, a larger state dimension n results in a comparatively smaller training set, which means that each epoch is completed faster. Hence, when limited by time, the training phase for higher dimensional problems contains more epochs.

In addition to the training statistics, Table 5.3 also presents the test errors for empirically verifying the accuracy of the NN predictions concerning the value function, the costate, and the control. Due to the heavy regularization employed to approximate the gradient correctly, the $\text{RML}^1{}_p$ and RMAE_u are low compared to the RMAE_V across all problem dimensions. This observation is consistent with the results of the previous section regarding the training and validation errors during optimization. Furthermore, the errors for the costate and control metrics grow as the dimension is increased, especially between $n = 20$ and 40. Nevertheless, we find that the NN

method can still handle large problems reasonably well, despite a shortened training phase of 100 seconds.

n	# trajectories	# data points	# data points per trajectory	# epochs	RMAE_V	$\mathrm{RML}^1{}_p$	RMAE_u
5	140	11306	81	41	2.29	0.34	0.24
20	38	6626	174	60	1.43	0.39	0.27
40	14	4895	350	90	3.60	1.29	2.39

TABLE 5.3: Training statistics and test errors after a fixed training time of 100 seconds, averaged over 10 runs, resulting from bilinear control problems of different dimensionality with target Y^{zero}.

For the second experiment series, the target is modified to the quadratic function Y^{quad}, which is harder to attain as it is unstable when left uncontrolled. Furthermore, the training is not terminated after a certain time limit but continues until the convergence criterion is satisfied. However, we increase the convergence tolerance to $C = 4 \times 10^{-4}$ so that the the computation time remains feasible. In addition to the fully trained model, denoted as $\mathrm{NN}^{\mathrm{full}}$, we also store a partially trained model, $\mathrm{NN}^{\mathrm{partial}}$, obtained after five seconds. This allows us to explore the impact of the problem dimension n on the performance of the final NN model, as well as of models at early stages of training. This becomes particularly relevant for warm starting the BVP solver as investigated in the following section. Table 5.4 presents the results of the empirical evaluation, along with the training time of the respective model.

Indeed, we are able to obtain high-fidelity models when training is completed. Moreover, the $\mathrm{NN}^{\mathrm{full}}$ models achieve similar relative errors on the test sets across all dimensions. Unsurprisingly, the training time required for convergence increases as n is enlarged, with the strongest increase observed from $n = 20$ to 40. This is mainly because generating data becomes more expensive and a larger number of iterations are needed. Nonetheless, we find that the computational effort scales reasonably with the problem dimension.

As anticipated, the partially trained models consistently exhibit higher test errors than the corresponding fully trained models. Despite expending merely negligible computational costs, the $\mathrm{NN}^{\mathrm{partial}}$ models still capture the main features of the value function, yielding useful low-fidelity models.

n	model	training time	RMAE_V	$\mathrm{RML}^1{}_p$	RMAE_u
5	$\mathrm{NN}^{\mathrm{partial}}$	5 s	1.05	0.55	2.14
	$\mathrm{NN}^{\mathrm{full}}$	113 s	0.54	0.42	0.93
20	$\mathrm{NN}^{\mathrm{partial}}$	5 s	2.00	0.73	2.32
	$\mathrm{NN}^{\mathrm{full}}$	159 s	0.45	0.31	0.87
40	$\mathrm{NN}^{\mathrm{partial}}$	5 s	0.49	0.46	1.69
	$\mathrm{NN}^{\mathrm{full}}$	502 s	0.43	0.34	1.17

TABLE 5.4: Training time and test errors of partially and fully trained models, used for testing NN warm start, resulting from bilinear control problems of different dimensionality with target Y^{quad}.

In this section, we have demonstrated the viability of the proposed DL method for computing closed-loop solutions for high dimensional OCPs. We have shown that, given a limited runtime, the training becomes less data-intensive and the accuracy of the resulting NN model drops as the state dimension grows. If we aim to

maintain a high level of accuracy, we need to invest more computational resources for training, but this still remains feasible. Besides that, it is possible to obtain rough predictions by models trained for just seconds, providing an initialization for solving BVPs, as we will explore in the following experiment.

5.2.5 Impact of BVP Solver Initialization on Data Generation Efficiency

Investigating different initialization techniques for the BVP solver, we aim to identify the most efficient way for data generation before, during and after training. Specifically, we are interested in the potential advantages of using time-marching and NN warm start over a simple initialization, as proposed in Section 3.3.2. This analysis involves finding a suitable time sequence for time-marching, as well as assessing the extent to which the effectiveness of warm starting depends on the model accuracy.

We revisit the set-up with the hyperbolic growth rate R_1 and the bilinear control S_2 as part of the state dynamics, and the target Y^{quad} in the problem's objective. Following our previous experiment on scalability, we alter the state dimension n in order to adjust the difficulty of the boundary value problem. For each considered dimension, we sample 100 initial conditions uniformly at random from $\mathbb{X}_0$. For a fair comparison, we then use the same set of initial conditions to test all initialization methods.

The simplest approach of initializing the BVP solver is to set the state values to the given initial condition x_0 and the costate and the value function to zero for all time steps. The boundary value problem is then solved directly over the entire time span $[t_0, t_f]$. We use this basic initialization as a baseline for comparison with more sophisticated strategies.

Firstly, we consider time-marching which is based on intermediate times $T_J = \{t_j\}_{j=0}^{J}$ to extend the solution sequentially to time intervals $[t_0, t_j]$ of increasing length. One option to implement this is by using an equidistant time sequence, that is

$$T_J^{\text{equ}} = \left\{ t_0 + \frac{j}{J}(t_f - t_0) | j \in \{0, \dots, J\} \right\}.$$

We start by comparing the results obtained using $T_{J=14}^{\text{equ}}$ with another sequence of twice as many equally spaced time points, i. e., $T_{J=29}^{\text{equ}}$. Alternatively, one can adapt the interval lengths to further improve the speed and convergence rate. Using the same number of time steps as in our first equidistant sequence, we experiment with initially shorter intervals given by

$$T_{J=14}^{\text{ada}} = \left\{ t_0 + h(t_f - t_0) | h \in \{0, 10^{-3}, 10^{-2}, 0.1, 0.125, 0.15, 0.2, 0.3, 0.4, \dots, 0.9, 1\} \right\}.$$

All these sequences are used in combination with an adaptive BVP solver tolerance. This means that we begin solving on $[t_0, t_1]$ with a tolerance of 0.1, and then halve the tolerance for each subsequent interval until we reach the desired tolerance of 10^{-5}. This trick helps to speed up the computation without compromising on the final accuracy.

Secondly, we explore the idea of using neural network simulations to provide initial guesses for the BVP solver. Since NN warm starting may be used throughout the training phase as well as after completion, we base our investigation on models available at different stages of training. Specifically, we employ the $\text{NN}^{\text{partial}}$ model,

optimized partially for five seconds, and the NN^{full} model, satisfying the convergence tolerance $C = 4 \times 10^{-4}$, as shown in Table 5.4. They represent a low- and high-fidelity model, respectively, enabling us to test NN warm starting for varying model accuracy.

According to Table 5.5, the convergence rate for each initialization technique generally decreases for higher state dimensions n. Hence, the time spent on failed attempts goes up, and the computations for the successfully solved BVPs also take longer. Besides that, the number of generated data points tends to grow as n increases. An increase of time may be offset if the computation yields a significantly larger amount of data points. Therefore, we also calculate the solution time per data point, as well as the total time, which is the sum of fail and solution time, per data point. However, even these ratios increase with n across all tested initializations. These observations follow from the fact that the BVP becomes more challenging for larger problems.

To begin with, we analyze the time-marching approach in order to determine the most appropriate sequence T_J. While time-marching seems to yield high convergence rates for moderately large problems independently of the applied time sequence, we notice a severe drop in performance for the equidistant sequence with $J = 14$ when solving the high dimensional problem with $n = 40$. With the overall lowest convergence rate of only 11 %, it is even outperformed by the basic initialization. We thus focus on the other two considered time sequences, namely $T^{\text{equ}}_{J=29}$ and $T^{\text{ada}}_{J=14}$, since they lead to reliable convergence across all problem dimensions. It appears that starting with smaller time intervals facilitates convergence for complicated problems. Furthermore, we find that the refined sequence with $J = 29$ produces more data points than the others with $J = 14$. However, doubling the amount of intermediate time steps is significantly more time-costly, even when dividing the computation time by the number of generated data points. We conclude that the sequence with adaptive interval lengths $T^{\text{ada}}_{J=14}$ yields the best trade-off in terms of convergence rate and speed and will therefore be used as a reference point for further analysis.

We proceed to examine the practicability of NN warm start using either partially or fully trained models. Due to the larger test errors, we expect the barely optimized models to be less effective in initializing the BVP solver. However, the $\text{NN}^{\text{partial}}$ model achieves remarkably high convergence rates even for rather difficult problems. Consequently, employing NN warm start is a viable option even at early stages of training. Using the final model NN^{full} improves the convergence rate further by five percentage points in the case of $n = 20$ and 40.

After considering the initialization techniques separately, we now compare them with each other regarding their reliability and speed of BVP convergence. For low and intermediate problem dimensions, the basic initialization technique results in the most failures with only 62 % convergent BVP solutions, whereas all other methods yield above 90 % convergence. For $n = 40$, the success rate of the basic initialization drops even lower to roughly half of the attempts. Hence, the application of time-marching with a properly tuned sequence or NN warm start gains in importance as the problem is enlarged. Both techniques yield similar convergence results, with warm starting using $\text{NN}^{\text{partial}}$ exhibiting a slightly lower rate and NN^{full} having a marginally higher rate than time-marching with the selected sequence $T^{\text{ada}}_{J=14}$.

Although the basic initialization has the smallest solution time, including when measured per data point, the time spent on failed attempts is immensely high, especially for larger problems. As mentioned before, the computation time generally increases with the problem dimension n. However, the speed of time-marching scales

particularly poorly with n. This highlights the benefits of utilizing NNs to aid in data generation. While NN warm start achieves approximately the same solution time per data point as time-marching with $J = 14$ time steps for $n = 5$, it is about three times faster for $n = 20$ and four times faster for $n = 40$. When n is large, NN warm start beats time-marching with respect to all recorded times, even when using a low-fidelity model. Therefore, using NN simulations for initialization is crucial to save costs for computing high dimensional BVP solutions.

n	init. technique	conv. rate	# data points	fail time	solution time	solution time per data point	total time per data point
5	basic	62 %	18897	50 s	6 s	3.0×10^{-4} s	2.9×10^{-3} s
	$T^{\text{equ}}_{J=14}$	97 %	10329	6 s	20 s	2.0×10^{-3} s	2.6×10^{-3} s
	$T^{\text{equ}}_{J=29}$	98 %	12486	7 s	48 s	3.8×10^{-3} s	4.4×10^{-3} s
	$T^{\text{ada}}_{J=14}$	98 %	9729	3 s	20 s	2.1×10^{-3} s	2.4×10^{-3} s
	$\text{NN}^{\text{partial}}$	93 %	7833	15 s	17 s	2.1×10^{-3} s	4.1×10^{-3} s
	NN^{full}	91 %	7543	22 s	14 s	1.9×10^{-3} s	4.8×10^{-3} s
20	basic	62 %	17812	335 s	56 s	3.2×10^{-3} s	2.2×10^{-2} s
	$T^{\text{equ}}_{J=14}$	96 %	19812	94 s	262 s	1.3×10^{-2} s	1.8×10^{-2} s
	$T^{\text{equ}}_{J=29}$	96 %	26529	45 s	979 s	3.7×10^{-2} s	3.9×10^{-2} s
	$T^{\text{ada}}_{J=14}$	96 %	20415	45 s	265 s	1.3×10^{-2} s	1.5×10^{-2} s
	$\text{NN}^{\text{partial}}$	91 %	19011	174 s	83 s	4.4×10^{-3} s	1.4×10^{-2} s
	NN^{full}	96 %	18388	73 s	82 s	4.5×10^{-3} s	8.5×10^{-3} s
40	basic	51 %	23512	810 s	223 s	9.5×10^{-3} s	4.4×10^{-2} s
	$T^{\text{equ}}_{J=14}$	11 %	6336	301 s	527 s	8.3×10^{-2} s	1.3×10^{-1} s
	$T^{\text{equ}}_{J=29}$	90 %	39800	497 s	4848 s	1.2×10^{-1} s	1.3×10^{-1} s
	$T^{\text{ada}}_{J=14}$	91 %	31144	394 s	1327 s	4.3×10^{-2} s	5.5×10^{-2} s
	$\text{NN}^{\text{partial}}$	88 %	43483	246 s	465 s	1.1×10^{-2} s	1.6×10^{-2} s
	NN^{full}	93 %	43433	146 s	470 s	1.1×10^{-2} s	1.4×10^{-2} s

TABLE 5.5: Convergence and speed of BVP solutions, measured for 100 random initial conditions, depending on the dimension of the bilinear control problem and the initialization technique including basic, time-marching and NN warm start.

In conclusion, we have shown that both advanced initialization techniques, time-marching and NN warm start, consistently produce convergent BVP solutions and are thus highly beneficial for solving complicated and high dimensional problems that cannot be tackled by a straightforward approach. It should be noted that the results presented in this experiment are based on uniformly at random sampled initial conditions, and using selected initial conditions in particularly challenging regions would likely yield even more significant advantages.

We found that the effectiveness of time-marching is strongly dependent on the sequence of intermediate times, with adaptive time steps of increasing distance performing best in terms of convergence and time-efficiency. NN warm start, on the other hand, does not require any tuning and works even with models of low accuracy trained for short periods of time. Moreover, its speed scales much better with the problem dimension n.

Therefore, the most efficient strategy for data generation is to use time-marching to generate the initial training and validation sets, D^0_{train} and D_{val}, and then switch to NN warm start for progressive data generation during the training phase once a

low-fidelity NN model is available. When producing the test set D_{test}, NN warm start is most effective as it is based on a fully trained high-fidelity model. For easier problems, though, a basic initialization of the BVP solver is sufficient, and using time-marching for initial data generation would only increase computation time unnecessarily. This explains why we did not employ this technique in the first experiments of this chapter.

5.2.6 Simulation for Out-of-Domain Initial Conditions

Last but not least, we simulate the control, the resulting state and costate trajectories, as well as the value function, for a specific initial condition that lies outside the domain $\mathbb{X}_0$ for which the neural network has been trained. Since the simulation is undisturbed, the open-loop solution computed by solving the associated boundary value problem yields the true optimal control against which the closed-loop solution obtained through the NN model is benchmarked.

Once again, we turn our attention to the bilinear control problem examined before. For the simulation, we utilize the NN^{full} model trained on a fine spatial discretization of $n = 40$, with previous numerical results displayed in Table 5.4 and 5.5. In this simulation, we aim to drive the state towards the quadratic target Y^{quad}, starting from the sine function scaled by a factor of two, that is

$$X_0^{\text{scaledsin}}(\omega) = 2X_0^{\text{sin}}(\omega).$$

Thus, the initial condition exceeds the interval $[-1.5, 1.5]$ from which the data used for optimization was sampled.

In the uncontrolled system, the non-linear reaction causes the state to explode for larger initial conditions such as $X_0^{\text{scaledsin}}$. Even though the model has not been trained for initial conditions of this magnitude, we find that the NN controller successfully prevents the blow-up. The same behavior can be observed for other initial conditions that lie far beyond $\mathbb{X}_0$.

Moreover, as shown in the top row of Figure 5.7, the state trajectory of the NN solution closely resembles that of the BVP solution. However, due to the initial condition being a sine curve, neither state fully reaches the quadratic function. Instead, the parabolic shape of the final state is slightly shifted towards the left within the spatial interval Ω, with the NN prediction having a slightly higher vertex compared to the true optimal solution provided via the BVP.

Larger differences show up in the costate plots in the bottom row of Figure 5.7 at the initial time t_0, where the costate of the NN solution fluctuates heavily but quickly smooths out as time progresses. To understand this behavior, we have to keep in mind that the costate is predicted as the gradient of the value function approximation based on the relation (2.15). Upon examining the right plot of Figure 5.8, the largest approximation error regarding the value function along the optimal state trajectory occurs indeed at the beginning of the time interval. At t_0, when some components of the discretized state fall outside the domain $\mathbb{X}_0$ from which the initial conditions for data generation were drawn, the model is confronted with data that is significantly different from the training and validation sets. However, as soon as the state is steered closer to the target, the model performs much better, as the state values become more familiar. Consequently, the predicted value function then runs fairly parallel to the true cost-to-go which is steadily decreasing until it reaches the final cost at t_f. As becomes evident from the graph of the value function as well

as from Table 5.6, the final cost of the NN solution is slightly larger because the state deviates more from the quadratic target in comparison to the BVP solution.

solution technique	running state cost	running control cost	final state cost	total cost
NN	1.34	1.56	0.12	3.02
BVP	1.36	0.20	0.10	1.66

TABLE 5.6: Costs with their components resulting from simulations for the bilinear control problem with initial condition $X_0^{\text{scaledsin}}$.

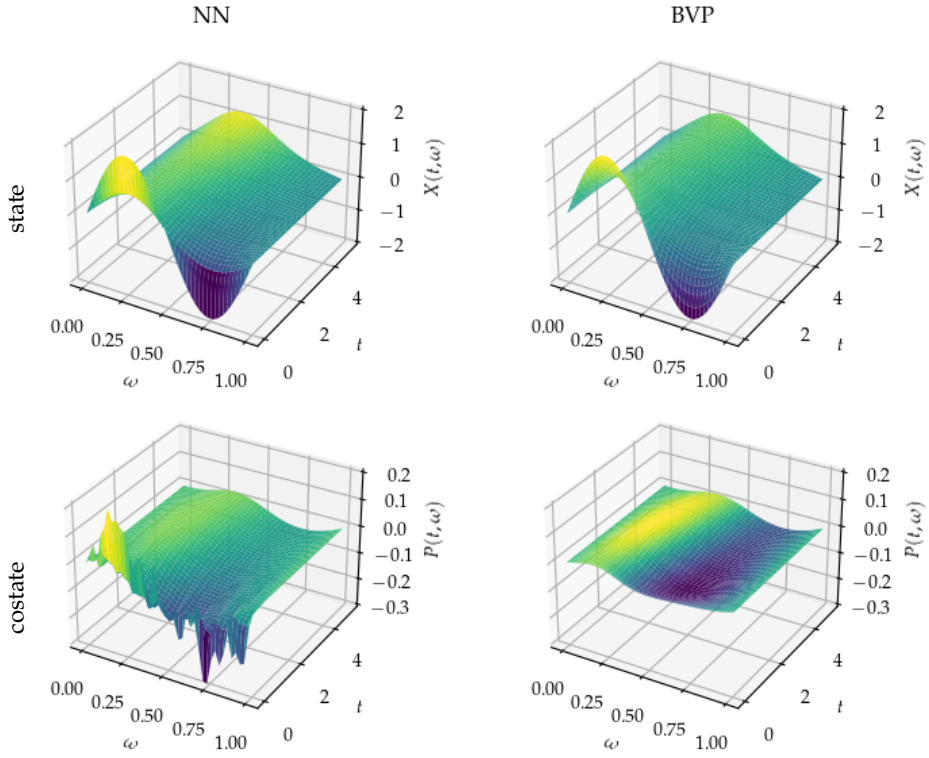

FIGURE 5.7: Simulated state (top row) and costate (bottom row) trajectories corresponding to the NN (left column) and BVP (right column) solutions for the bilinear control problem starting from the initial condition $X_0^{\text{scaledsin}}$ and aiming to reach the target Y^{quad}.

Since the feedback control law is based on the costate prediction, the initial inaccuracy is also reflected in the NN controller as shown on the left of Figure 5.8. In the optimal BVP solution, the control effort is relatively large in the beginning to drive the state from the initial condition towards the target at an exponential rate. This effectively avoids high running state costs during the remaining time interval. Subsequently, the optimal control stays close to zero, merely compensating for the diffusion process. In case of the NN solution, however, the initial control use is much higher. According to Table 5.6, this results in significantly larger running control costs which can not be balanced out by slightly smaller running state costs. As a result, the total cost of the NN solution is almost double that of the BVP solution.

Although the data-driven approach has a local nature, the NN model still manages to handle state values outside the training domain. In particular, it is able to avoid blow-ups but with an increased objective functional value. Hence, we have

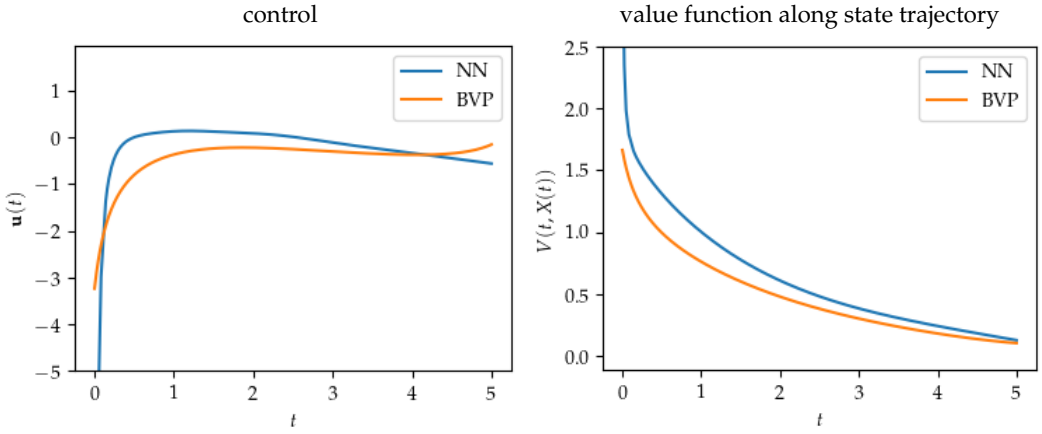

FIGURE 5.8: Simulated controls (left) and value functions (right) for the bilinear control problem with initial condition $X_0^{\text{scaledsin}}$.

shown that the DL-based method does not immediately break down when encountering an initial condition for which no or only sparse training data is available. Combining these findings with our results from noisy simulations in Section 5.2.1, we have good reason to believe that the NN controller can tackle practical scenarios in which the dynamical system experiences heavy disturbances that propel the state beyond the domain learned by the model. In such cases, we expect the neural network to effectively guide the system back, although at the expense of suboptimal costs. The demonstrated robustness makes the closed-loop NN solution highly valuable for real-world applications.

Chapter 6

Conclusion and Future Research

6.1 Conclusion

This thesis addresses the computational bottleneck of lacking effective algorithms for feedback design in general optimal control problems (OCPs). Specifically, we have established an adaptive deep learning (DL) framework to solve Hamilton-Jacobi-Bellman (HJB) equations over semi-global domains, aiming to compute candidate optimal closed-loop controllers. At the core of our framework lies a neural network (NN) model which is trained on data obtained through the open-loop solutions resulting from associated boundary value problems (BVPs) derived via Pontryagin's Minimum Principle (PMP). Building data sets progressively opens up a large range of possibilities, allowing us to determine when, how much and where data is generated. The adaptivity of the method is achieved by carefully selecting the locations for data generation, ensuring that the NN approximation of the value function and the predicted control are particularly precise in the areas of interest. Using a NN-based approach comes usually with the drawback of not having mathematically proven error upper bounds available. However, our causality-free data generation method enables us to compute approximate errors to empirically validate the model accuracy.

In order to be tractable for high-dimensional problems with complicated state dynamics, our approach places a strong emphasis on exploiting prior information, using data efficiently, promoting generalization, and reducing the sensitivity to hyperparameter tuning. To accomplish these objectives, we pay special attention to several critical aspects, including:

- designing a well-suited NN model architecture for value function approximation,
- developing an initialization strategy for the BVP solver,
- incorporating stochastic and physics-informed learning techniques, and
- utilizing sophisticated heuristics guiding the data generation, such as early stopping, sample size selection and adaptive sampling.

In numerical examples, we have demonstrated that deep neural networks can be used as an effective tool for handling non-linear states and bilinear controls. Moreover, we showed the potential for scalability of the DL-based method by tackling OCPs in up to 40 dimensions. Empirical validation and specific simulations have yielded encouraging results, suggesting that the curse of dimensionality can be overcome or at least mitigated by employing our framework. Furthermore, the NN models seem to generalize fairly well even beyond the domains for which they were optimized. This, combined with their ability to react to model uncertainty and

external disturbances, highlights the robustness of the NN controllers. Since the computational effort is mainly tackled offline during the training phase, the online evaluation of the feedback law is accomplished with remarkable speed. As a result, the adaptive deep learning framework is highly suitable for practical settings that involve noise and require real-time control.

6.2 Future Research

In the scope of this thesis, we have explored various promising ideas, but there are still open questions and avenues for future development. For instance, it remains unclear how well the proposed DL-based approach works for even larger problems or in other challenging applications. Since theoretical results regarding network behavior, convergence properties and generalization capabilities are rare, we have to rely mainly on numerical experiments. Further investigation, including a rigorous stability analysis, is crucial for enhancing the existing framework and paves the way for even more reliable and efficient closed-loop controllers. An intriguing direction for further examination is testing alternative model structures or network architectures, learning methods and optimizers, as well as algorithms for data generation. Special attention has to be paid to fine-tuning the respective hyperparameters. Adjusting training parameters, such as learning rates, regularization weights, and batch sizes, during the optimization process remains an active area of research. Developing more efficient and automated techniques for hyperparameter tuning can significantly improve the convergence and overall performance of the NN solution.

We further note that the underlying concept of progressive data generation is quite general and thus applicable to other data-driven problems where data is scarce but can be generated gradually. However, this work contains only a brief numerical evaluation of the heuristical components given by the early stopping, the data set size selection and the adaptive sampling criterion. Therefore, we recommend further research to examine these criteria isolated from each other, in order to understand their individual effects on computational costs and model performance. By employing progressive data augmentation in different settings beyond the OCPs addressed in this work, we can gain further insight into the circumstances under which the use of certain heuristics may be beneficial.

Of particular interest are extensions of the adaptive deep learning framework to solve infinite-horizon or minimum-time problems, as well as to handle state and control constraints. Besides that, the treatment of infinite dimensional controlled systems presents a possible research direction. In general problem settings, where the solution may not be unique, learning the globally or locally optimal solution becomes a further challenge. Moreover, the algorithm could be modified to tackle partial differential equations (PDEs) other than HJB equations, expanding its applicability beyond the field of optimal feedback design.

Bibliography

[1] E.G. Al'brekht. "On the optimal stabilization of nonlinear systems". In: *Journal of Applied Mathematics and Mechanics* 25.5 (1961), pp. 1254–1266. ISSN: 0021-8928. DOI: 10.1016/0021-8928(61)90005-3.

[2] Achref Bachouch, Côme Huré, Nicolas Langrené, and Huyên Pham. "Deep Neural Networks Algorithms for Stochastic Control Problems on Finite Horizon: Numerical Applications". In: *Methodology and Computing in Applied Probability* 24.1 (2021), pp. 143–178. DOI: 10.1007/s11009-019-09767-9.

[3] M. Bardi and I. Capuzzo-Dolcetta. *Optimal Control and Viscosity Solutions of Hamilton-Jacobi-Bellman Equations*. Modern Birkhäuser Classics. Birkhäuser Boston, 2009. ISBN: 9780817647551.

[4] R. Bellman, R.E. Bellman, and Karreman Mathematics Research Collection. *Adaptive Control Processes: A Guided Tour*. Princeton Legacy Library. Princeton University Press, 1961. ISBN: 9780691079011.

[5] Olivier Bokanowski, Jochen Garcke, Michael Griebel, and Irene Klompmaker. "An Adaptive Sparse Grid Semi-Lagrangian Scheme for First Order Hamilton-Jacobi Bellman Equations". In: *Journal of Scientific Computing* 55 (June 2013). DOI: 10.1007/s10915-012-9648-x.

[6] J. Frédéric Bonnans. *Optimal control of ordinary differential equations*. Lecture notes. 2008. URL: http://www.cmap.polytechnique.fr/~bonnans/notes/oc/traj.pdf.

[7] J. Frédéric Bonnans. *Part I: the Pontryagin approach*. Lecture notes. 2019. URL: http://www.cmap.polytechnique.fr/~bonnans/notes/oc/ocbook.pdf.

[8] H. Brezis. *Functional Analysis, Sobolev Spaces and Partial Differential Equations*. Universitext. Springer New York, 2010. ISBN: 9780387709147.

[9] Richard Byrd, Gillian Chin, Jorge Nocedal, and Yuchen Wu. "Sample Size Selection in Optimization Methods for Machine Learning". In: *Mathematical Programming* 134 (Aug. 2012), 127–155. DOI: 10.1007/s10107-012-0572-5.

[10] Simone Cacace, Emiliano Cristiani, Maurizio Falcone, and Athena Picarelli. "A Patchy Dynamic Programming Scheme for a Class of Hamilton–Jacobi–Bellman Equations". In: *SIAM Journal on Scientific Computing* 34.5 (2012), A2625–A2649. DOI: 10.1137/110841576.

[11] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. "Neural Ordinary Differential Equations". In: *Advances in Neural Information Processing Systems* 31 (2018). Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett.

[12] Tao Cheng, Frank Lewis, and Murad Abu-Khalaf. "Fixed-Final-Time-Constrained Optimal Control of Nonlinear Systems Using Neural Network HJB Approach". In: *Neural Networks, IEEE Transactions on* 18 (Dec. 2007), pp. 1725 –1737. DOI: 10.1109/TNN.2007.905848.

[13] Tao Cheng, Frank L. Lewis, and Murad Abu-Khalaf. "A neural network solution for fixed-final time optimal control of nonlinear systems". In: *Automatica* 43.3 (2007), pp. 482–490. ISSN: 0005-1098. DOI: 10.1016/j.automatica.2006.09.021.

[14] François Chollet. *Deep Learning with Python*. Manning Publications Company, 2017. ISBN: 9781617294433.

[15] Yat Tin Chow, Wuchen Li, Stanley Osher, and Wotao Yin. *Algorithm for Hamilton-Jacobi equations in density space via a generalized Hopf formula*. 2018. DOI: 10.48550/ARXIV.1805.01636.

[16] Jérôme Darbon and Stanley Osher. *Algorithms for Overcoming the Curse of Dimensionality for Certain Hamilton-Jacobi Equations Arising in Control Theory and Elsewhere*. 2016. DOI: 10.48550/ARXIV.1605.01799.

[17] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. "Augmented Neural ODEs". In: *Advances in Neural Information Processing Systems* 32 (2019). Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett.

[18] A. Ern and J.L. Guermond. *Theory and Practice of Finite Elements*. Applied Mathematical Sciences. Springer New York, 2004. ISBN: 9780387205748.

[19] Maurizio Falcone and Roberto Ferretti. *Semi-Lagrangian Approximation Schemes for Linear and Hamilton—Jacobi Equations*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2013. DOI: 10.1137/1.9781611973051.

[20] P.C. Fife. *Mathematical Aspects of Reacting and Diffusing Systems*. Lecture notes in biomathematics. Springer-Verlag, 1979. ISBN: 9780387091174.

[21] P. Garabedian. *Partial Differential Equations*. AMS Chelsea Publishing Series. Chelsea Publishing Company, 1986. ISBN: 9780821813775.

[22] Jochen Garcke and Axel Kröner. "Suboptimal Feedback Control of PDEs by Solving HJB Equations on Adaptive Sparse Grids". In: *J. Sci. Comput.* 70.1 (2017), 1–28. ISSN: 0885-7474. DOI: 10.1007/s10915-016-0240-7.

[23] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 1st. O'Reilly Media, Inc., 2017. ISBN: 1491962291.

[24] Elisa Giesecke and Axel Kröner. "Classification with Runge-Kutta networks and feature space augmentation". In: *Journal of Computational Dynamics* 8.4 (2021), pp. 495–520. ISSN: 2158-2491. DOI: 10.3934/jcd.2021018.

[25] Daeyung Gim and Hyungbin Park. *A deep learning algorithm for optimal investment strategies*. 2021. DOI: 10.48550/ARXIV.2101.12387.

[26] Roland Glowinski, Yongcun Song, Xiaoming Yuan, and Hangrui Yue. "Bilinear Optimal Control of an Advection-Reaction-Diffusion System". In: *SIAM Review* 64.2 (2022), pp. 392–421. DOI: 10.1137/21M1389778.

[27] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[28] Eldad Haber and Lars Ruthotto. "Stable architectures for deep neural networks". In: *Inverse Problems* 34.1 (2017), p. 014004. ISSN: 1361-6420. DOI: 10.1088/1361-6420/aa9a90.

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[30] C.F. Higham and D.J. Higham. "Deep learning: an introduction for applied mathematicians". In: *SIAM Rev.* 61.4 (2019), pp. 860–891. ISSN: 0036-1445. DOI: 10.1137/18M1165748.

[31] Côme Huré, Huyên Pham, Achref Bachouch, and Nicolas Langrené. "Deep Neural Networks Algorithms for Stochastic Control Problems on Finite Horizon: Convergence Analysis". In: *SIAM Journal on Numerical Analysis* 59.1 (2021), pp. 525–557. DOI: 10.1137/20m1316640.

[32] Dario Izzo, Ekin Öztürk, and Marcus Märtens. "Interplanetary transfers via deep representations of the optimal policy and/or of the value function". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 2019. DOI: 10.1145/3319619.3326834.

[33] Frank Jiang, Glen Chou, Mo Chen, and Claire J. Tomlin. *Using Neural Networks to Compute Approximate and Guaranteed Feasible Hamilton-Jacobi-Bellman PDE Solutions*. 2016. DOI: 10.48550/ARXIV.1611.03158.

[34] Dante Kalise and Karl Kunisch. "Polynomial Approximation of High-Dimensional Hamilton–Jacobi–Bellman Equations and Applications to Feedback Control of Semilinear Parabolic PDEs". In: *SIAM Journal on Scientific Computing* 40.2 (2018), A629–A652. DOI: 10.1137/17m1116635.

[35] Wei Kang, Qi Gong, and Tenavi Nakamura-Zimmerer. *Algorithms of Data Development For Deep Learning and Feedback Design*. 2019. DOI: 10.48550/ARXIV.1912.00492.

[36] Wei Kang and Lucas C. Wilcox. *Mitigating the Curse of Dimensionality: Sparse Grid Characteristics Method for Optimal Feedback Control and HJB Equations*. 2015. DOI: 10.48550/ARXIV.1507.04769.

[37] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima". In: *International Conference on Learning Representations*. 2017.

[38] Jacek Kierzenka and Lawrence F. Shampine. "A BVP Solver Based on Residual Control and the Maltab PSE". In: *ACM Trans. Math. Softw.* 27.3 (2001), 299–316. ISSN: 0098-3500. DOI: 10.1145/502800.502801.

[39] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations* (Dec. 2014).

[40] Axel Kröner. "Optimal Control of Partial Differential Equations". Lecture notes. 2019.

[41] Karl Kunisch and Daniel Walter. *Semiglobal optimal feedback stabilization of autonomous systems via deep neural network approximation*. 2020. DOI: 10.48550/ARXIV.2002.08625.

[42] Christina Kuttler. *Reaction-Diffusion equations with applications*. Lecture notes. 2011. URL: https://www-m6.ma.tum.de/~kuttler/script_reaktdiff.pdf.

[43] Irena Lasiecka and Roberto Triggiani. *Control Theory for Partial Differential Equations: Continuous and Approximation Theories I*. Vol. I. Feb. 2000, i–xxii and 645. ISBN: 0521584019. DOI: 10.1017/CBO9780511574801.

[44] Daniel Liberzon. *Calculus of Variations and Optimal Control Theory: A Concise Introduction*. Princeton University Press, 2012. ISBN: 9780691151878.

[45] P.-L Lions and Jean Charles Rochet. "Hopf Formula and Multitime Hamilton-Jacobi Equations". In: *Proceedings of The American Mathematical Society - PROC AMER MATH SOC* 96 (Jan. 1986), pp. 79–79. DOI: 10.1090/S0002-9939-1986-0813815-5.

[46] D. L. Lukes. "Optimal Regulation of Nonlinear Dynamical Systems". In: *SIAM Journal on Control* 7.1 (1969), pp. 75–100. DOI: 10.1137/0307007.

[47] O. L. Mangasarian. "Sufficient Conditions for the Optimal Control of Nonlinear Systems". In: *SIAM Journal on Control* 4.1 (1966), pp. 139–152. DOI: 10.1137/0304013.

[48] W. Thomas Miller, Richard S. Sutton, and Paul J. Werbos, eds. *Neural Networks for Control*. Cambridge, MA, USA: MIT Press, 1990. ISBN: 0262132613.

[49] Tenavi Nakamura-Zimmerer, Qi Gong, and Wei Kang. "Adaptive Deep Learning for High-Dimensional Hamilton–Jacobi–Bellman Equations". In: *SIAM Journal on Scientific Computing* 43.2 (2021), A1221–A1247. DOI: 10.1137/19m1288802.

[50] Carmeliza Navasca and Arthur J. Krener. "Patchy Solutions of Hamilton-Jacobi-Bellman Partial Differential Equations". In: *Modeling, Estimation and Control: Festschrift in Honor of Giorgio Picci on the Occasion of his Sixty-Fifth Birthday*. Ed. by Alessandro Chiuso, Stefano Pinzoni, and Augusto Ferrante. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 251–270. ISBN: 978-3-540-73570-0. DOI: 10.1007/978-3-540-73570-0_20.

[51] L.S. Pontryagin. *The Mathematical Theory of Optimal Processes*. International series of monographs in pure and applied mathematics. Pergamon Press; [distributed in the Western Hemisphere by Macmillan, New York], 1964. ISBN: 9780080101767.

[52] Lutz Prechelt. "Early Stopping — But When?" In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 53–67. ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_5.

[53] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378 (2019), pp. 686–707.

[54] Rolf Rannacher. *Numerische Mathematik 1: Numerik gewöhnlicher Differentialgleichungen*. Lecture notes. 2014. URL: https://ganymed.math.uni-heidelberg.de/~lehre/notes/num1/Numerik_1.pdf.

[55] H. Schättler and U. Ledzewicz. *Geometric Optimal Control: Theory, Methods and Examples*. Interdisciplinary Applied Mathematics. Springer New York, 2012. ISBN: 9781461438342.

[56] Justin Sirignano and Konstantinos Spiliopoulos. "DGM: A deep learning algorithm for solving partial differential equations". In: *Journal of Computational Physics* 375 (2018), pp. 1339–1364. DOI: 10.1016/j.jcp.2018.08.029.

[57] Eduardo D. Sontag. *Mathematical Control Theory: Deterministic Finite Dimensional Systems (2nd Ed.)* Berlin, Heidelberg: Springer-Verlag, 1998. ISBN: 0387984895.

[58] Yuval Tassa and Tom Erez. "Least Squares Solutions of the HJB Equation With Neural Network Value-Function Approximators". In: *Neural Networks, IEEE Transactions on* 18 (Aug. 2007), pp. 1031 –1041. DOI: 10.1109/TNN.2007.899249.

[59] E Weinan. "A proposal on machine learning via dynamical systems". In: *Communications in Mathematics and Statistics* 5.1 (2017), pp. 1–11.

[60] Ivan Yegorov and Peter M. Dower. "Perspectives on Characteristics Based Curse-of-Dimensionality-Free Numerical Approaches for Solving Hamilton–Jacobi Equations". In: *Applied Mathematics &amp Optimization* 83.1 (2018), pp. 1–49. DOI: 10.1007/s00245-018-9509-6.

Selbstständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen, einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Berlin, den 06.06.2023

Elisa Giesecke