12-2024

Neural Networks and Approximation of High-dimensional Functions: Applications in Control and Partial Differential Equations

Nathan Gaby
Georgia State University

Neural Networks and Approximation of High-dimensional Functions: Applications in Control and Partial Differential Equations

by

Nathan Gaby

Under the Direction of Xiaojing Ye, Ph.D.

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

Year 2024

ABSTRACT

We investigate the usefulness of deep learning when applied to both control theory and partial differential equations (PDEs). We will develop new network architectures and methodologies to approach the solving of high-dimensional problems. Specifically, we develop a network architecture called Lyapunov-Net for approximating Lyapunov functions in high-dimensions and a new methodology called Neural Control for finding solution operators for high-dimensional parabolic PDEs. The theoretical accuracy and numerical efficiency of these approaches will be investigated along with implementation details to use them in practice.

INDEX WORDS: Lyapunov Functions, Deep Neural Networks, Control, Operator Learning, Partial Differential Equations

Neural Networks and Approximation of High-Dimensional Functions: Application in Control and Partial Differential Equations

by

Nathan Gaby

DEDICATION

To my wonderful wife Mary Clayton. I could not have done all that I do, with as much sanity as I do, without your love and encouragement.

ACKNOWLEDGMENTS

There are numerous people who have assisted, encouraged, taught, influenced, guided, and inspired me over the years it has taken to complete this dissertation. Many of these people were influential in my research, my love of mathematics, and my life more generally. First, I have to thank my advisor Xiaojing Ye, he was the reason I came to Georgia State to receive my PhD. Without him, I would not have been able to develop the research skills needed to engage in this work, or the tenacity needed to persist. Dr. Ye has inspired me, taught me, pushed me, and encouraged me over these four and a half years and I am immensely grateful for all of his mentorship, teaching, and encouragement along the way.

I want to thank my committee, Haomin Zhou for your wisdom and perspective on this work and your kind words of support, Alexandra Smirnova and Michael Stewart for graciously serving on my committee and for your thoughtful comments all while taking the time out of your busy days serving as the chair and co-chair of our department.

I also want to thank and acknowledge my undergraduate advisor Ron Taylor, I would not have ever taken this journey or been as prepared as I was to start it without his mentorship, teaching, and friendship. I want to thank my parents and the enormous support they have been throughout my many years of schooling. Thank you to my wife Mary Clayton, she has been there through it all and has been the firmest supporter of our family throughout my education. Finally, I thank the Lord for the numerous blessings he has given, none of which I have ever deserved.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

PART 1
INTRODUCTION

In recent years, deep learning methods have been used to solve many high-dimensional problems previously intractable to traditional numerical schemes. Led by the rapid expansion of computing power enabled through ever smaller and more powerful microchips, deep learning has seen success in applications ranging from computer vision [69], to natural language processing [116], to generative modeling [35, 105], to function approximation [123]. The power of huge models called deep neural networks (DNNs) has been of particular interest. These models are not new [2], however, modern computers and advances in numerical calculus through tools like automatic differentiation have made DNNs the central focus of much modern research in machine learning, and more generally in artificial intelligence.

For this dissertation, we will investigate DNNs and their usefulness in approximating functions. Specifically, we will look at the problem of finding high-dimensional Lyapunov functions with applications in control theory and the problem of finding solution operators to solve high-dimensional Partial Differential Equations (PDEs). These two applications represent the power of DNNs when solving high-dimensional problems of the sort previously unapproachable by traditional methods.

1.1 Neural Networks and Universal Approximation

DNNs can be viewed as nonlinear reduced-order models, and are powerful tools in solving high-dimensional PDEs in recent years [7, 28, 43, 92, 93, 94, 124]. Mathematically, a DNN can be expressed as the composition of a series of simple linear and nonlinear functions.

In the deep learning context, a typical building block of DNNs is called a *layer*, which is a mapping $h : \mathbb{R}^d \to \mathbb{R}^{d'}$ for some compatible input dimension d and output dimension d' defined by:

$$h(z; W, b) := \sigma(Wz + b), \tag{1.1}$$

where $z \in \mathbb{R}^d$ is the input variable of h, the matrix $W \in \mathbb{R}^{d' \times d}$ and vector $b \in \mathbb{R}^{d'}$ are called the weight and bias respectively, and $\sigma : \mathbb{R} \to \mathbb{R}$ is a nonlinear function that operates componentwise on its d'-dimensional argument vector $Wz + b$ (hence σ is effectively a mapping from $\mathbb{R}^{d'}$ to $\mathbb{R}^{d'}$). While in theory any non-polynomial function can serve as an activation function, some common choices of activation functions include the hyperbolic tangent (tanh) and rectified linear unit (ReLU) $\sigma(z) = \max(0, z)$ functions. A commonly used DNN structure u_θ, often called a feed-forward network (FFN), is defined as the composition of multiple layer functions of form (1.1) as follows:

$$u_\theta(x) := u(x; \theta) = w^\top z_L + b, \tag{1.2}$$
$$\text{where} \quad z_0 = x, \quad z_l = h_l(z_{l-1}) := h(z_{l-1}; W_l, b_l), \quad l = 1, \dots, L,$$

and the lth hidden layer $h(\cdot; W_l, b_l) : \mathbb{R}^{d_{l-1}} \to \mathbb{R}^{d_l}$ is determined by its weight and bias parameters $W_l \in \mathbb{R}^{d_l \times d_{l-1}}$ and $b_l \in \mathbb{R}^{d_l}$ for $l = 1, \dots, L$ and $d_0 = d$. Here the output of u_θ is set to the affine transform of the last hidden layer $z_{NN} = h_L(z_{L-1})$ using weight $w \in \mathbb{R}^{d_L}$ and bias $b \in \mathbb{R}$. The *network parameters* θ refers to the collection of all learnable parameters (stacked as a vector in $\mathbb{R}^m$) of u_θ, i.e.,

$$\theta := (w, b, W_L, b_L, \ldots, W_1, b_1) \in \mathbb{R}^m. \tag{1.3}$$

The process of training the network u_θ refers to finding the minimizer θ of some properly designed loss function which takes θ as input.

Early research on the approximation capabilities of neural networks dates back to the 1980s from which we get one of the first universal approximation theorems due to [48].

Theorem 1. *Let $\Omega \subset \mathbb{R}^d$ be compact, $g \in C(\Omega)$ and $\epsilon > 0$. For a DNN of the form* (1.2) *with any continuous non-constant activation function σ there exists a large enough neural network u_θ such that*

$$\sup_{x\in\Omega} |g(x) - u_\theta(x)| < \epsilon.$$

In particular, Hornik in [48] showed that a single hidden layer was sufficient for universal approximation. Following this early result and others in the 90s the interest in neural networks waned. However, as previously mentioned, in recent years DNNs have been investigated and new studies have shown their power in approximating high-dimensional functions, see e.g. [39, 40, 66, 70, 89, 123]. Compared to the approach of Hornik, many of the modern universal approximation theorems (such as [123]) come with approximation rates. These rates show that the number of parameters of a DNN needed to approximate a Lipschitz function to error ϵ scales as $O(\epsilon^{-d})$ in the worst case. This theory suggests weights should scale exponentially in dimension, yet in practice this does not play out, suggesting there yet remains a gap in the theory.

Ultimately, these results provide the theoretical underpinning which justify the use of DNNs in different contexts. For the rest of this dissertation, we will seek to show how DNNs can be utilized in both control and PDEs to develop novel techniques to solve classical numerical problems in high-dimensions. The theoretical advances presented in this dissertation will make use of varying modern universal approximation results stated for the more general Sobolev spaces. We will use the standard notation to define the Sobolev space $W^{k,p}(\Omega)$ and will use the norm defined by

$$\|g\|_{W^{k,p}(\Omega)} = \max\left\{\|g(x)\|_p, \max_{1\le i\le d}\|D_i g(x)\|_p\right\}$$

where $D_i g(x)$ is the weak partial derivative of g with respect to x_i at x, $\|\cdot\|_p$ is the regular $L^p(\Omega)$ norm. For $p=\infty$ we have

$$\|g\|_{W^{k,p}(\Omega)} = \max\left\{\operatorname*{ess\,sup}_{x\in\Omega}|g(x)|, \max_{1\le i\le d}\operatorname*{ess\,sup}_{x\in\Omega}|D_i g(x)|\right\}.$$

For Sobolev spaces it has been shown in works such as [39] that for any $M, \varepsilon > 0$, $k \in \mathbb{N}$, $p \in [1,\infty]$, and $\Omega = (0,1)^d \subset \mathbb{R}^d$, denote $\mathcal{F} := \{f \in W^{k,p}(\Omega;\mathbb{R}) \,|\, \|f\|_{W^{k,p}(\Omega)} \le M\}$, there exists a DNN structure u_θ of form (1.2) with sufficiently large m and L (which depend on M, ε, d and p only), such that for any $f \in \mathcal{F}$, there is $\|u_\theta - f\|_{W^{k,p}(\Omega)} \le \varepsilon$ for some $\theta \in \mathbb{R}^m$. We will use variations on this result and others throughout the rest of this work and will recite or reference these variations in the text as necessary.

PART 2

LYAPUNOV-NET: A DEEP NEURAL NETWORK ARCHITECTURE FOR LYAPUNOV FUNCTION APPROXIMATION

2.1 Introduction

In this part, we will discuss the first of two application of DNNs to high-dimensional problems previously unapproachable to traditional methods. This problem is approximating Lyapunov functions for use in control. We first recall the definition of Lyapunov function for a given general dynamical system $x' = f(x)$ with Lipschitz continuous f on the problem domain Ω. We assume that $x = 0$ is the unique equilibrium of the system dynamics that lies within Ω. If $x = 0$ is asymptotically stable, then by the converse Lyapunov theory, we can find a Lyapunov function V defined as below.

Definition 1 (Lyapunov function)**.** *Let $\Omega \subset \mathbb{R}^d$ be a bounded open set and $0 \in \Omega$, and $f : \Omega \to \mathbb{R}^d$ a Lipschitz function. Then $V : \Omega \to \mathbb{R}$ is called a* Lyapunov function *if (i) V is positive definite, i.e., $V(x) \geq 0$ for all $x \in \Omega$ and $V(x) = 0$ if and only if $x = 0$; and (ii) V has negative orbital-derivative, i.e., $DV(x) \cdot f(x) < 0$ for all $x \neq 0$.*

DNNs have emerged for the approximation of Lyapunov functions of nonlinear and non-polynomial dynamical systems in high-dimensional spaces [16, 22, 36, 98, 120].

For control Lyapunov functions, DNNs can also be used to approximate the control laws, eliminating the restrictions on control laws to specific function types (e.g., affine functions) in classical control methods such as linear-quadratic regulator (LQR).

We shall propose a general framework to approximate Lyapunov functions and control laws using DNNs. This part's main contributions will lie in the following aspects:

1. We propose a highly versatile network architecture, called *Lyapunov-Net*, to approximate Lyapunov functions. Specifically, this network architecture guarantees the desired positive definiteness property. This leads to simple parameter tuning, significantly accelerated convergence during network training, and greatly improved solution quality. This will allow for fast adoption of neural networks in (control) Lyapunov function approximation in a broad range of applications.

2. We show that the proposed Lyapunov-Nets are dense in a general class of Lyapunov functions. More importantly, we prove that the Lyapunov-Nets trained by minimizing empirical risk functions using finitely many collocation points are Lyapunov functions, which provides a theoretical certification guarantee of neural network based Lyapunov function approximation.

3. We test the proposed Lyapunov-Net to solve several benchmark problems numerically. We show that our method can effectively approximate Lyapunov functions in these problems with very high state dimensionality. Moreover, we demonstrate that our method can be used to find control laws in the control Lyapunov problem setting.

2.2 Background

Approximating Lyapunov functions using neural networks can be dated back to [74, 106]. In [74] the authors attempted the idea assuming that a shallow neural network can exactly represent the target Lyapunov function. In [106], stabilization problems in control using neural networks with one or two hidden layers are considered. In [57], the control Lyapunov functions (CLF) using quadratic Lyapunov function candidate is considered. A DNN approach to CLF is considered in [1, 16] which are similar. Approximating stabilizing controllers using neural networks is considered in [65]. Time discretized dynamics and successive parameter updates are considered in [98, 100]. Specifically, [98] considers jointly learning the Lyapunov function and its decreasing region which is expected to match the Region of Attraction (ROA). Still more recently some authors have considered finding Lyapunov functions by solving specific PDEs given by Zubovs equations [72].

Special network architectures to approximate Lyapunov functions are also considered in [98]. In [98], a Lyapunov neural network of form $\|\phi_\theta(\cdot)\|^2$ is proposed to ensure positive semi-definiteness, where $\phi_\theta(\cdot)$ is a DNN. To ensure positive definiteness, the authors restrict ϕ_θ to be a feed-forward neural network, all weight matrices to have full column rank, all biases to be zero, and all activation functions to have trivial null space (e.g., tanh or leaky ReLU but not sigmoid, swish, softmax, ReLU, or RePU). Compared to the architecture in [98], the network architecture in the present work does not have any of these restrictions. Moreover, the positive definite weight matrix constructions in [98] can make $\|\phi_\theta(x)\|^2$ grow excessively fast as $\|x\|$ increases, whereas ours does not have this issue.

In [36], the author considers dynamical systems with small-gain property, which yields a compositional Lyapunov function structure that has decoupled components. In this case, it is shown that the size of the DNN used to approximate such Lyapunov functions can be dependent exponentially on the maximal dimension of these components rather than the original state space dimension. Still other authors propose neural networks for other control certificates such as barrier functions [22, 99, 120] and contraction metrics [114, 115]. While these are a different direction compared to the approach in this section, they present other promising angles for neural networks in control.

2.3 Proposed Method

In this section, we propose the Lyapunov-Net architecture to approximate Lyapunov functions and discuss its key properties and associated training strategies. Our approach is to approximate a Lyapunov function using a specially designed deep neural network. Due to limited network size and finite collocation points for training in practice, we only guarantee that our approximation function V is positive definite in Ω and satisfies a slightly relaxed condition of Definition 1 (ii).

Denote $B(x;\delta) := \{y \in \mathbb{R}^d : \|y - x\| < \delta\}$ as the open ball of radius $\delta > 0$ centered at x, and $\Omega_\delta := \Omega \setminus B(0;\delta)$ the problem domain with $B(0;\delta)$ excluded. Then the slightly relaxed condition (ii) is as follows:

$$DV(x) \cdot f(x) < 0, \quad \text{for all } x \in \Omega_\delta. \tag{2.1}$$

where $\delta > 0$ is arbitrary and prescribed by the user. We term such a function V as a *δ-accurate Lyapunov function.*

A δ-accurate function can be used as a Lyapunov function for $x' = f(x)$ (or control-

Lyapunov function for $f(x, u(x))$ where the control u is to be found jointly with the Lyapunov function, see Section 2.3.4) to prove that the solution $x(t)$ will be *ultimately bounded* within a small compact set (i.e., $\overline{B(0;\delta)}$) [56]. As $\delta \to 0$, the size of this compact set will converge to 0, hence asymptotic stability is established. In practice, the smaller the value of δ, the larger the network size and the more collocation points needed to train such a δ-accurate Lyapunov function.

With this relaxed condition of Lyapunov function in mind, we now proceed by building a versatile deep network architecture that is particularly suitable for finding V_θ for dynamics evolving in high state dimension d. Then we will provide theoretical justifications for this network. Finally, we will demonstrate that the training of this "Lyapunov-Net" renders a minimization problem of a simple risk function, and thus requires much less manual hyper-parameter tuning and achieves high optimization efficiency during practical computations.

2.3.1 Lyapunov-Net and its properties

We first construct an arbitrary DNN $\phi_\theta(\cdot) : \mathbb{R}^d \to \mathbb{R}$ with the set of all its m trainable parameters denoted by $\theta \in \mathbb{R}^m$. This network has input dimension d and output dimension 1. Then we build a scalar-valued network $V_\theta : \mathbb{R}^d \to \mathbb{R}$ from ϕ_θ as follows:

$$V_\theta(x) := |\phi_\theta(x) - \phi_\theta(0)| + \bar{\alpha}\|x\|, \tag{2.2}$$

where $\bar{\alpha} > 0$ is a small user-chosen parameter and $\|\cdot\|$ is the standard 2-norm in Euclidean space. Then it is easy to verify that $V_\theta(0) = 0$ and

$$V_\theta(x) \geq \bar{\alpha}\|x\| > 0, \quad \forall\, x \neq 0.$$

In other words, V_θ is a *candidate* Lyapunov function that already satisfies condition (i) in Definition 1: *for any network structure ϕ_θ with any parameter θ*, V_θ is positive definite and only vanishes at the equilibrium 0. We call the neural network V_θ with architecture specified in (2.2) a *Lyapunov-Net*.

We now make several remarks regarding the Lyapunov-Net architecture (2.2). First, we use the augmented term $\bar{\alpha}\|x\|$ to lower bound the function $V_\theta(x)$ in order to ensure positive definiteness. Other positive definite function $r : \mathbb{R}^d \to \mathbb{R}_+$ such that $r(x) = 0$ if and only if $x = 0$ can be chosen as such lower bound, such as $\bar{\alpha}\|x\|^2$ or $\bar{\alpha}\log(1 + \|x\|^2)$, etc.

Second, the term $|\phi_\theta(x) - \phi_\theta(0)|$ in (2.2) can be replaced with $\psi(\phi_\theta(x) - \phi_\theta(0))$ for any non-negative function $\psi : \mathbb{R}^m \to \mathbb{R}_+$ with $\psi(0) = 0$. We chose $\psi(\cdot) = |\cdot|$ for its application to our theory and simplicity. Note that one could also use a vector-valued DNN $\phi_\theta : \mathbb{R}^d \to \mathbb{R}^{d'}$ where d' is arbitrary which could further improve network capacity. So long as ϕ_θ is Lipschitz continuous then V_θ is Lipschitz continuous and hence weakly differentiable. In practice, we can also use $\|\cdot\|^2$ or Huber norm to smooth out V_θ.

Third, a detailed characterization of the $\bar{\alpha} > 0$ is provided in Section 2.3.3. However, in practice, one chooses $\bar{\alpha}$ freely as Lyapunov functions can be arbitrarily scaled by a positive constant.

Fourth, if the equilibrium is at x^* instead of 0, then one can simply replace $\phi_\theta(0)$ and $\|x\|$ in (2.2) with $\phi_\theta(x^*)$ and $\|x - x^*\|$, respectively. Without loss of generality, we assume the equilibrium is at 0 hereafter in this section.

The properties remarked above show that V_θ defined in (2.2) serves as a versatile network architecture for approximating Lyapunov functions. This architecture significantly eases network training and yields an accurate approximation of Lyapunov functions in practice as we will demonstrate.

2.3.2 Training of Lyapunov-Net

The training of Lyapunov-Net V_θ in (2.2) refers to finding a specific network parameter θ such that the negative-orbital-derivative condition $DV_\theta(x) \cdot f(x) < 0$ is satisfied at every $x \in \Omega \setminus \{0\}$. This is achieved by minimizing a risk function that penalizes V_θ if the negative-orbital-derivative condition fails to hold at some x. We can choose the following as such a risk function:

$$\ell(\theta) := \frac{1}{|\Omega|} \int_\Omega (DV_\theta(x) \cdot f(x) + \gamma \|x\|)_+^2 \, dx, \tag{2.3}$$

where $(z)_+ := \max(z, 0)$ for any $z \in \mathbb{R}$. Here γ is a user-chosen parameter. It is clear that the risk function $\ell(\theta)$ reaches the minimal function value 0 if and only if $DV_\theta(x) \cdot f(x) \le -\gamma \|x\|$ for all $x \in \Omega$, which, in conjunction with V_θ already being positive definite, ensures that V_θ is a Lyapunov function.

In practice, the integral in (2.3) does not have an analytic form, and so Monte-Carlo integration is used to approximate it. This is suitable for high-dimensional problems. To this end, we notice that $\ell(\theta) = \mathbb{E}_{X \sim U(\Omega)}[(DV_\theta(X) \cdot f(X) + \gamma \|x\|)_+^2]$ where $U(\Omega)$ stands for the uniform distribution on Ω and hence has distribution $1/|\Omega|$. Therefore, we approximate $\ell(\theta)$ in (2.3) using the empirical expectation

$$\hat{\ell}(\theta) := \frac{1}{N} \sum_{i=1}^{N} (DV_\theta(x_i) \cdot f(x_i) + \gamma \|x_i\|)_+^2, \tag{2.4}$$

where $\{x_i : i \in [N]\}$ are independent and identically distributed (i.i.d.) samples from $U(\Omega)$. Then we train the Lyapunov-Net V_θ by minimizing $\hat{\ell}(\theta)$ in (2.4) with respect to θ. In this case, standard network training algorithms, such as ADAM [59], can be employed in conjunction with automatic differentiation to calculate gradients. Note that techniques to improve the efficiency of Monte-Carlo integration, such as importance sampling, can be applied. For simplicity, we use uniform sampling in the experiments and leave improved sampling strategies for future investigation.

As discussed earlier, the empirical risk function defined using finitely many sampling points introduces inaccuracies near 0, which is common in the literature. Several existing works [16, 36] observed that the deep-learning-based Lyapunov function approximation may violate the condition $DV_\theta(x)\cdot f(x) < 0$ within a small neighborhood of the equilibrium. Hence why we will only search for δ-accurate Lyapunov functions. This ensures our Lyapunov function will establish ultimate boundedness of the solution $x(t)$, which means that $x(t)$ converges to a small neighborhood $\overline{B(0;\delta)}$ of 0. We note again that $\delta > 0$ can be set arbitrarily small at the expense of a larger network size of V_θ and a greater amount of sampling points. In this case, we can replace Ω with Ω_δ in (2.3) and exclude the points in $B(0;\delta)$ in (2.4). In practical implementation, we just apply standard uniform sampling without such exclusion for simplicity.

The main advantage of the Lyapunov-Net architecture (2.2) is that the risk function (2.3) (or the empirical risk function (2.4)) consists of a single term only with a single parameter. This is in contrast to existing works [16, 36] where the risk functions have multiple terms to penalize the violations of the negative-orbital-derivative condition, positive definiteness condition, bound requirements, etc. Thus, network training in these works requires experienced users to carefully tune the hyper-parameters to properly weigh these penalty terms in order to obtain a reasonable solution. On the other hand, the proposed empirical risk function for Lyapunov-Net V_θ requires little effort in parameter-tuning, and the convergence is much faster in network training, as will be demonstrated in our numerical experiments below.

2.3.3 Lyapunov-Net approximation and certification theory

In this section, we provide theoretical guarantees on the approximation ability of Lyapunov-Net. We shall concern ourselves with the activation function RePU which in this section shall refer to the function $\text{RePU}(x) = \max\{0, x\}^2$. RePU is preferred to the common ReLU as it is C^1 and hence more useful for approximating Lyapunov functions. We note that the forthcoming theory can be extended to other even smoother activation functions (such as tanh or sigmoid). We present RePU in this paper due to its ability to exactly represent polynomials [66] allowing for some proof simplification.

Let $\Omega = [-1, 1]^d$ and $\Omega_\delta = \Omega \setminus B(0; \delta)$. Let $x' = f(x)$ be the dynamical system defined by f. Further, we denote $W^{k,p}(\Omega)$ as the regular Sobolev space over Ω, and $C^k(\Omega)$ as the space of k-times continuously differentiable functions.

Assumption 1. *f is L_f-Lipschitz continuous on Ω for some $L_f > 0$ and $f(0) = 0$.*

Assumption 2. *There exist $V^* \in C^1(\Omega; \mathbb{R})$ and constants $\alpha, \beta, \gamma > 0$, such that for all*

$x \in \Omega$ there are

$$\alpha\|x\| \le V^*(x) \le \beta\|x\|, \tag{2.5a}$$

$$DV^*(x) \cdot f(x) \le -\gamma\|x\|. \tag{2.5b}$$

Assumption 1 implies that $\|f\|_{C(\Omega)} \le L_f\sqrt{d} < \infty$ and $f \in W^{1,\infty}(\Omega)$. Assumption 2 can also be relaxed to a more general case where $\alpha, \beta \in \mathcal{K}_\infty$ and $\sigma \in \mathcal{L}$ such that

$$\alpha(\|x\|) \le V^*(x) \le \beta(\|x\|)$$

$$DV^*(x) \cdot f(x) \le -\rho(x)$$

and ρ is a positive definite function such that $\rho(x) \ge \alpha(\|x\|)\sigma(\|x\|)$. Here $\mathcal{K}_\infty := \{\alpha : \mathbb{R}_+ \to \mathbb{R}_+ : \alpha(0) = 0, \lim_{s\to\infty} \alpha(s) = \infty$, α is strictly increasing and continuous$\}$ and $\mathcal{L} := \{\sigma : \mathbb{R}_+ \to \mathbb{R}_{++} : \lim_{s\to\infty} \sigma(s) = 0$, σ is strictly decreasing and continuous$\}$ [53]. For simplicity, we consider (2.5a) and (2.5b) in this paper, which hold in a large class of real-world problems. Assumption 2 also indicates that V^* is a Lyapunov function and 0 is the global asymptotically stable equilibrium in Ω.

We will need the following result on the universal approximation power of neural networks.

Lemma 1 (Theorem 4.9 [40])**.** *Let $d \in \mathbb{N}$, $B > 0$, and $\|g\|_{W^{1,\infty}(\Omega)} \le B$. For all $\epsilon \in (0, 1/2)$, there exists a feed-forward neural network ϕ_θ with network parameter θ and RePU activation* *such that*

$$\|\phi_\theta - g\|_{W^{1,\infty}(\Omega)} \leq \epsilon,$$

We now show a sort of universal approximation for the general Lyapunov-Net.

Lemma 2. *For any $\delta > 0$, the set of Lyapunov-Nets of form $V_\theta(x) = |\phi_\theta(x) - \phi_\theta(0)| + \bar{\alpha}\|x\|$ with RePU network ϕ_θ and bounded weights $\theta \in [-1, 1]^m$, where m is the number of trainable parameters in ϕ_θ, is dense in the function space $\mathcal{S} := \{\bar{g}(x) + \alpha\|x\| : \bar{g} \in C^1(\Omega; \mathbb{R}_+),\ \bar{g}(0) = 0\}$ under the $W^{1,\infty}(\Omega_\delta)$ norm so long as $\bar{\alpha} \in (0, \alpha)$.*

Proof. Let $h \in \mathcal{S}$ be arbitrary, where $h(x) = \bar{g}(x) + \alpha\|x\|$ for some $\bar{g}$ as characterized by the definition of $\mathcal{S}$. Further let $\delta > 0$ and $\epsilon > 0$. Let $\bar{\alpha} \in (0, \alpha)$ and define $g(x) := \bar{g}(x) + (\alpha - \bar{\alpha})\|x\|$. We will use $V_\theta(x)$ to approximate $h(x) = g(x) + \bar{\alpha}\|x\|$. We note that for all $x \in \Omega_\delta$ we have $g(x) \geq a := \inf_{x \in \Omega_\delta} g(x) \geq (\alpha - \bar{\alpha})\delta > 0$. From Lemma 1 we know there exists a RePU neural network $\phi_\theta : \mathbb{R}^d \to \mathbb{R}$ such that $\|\phi_\theta - g\|_{W^{1,\infty}(\Omega)} \leq \min(a/2, \epsilon/2)$. Noting $g(0) = \bar{g}(0) = 0$ we find that $|\phi_\theta(0)| = |\phi_\theta(0) - g(0)| \leq a/2$. On the other hand, for all $x \in \Omega$, there is $|\phi_\theta(x) - g(x)| \leq a/2$ which implies $\phi_\theta(x) \geq g(x) - a/2 \geq a/2$. Hence $\phi_\theta(x) - \phi_\theta(0) \geq 0$ for all $x \in \Omega_\delta$. Furthermore, for all $x \in \Omega$, there is

$$\begin{aligned}
|V_\theta(x) - h(x)| &= |\phi_\theta(x) - \phi_\theta(0) - g(x) + g(0)| \\
&\leq |\phi_\theta(x) - g(x)| + |\phi_\theta(0) - g(0)| \\
&\leq \epsilon.
\end{aligned}$$

From this we find

$$\begin{aligned}|D_i V_\theta(x) - D_i h(x)| &= |D_i(\phi(x) - \phi(0)) - D_i g(x)| \\ &\leq |D_i \phi(x) - D_i g(x)| \\ &\leq \frac{\epsilon}{2},\end{aligned}$$

for all $x \in \Omega_\delta$ a.e. Thus

$$\|V_\theta - h\|_{W^{1,\infty}(\Omega_\delta)} \leq \max\left(\epsilon, \frac{\epsilon}{2}\right) = \epsilon.$$

To show the above is true for RePU networks whose weight matrices are bounded we note the following observation: Suppose we have any layer of a RePU network of the form $RePU(A_\ell z_{\ell-1} + b_\ell)$, where $A_\ell \in \mathbb{R}^{w_\ell \times w_{\ell-1}}$ and $b_\ell \in \mathbb{R}^{w_\ell}$ with w_ℓ the width of layer ℓ. If A_ℓ and b_ℓ have max entry L, then both can be decomposed into

$$A_\ell = A_1^{(\ell)} + \cdots + A_N^{(\ell)}, \quad b_\ell = b_1^{(\ell)} + \cdots + b_N^{(\ell)},$$

such that each $A_i^{(\ell)} \in [-1,1]^{w_\ell \times w_{\ell-1}}$ and $b_i^{(\ell)} \in [-1,1]^{w_\ell}$. As RePU activation functions can exactly represent the identity function Id using weights in $[-1,1]$ (see for example Lemma

2.1 [66]), we rewrite the layer $RePU(A_\ell z_{\ell-1} + b_\ell)$ as

$$\begin{aligned} RePU(B_\ell Id(\bar{A} z_{\ell-1} + \bar{b})) &= RePU([A_1^{(\ell)} z_{\ell-1} + b_1^{(\ell)}] \\ &\qquad + \cdots + [A_N^{(\ell)} z_\ell + b_N^{(\ell)}]) \\ &= RePU(A_\ell z_{\ell-1} + b_\ell), \end{aligned}$$

where $N \leq \lceil L \rceil$, $B_\ell = [I_{w_\ell}, I_{w_\ell}, \ldots I_{w_\ell}]$ contains N order w_ℓ identity matrices,

$$\bar{A} = [(A_1^{(\ell)})^T, \ldots, (A_N^{(\ell)})^T]^T,$$

and $\bar{b} = [(b_1^{(\ell)})^T, \ldots, (b_N^{(\ell)})^T]^T$.

Combining the above observation and the proof for general RePU feed-forward networks completes the proof for some bounded weights $\theta \in [-1, 1]^m$. $\square$

In practice, the usefulness of bounding the weights of our network is that once the depth and size of Lyapunov-Net is fixed, we can ensure the Lipschitz constant of DV_θ does not grow unbounded during training. This guarantees a robust verification method through use of this Lipschitz constant to ensure our approximating network is indeed a Lyapunov function after tuning for size and width. We note that while many others [17, 39, 89, 101, 102, 123] consider the needed size of networks to approximate functions in certain spaces, there are much fewer results on Lipschitz bounded networks (See for example [51, 110] and references therein). As network size bounds usually do not reflect the practical reality of many applications of Lyapunov-Net, we will not exploit such analysis here.

From here on we assume $\Theta = [-1, 1]^m$ for some $m \in \mathbb{N}$.

Lemma 3. *For any $\varepsilon \in (0, \frac{\gamma}{\sqrt{d}L_f})$ and $\delta > 0$, there exists $\theta^* \in \Theta$ such that V_{θ^*} satisfies* (2.5a) *and*

$$DV_{\theta^*}(x) \cdot f(x) \leq -a_{\gamma,\varepsilon}\|x\|, \quad \forall x \in \Omega_\delta. \tag{2.6}$$

where $a_{\gamma,\varepsilon} := \gamma - \varepsilon L_f > 0$.

Proof. By Lemma 2, the set of Lyapunov-Nets, denoted by $\mathcal{V}_\Theta^\epsilon$, forms an ε-net of $\mathcal{S}$ in the $W^{1,\infty}(\Omega_\delta)$ sense. That is, for any $h \in \mathcal{S}$, there exists $\theta \in \Theta$ such that $V_\theta \in \mathcal{V}_\Theta^\varepsilon$ and $\|V_\theta - h\|_{W^{1,\infty}(\Omega_\delta)} < \varepsilon$. Since $V^* \in \mathcal{S}$, we know there exists $\theta^* \in \Theta$ such that $V_{\theta^*} \in \mathcal{V}_\Theta^\varepsilon$ and $\|V_{\theta^*} - V^*\|_{W^{1,\infty}(\Omega_\delta)} < \varepsilon$. Therefore

$$\begin{aligned}
DV_{\theta^*}(x) \cdot f(x) &= DV^*(x) \cdot f(x) + (DV^*(x) - DV_{\theta^*}(x)) \cdot f(x) \\
&\leq -\gamma\|x\| + \|DV^*(x) - DV_\theta(x)\| \cdot \|f(x)\| \\
&\leq -\gamma\|x\| + \varepsilon\sqrt{d}L_f\|x\| \\
&= -a_{\gamma,\varepsilon}\|x\|
\end{aligned}$$

for all $x \in \Omega_\delta$, where we used the facts that $\|V_\theta - V^*\|_{W^{1,\infty}(\Omega_\delta)} < \varepsilon$ and f is L_f-Lipschitz in Ω, and $f(0) = 0$ to obtain the last inequality. □

With the set of bounded weights Θ and fixed network V_θ, we know there exists $M > 0$ such that $DV_\theta(\cdot) \cdot f(\cdot)$ is M-Lipschitz on Ω for all $\theta \in \Theta$. Moreover, as we discussed above, minimizing an empirical loss function based on finitely many sample collocation points cannot guarantee the result is a Lyapunov function. Hence we provide a theoretical guarantee for finding an δ-accurate Lyapunov function. The following Lemma will be needed:

Lemma 4. *For any $\delta, c \in (0,1)$, there exists $N = N(\delta, c) \in \mathbb{N}$, such that for some N sampling points $x^{(i)} \in \Omega_\delta$ where $i = 1, \dots, N$, we have $\Omega_\delta \subset \cup_{i=1}^{N} B(x^{(i)}; c\|x^{(i)}\|)$.*

Proof. Define the radi $r_1, \dots, r_k, r_{k+1}$ such that $\delta/\sqrt{d} = r_1 < r_2 < \dots < r_k \leq 1 < r_{k+1}$ and $r_{i+1} - r_i = cr_i$. The above relation requires $r_i = (1+c)^{i-1}\delta/\sqrt{d}$. Thus $k = \lfloor -\ln(\delta/\sqrt{d})/\ln(1+c) \rfloor + 1$. Now define $I = \{\pm r_j : j \in [k]\}$ where $[k] := \{1, \dots, k\}$ and let $X = \{x = (x_1, \dots, x_d) \in \Omega : x_i \in I,\, i \in [d]\}$. Hence we set $N = |X| = (2k)^d$.

Let $y \in \Omega_\delta$ be arbitrary. Then for each component y_j of y we know there exists integer $k_j \in [k]$ such that $|y_j| \in [r_{k_j}, r_{k_j+1}]$. Therefore, we can choose the grid point $x = (\text{sign}(y_1)r_{k_1}, \dots, \text{sign}(y_d)r_{k_d}) \in X$ for which

$$\begin{aligned}
\|x - y\|^2 &= (r_{k_1} - |y_1|)^2 + \dots + (r_{k_d} - |y_d|)^2 \\
&\leq (r_{k_1} - r_{k_1+1})^2 + \dots + (r_{k_d} - r_{k_d+1})^2 \\
&= (cr_{k_1})^2 + \dots + (cr_{k_d})^2 \\
&\leq c^2\|x\|^2.
\end{aligned}$$

Thus $\|x - y\| \leq c\|x\|$. As $y \in \Omega_\delta$ is arbitrary, we know $\Omega_\delta \subset \cup_{i=1}^{N} B(x^{(i)}; c\|x^{(i)}\|)$. □

Theorem 2. *For any $\delta \in (0,1)$ and $a_{\gamma,\epsilon}$ as defined in Lemma 3, choose any $\bar{\gamma} \in (0, a_{\gamma,\epsilon})$ and $0 < c < \frac{\bar{\gamma}}{M} < 1$ (if $\frac{\bar{\gamma}}{M} \geq 1$, then choose any $c \in (0,1)$). Let N and $X := \{x^{(i)} : i \in [N]\}$ be given as in Lemma 4 and the empirical risk function $\hat{\ell}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(DV_\theta(x^{(i)}) \cdot f(x^{(i)}) + \bar{\gamma}\|x^{(i)}\|)_+$. Then minimizer of $\hat{\ell}$ must exist and achieve minimum function value 0. Moreover,* *for any minimizer $\hat{\theta}$ of $\hat{\ell}(\cdot)$, the corresponding $V_{\hat{\theta}}$ is an δ-accurate Lyapunov function of f on Ω.*

Proof. By Lemma 3, we know there exists $\theta^* \in \Theta$ such that V_{θ^*} is positive definite and $DV_{\theta^*}(x) \cdot f(x) \leq -a_{\gamma,\epsilon}\|x\| \leq -\bar{\gamma}\|x\|$ for all $x \in \Omega_\delta$. Hence $\hat{\ell}(\theta^*) = 0$. Given that $\hat{\ell}(\theta)$ is nonnegative for any θ, we know the minimum value of $\hat{\ell}$ on Θ is 0. Let $\hat{\theta}$ be any minimizer of $\hat{\ell}$, then there is also $\hat{\ell}(\hat{\theta}) = 0$, which implies $DV_{\hat{\theta}}(x) \cdot f(x) \leq -\bar{\gamma}\|x\|$ for all $x \in X$.

Now for any $y \in \Omega_\delta$, there exists $x \in X$, such that $y \in B(x; c\|x\|)$ due to Lemma 4. Hence

$$\begin{aligned} DV_{\hat{\theta}}(y) \cdot f(y) &\leq DV_{\hat{\theta}}(x) \cdot f(x) + |DV_{\hat{\theta}}(y) \cdot f(y) - DV_{\hat{\theta}}(x) \cdot f(x)| \\ &\leq -\bar{\gamma}\|x\| + M\|y - x\| \\ &\leq -\bar{\gamma}\|x\| + Mc\|x\| \\ &\leq -(\bar{\gamma} - Mc)\delta. \end{aligned}$$

Since y is arbitrary, we know $V_{\hat{\theta}}$ is a δ-accurate Lyapunov function. □

2.3.4 Application to control and others

In light of the power of Lyapunov functions, we can employ the proposed Lyapunov-Net to many control problems of nonlinear dynamical systems in high-dimension. In this subsection, we instantiate one of such applications of Lyapunov-Net to approximate control Lyapunov function.

Consider a nonlinear control problem $x' = f(x, u)$ where $u : \mathbb{R}^d \to \mathbb{R}^n$ (n is the dimension of the control variable at each x) is an unknown state-dependent control in order to steer the state x from any initial to the equilibrium state 0. To this end, we parameterize the control as a deep neural network $u_\eta : \mathbb{R}^d \to \mathbb{R}^n$ (a neural network with input dimension d and output dimension n) where η represents the network parameters of u_η. In practice, the control variable is often restricted to a compact set in $\mathbb{R}^d$ due to physical constraints. This can be easily implemented in a neural network setting. For example, if the magnitude of the control is required to be in $[-\beta, \beta]$ componentwisely, then we can simply apply $\beta \cdot \tanh(\cdot)$ to the last, output layer of u_η.

Once the network structure u_η is determined, we can define the risk of the control-Lyapunov function (CLF):

$$\ell_{\text{CLF}}(\theta, \eta) := \frac{1}{|\Omega|} \int_\Omega (DV_\theta(x) \cdot f(x, u_\eta(x)) + \gamma \|x\|)_+^2 \, dx. \tag{2.7}$$

Minimizing (2.7) yields the optimal parameters θ and η. In practice, we again approximate $\ell_{\text{CLF}}(\theta, \eta)$ by its empirical expectation $\hat{\ell}_{\text{CLF}}(\theta, \eta)$ at sampled points in Ω, as an analogue to $\ell(\theta)$ versus $\hat{\ell}(\theta)$ above. Then the minimization can be implemented by alternately updating θ and η using (stochastic) gradient descent on the empirical risk function $\hat{\ell}_{\text{CLF}}$. Similar as (2.3), we have a single term in the loss function in (2.7), which does not have hyper-parameters to tune and the training can be done efficiently.

2.4 Numerical Experiments

2.4.1 Experiment setting

In this section, we demonstrate the effectiveness of the proposed method through a number of numerical experiments. In our experiments, the value of $\bar{\alpha}$ used in (2.2) and the depth and size of ϕ_θ used in V_θ for the three test problems are summarized in Table 2.1. We minimize the empirical risk function $\hat{\ell}$ using the Adam Optimizer with learning rate 0.005 and $\beta_1 = 0.9$, $\beta_2 = 0.999$ and Xavier initializer. In all tests, we iterate until the associated risk of (2.4) is below a prescribed tolerance of 10^{-4}. We use a sample size N (values shown in Table 2.1), i.e., the number of sampled points in Ω in (2.4), such that the associated risk reduces reasonably fast while maintaining good uniform results over the domain. We note these points are drawn using a uniform sampling method for simplicity. We finally note in all experiments the weights remain bounded in $[-1, 1]^m$ as noted in our theory.

All the implementations and experiments are performed using PyTorch in Python 3.9 in Windows 10 OS on a desktop computer with an AMD Ryzen 7 3800X 8-Core Processor at 3.90 GHz, 16 GB of system memory, and an Nvidia GeForce RTX 2080 Super GPU with 16 GB of graphics memory. The number of iterations needed to reach our stopping criteria in (2.4) and training time (in seconds) for the three tests are also given in Table 2.1. As discussed in Chapter 1, how the width and depth of a neural network are related to approximation power is an area of active research. In our experiments, we manually selected the width and depth as shown in Table 2.1 which yielded good results.

Table 2.1 Network parameter setting and training time in the tests.

Test Problem	Time	Iter.	N	Depth/Width	$\bar{\alpha}$
Curve Tracking	0.5 s	2	100K	3/10	0.5
10d Synthetic DS	1.5 s	2	200K	3/20	0.01
30d Synthetic DS	2.5 s	15	400K	5/20	0.01

2.4.2 Experimental results

To demonstrate the effectiveness of the proposed method, we apply Lyapunov-Net (2.2) to three test problems: a two-dimensional (2d) nonlinear system from the curve-tracking application [83], a 10d and 30d synthetic dynamical system (DS) from [36].

2.4.2.1 2-dimensional DS in curve tracking

We apply our method to find the Lyapunov function for a 2d nonlinear DS in a curve-tracking problem [83]. The DS of $x = (\rho, \varphi)$ is given by

$$\dot{\rho} = -\sin(\varphi), \tag{2.8a}$$

$$\dot{\varphi} = (\rho - \rho_0)\cos(\varphi) - \mu \sin(\varphi) + e. \tag{2.8b}$$

We use the following constants in our experiments: $e = 0.15$, $\rho_0 = 1$, and $\mu = 6.42$ from [83] as well as RePU activation.

The problem domain was mapped to Cartesian coordinates around the critical point $x^* = (1, 0)$, and converted back when the training is done. The result shows that after only 2 iterations the positive definiteness and negative-orbital-derivative conditions are met. We used RePU activation which is consistent with the theoretical results provided in Section

III-C. However, these results still hold by readily modifying the proofs if the common tanh activation is used. We have also tested tanh activation in Lyapunov-Net and obtained a similar performance.

2.4.2.2 10d Synthetic DS

We consider a 10d synthetic DS from [36], which is a vector field defined on $[-1, 1]^{10}$ and has an equilibrium at 0. The iteration number and computation time needed for training are shown in Table 2.1.

2.4.2.3 30d Synthetic DS

We consider the same system as the 10d Synthetic DS but concatenate the function f three times to get a 30d problem. Figure 2.1 graphed the approximated Lyapunov function V_θ (top solid) and $DV_\theta \cdot f$ (bottom wire) in the (x_2, x_8) and (x_{10}, x_{13}) planes, using RePU activation. These plots show that Lyapunov-Net can effectively approximate Lyapunov functions in such high-dimensional problem. We shall also use this example to compare the convergence speeds to the Neural Network structures of [16, 36] in the following section.

2.4.3 Comparison with existing DL methods

To demonstrate the significant improvement of Lyapunov-Net over existing DL methods in approximation efficiency, we compare the proposed Lyapunov-Net to two recent approaches [16, 36] that also use deep neural networks to approximate Lyapunov functions in continuous DS setting. Specifically, we use the 30d synthetic DS described above as the test problem in this comparison. Both [16, 36] employ generic deep network structure of V_θ, and thus

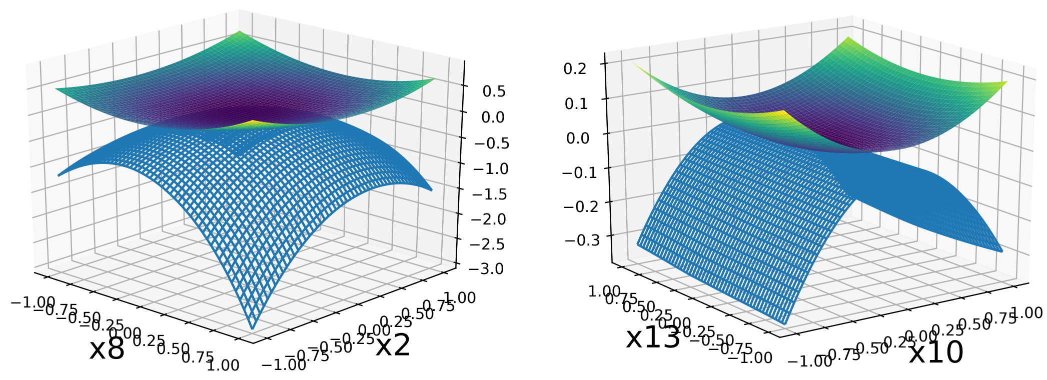


Figure 2.1 Plot of the learned Lyapunov-Net V_θ (top solid) and $DV_\theta \cdot f$ (bottom wire) in the (x_2, x_8)-plane (left) and (x_{10}, x_{13})-plane (right) for the 30d synthetic DS.

require additional terms in their risk functions to enforce positive definiteness of the networks. Specifically, the following risk function is used from [36]:

$$\begin{aligned}\hat{\ell}_1(\theta) =& \tfrac{1}{N}\sum_{i=1}^{N}\Big(\big(DV_\theta^{DL}(x_i)\cdot f(x_i) + \|x_i\|^2\big)_+^2 \\ &+ \big(20\|x_i\|^2 - V_\theta^{DL}(x_i)\big)_+^2 + \big(V_\theta^{DL}(x_i) - 0.2\|x_i\|^2\big)_+^2 \Big),\end{aligned} \tag{2.9}$$

which aims at an approximate Lyapunov function V_θ^{DL} satisfying $0.2\|x\|^2 \leq V_\theta^{DL}(x) \leq 20\|x\|^2$ and $DV_\theta^{DL}(x)\cdot f(x) \leq -\|x\|^2$ for all x. In [16], the following risk function is used:

$$\hat{\ell}_2(\theta) = V_\theta^{NL}(0)^2 + \tfrac{1}{N}\sum_{i=1}^{N}[(DV_\theta^{NL}(x_i)\cdot f(x_i))_+ + (-V_\theta^{NL}(x_i))_+], \tag{2.10}$$

which aims at an approximate Lyapunov function V_θ^{NL} such that $V_\theta^{NL}(0) = 0$, $V_\theta^{NL}(x) \geq 0$ and $DV_\theta^{NL}(x)\cdot f(x) \leq 0$ for all x. The activation functions are set to softmax in (2.9) as suggested in [36] and tanh in (2.10) as suggested in [16]. We shall label these models as Deep Lyapunov (DL) and Neural Lyapunov (NL) respectively. For Lyapunov-Net we use (2.4) as it already satisfies the positive definiteness condition. We again do not impose any structural information of the problem into our training, and thus all test methods recognize the DS as a generic 30d system for the sake of a fair comparison.

We use the value of the less stringent empirical risk function $\hat{\ell}_2$ with $N = 400,000$ defined in (2.10) as a metric to evaluate all three methods. Specifically, we plot the values of $\hat{\ell}_2$ (in log scale) versus iteration number and wall-clock training time (in seconds) in Figure 2.2 using the same learning rate for all methods.

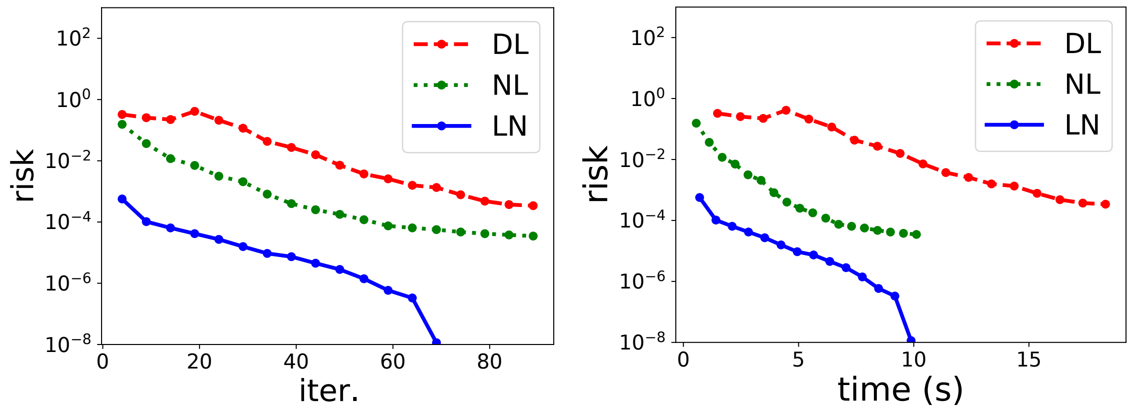


Figure 2.2 The value of empirical risk function $\hat{\ell}_2$ (in log scale) defined in (2.10) versus iteration number (left) and wall-clock training time in seconds (right) for the three network and training settings: DL [36] (red dashed), NL [16] (green dotted), and Lyapunov-Net denoted LN (blue solid). Note that the risk of Lyapunov-Net often hits 0, the most desired risk function value, explaining the dropoff.

We see in Figure 2.2 that Lyapunov-Net (LN) has risk value decaying much faster than the other methods. Further, Lyapunov-Net training does not need hyper-parameter tuning to achieve this speed, whereas DL and NL require careful and tedious tuning to balance the different terms in the risk function in order to achieve satisfactory results as shown in Figure 2.2. This highlights the efficiency and simplicity of Lyapunov-Net in finding the desired Lyapunov functions.

We note that the performance of all methods can be further improved using additional structural information as discussed in [36] and falsification techniques in training as in [16]. We leave these improvements to future investigations.

2.4.4 Application in Control

In this test, we compare Lyapunov-Net and SOS/SDP methods in the problem of estimating the Region of Attraction (ROA) for the classical inverted pendulum control problem. We shall show how the Lyapunov-Net framework allows simultaneous training of the control policy and outperforms SOS/SDP methods by producing a larger ROA.

The inverted pendulum is an often considered problem in control theory see [16, 98]. For this problem we have dynamics $x = (\theta, \dot{\theta})$ governed by $ml^2\ddot{\theta} = mgl\sin\theta - \beta\dot{\theta} + u$, where $u = u(x)$ is the control. We use $g = 9.82$, $l = 0.5$, $m = 0.15$, and $\beta = 0.1$ for our experiments. We shall use this model to compare SOS/SDP type methods to Lyapunov-Net when it comes to estimating the ROA of this problem using the Lyapunov candidate function they both find within respective valid regions.

We adjust our model slightly so that we learn a u policy alongside V_θ. This is done by implementing a single layer neural network $\bar{u}$ which performs a simple linear transform for some learned matrix A. We use such a simple control so that the control can be used in the SOS/SDP algorithm. In the absence of such a comparison, we could make $\bar{u}$ a much more complex and further improved control policy. The training method is the same for both networks.

In Figure 2.3, we use tanh as activation in V_θ because while the error bounds are similar

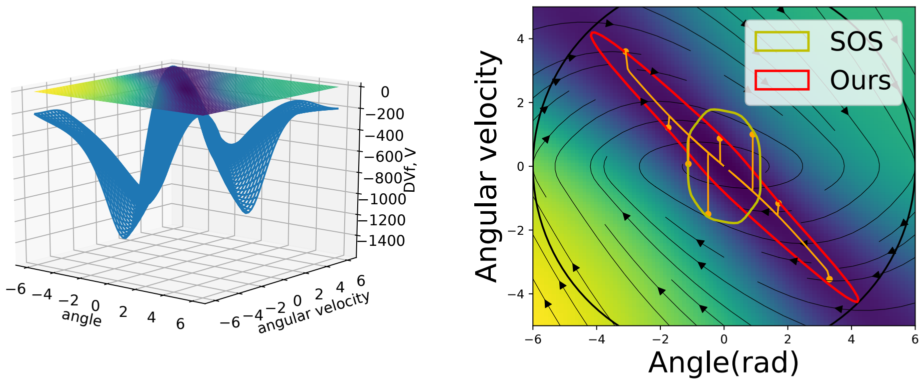


Figure 2.3 Left: learned Lyapunov-Net V_θ (top solid) and $DV_\theta \cdot f$ (bottom wire) for the 2d inverted pendulum in phase and angular velocity space. Right: Comparison of ROA for standard SOS/SDP methods and for Lyapunov-Net with orange sample trajectories.

with RePU, tanh has slightly better performance in this case. Once trained the u-policy found is used for the SOS type method. As SOS methods are only applicable to polynomial systems, we apply a Taylor approximation to the dynamics of the system and then compute a sixth-order polynomial candidate using the standard SOS/SDP approach. We finally compute the ROA determined by both algorithms using the level-sets of the functions over the valid regions they found. We find that the area of the region found by Lyapunov-Net is larger than that found by SOS. This result is plotted in Figure 2.3 where the orange paths are a few sample trajectories. We note the similarity of our results to [16], who in addition to the above, found examples where SOS methods produced incorrect ROA regions on this same problem.

2.5 Conclusions

In this part, we have constructed a versatile deep neural network architecture called Lyapunov-Net to approximate Lyapunov function for general high-dimensional dynamical systems. We provided theoretical justifications on approximation power and certificate guarantees of Lyapunov-Nets. Applications to control Lyapunov functions are also considered. We demonstrated the effectiveness of our method on several test problems. The Lyapunov-Net framework developed in the present work is expected to be applicable to a much broader range of control and stability problems.

PART 3
NEURAL NETWORK CONTROL FOR HIGH-DIMENSIONAL EVOLUTION PDES

We now switch gears and consider applications of DNNs to another high-dimensional problem. Instead of concerning ourselves with approximating functions as in Lyapunov-net, we will develop a novel approach for finding solution operators for PDEs. This is an infinite-dimensional problem, but we shall discuss in this and the next part how to reduce this to a finite-dimensional optimization problem in a similar vein as what we did when developing Lyapunov-net. We call the forthcoming approach "Neural Control" and will introduce the relevant literature on the topic as we proceed. As Partial differential equations (PDEs) are ubiquitous in modeling and are vital in numerous applications from finance, engineering, and science [30] obtaining approximations of their solutions is of vital importance.

3.1 Introduction

DNNs have emerged as powerful tools for solving high-dimensional PDEs [7, 21, 28, 43, 44, 46, 62, 94]. For example, in [7, 21, 28, 94, 124], the solution of a given PDE is parameterized as a DNN, and the network parameters are trained to minimize potential violations (in various definitions) to the PDE. These methods have shown numerous successes in solving a large variety of PDEs empirically. These methods aim at solving specific instances of PDEs, and as a consequence, they need to start from scratch for the same PDE whenever the initial and/or boundary value changes.

To solve the problem of finding PDEs for more than one instance, recent studies have attempeted to find solution operators of PDEs [67, 76]. These methods aim at finding the map from the problem's parameters to the corresponding solution. Finding solution operators has substantial applications as the same PDE may need to run many times with different initial or boundary value configurations. However, existing methods fall short in tackling high-dimensional problems as many require spatial discretization to represent the solution operators using DNNs.

In the next two chapters, we will propose a new approach for finding solution operators of high-dimensional evolution PDEs. For a given PDE, we first parameterize its solution as a DNN, whose parameters denoted as θ are to be determined. Then we seek to find a vector field on the parameter space that describes how θ evolves in time. This vector field essentially acts as a controller on the parameter space, steering the parameters so that the induced DNN evolves and approximates the PDE solution for all time. Once such a vector field is found, we can easily change the initial conditions of the PDE by simply starting at a new point in the parameter space. Then we follow the control vector field to find the parameters trajectory that gives an approximation of the time-evolving solution. Thus, different initial conditions can be considered for the same PDE without solving it repeatedly. The new contributions of this part to the existing literature can be summarized as follows:

1. We develop a new computational framework to find the solution operator of any given initial value problem (IVP) defined by high-dimensional nonlinear evolution PDEs. This framework is purely based on the evolution PDE itself and does not require any solutions of the PDE for training. Once we find the solution operator, we can quickly

compute solutions of the PDE with any initial value at a low computational cost.

2. We provide comprehensive theoretical analysis to establish error bounds for the proposed method when solving linear PDEs and some special nonlinear PDEs.

3. We conduct a series of numerical experiments to demonstrate the effectiveness of the proposed method in solving a variety of linear and nonlinear PDEs.

3.2 Background

3.2.1 Classical methods for solving PDEs

Classical numerical methods for solving PDEs, such as finite difference [113] and finite element methods [55], discretize the spatial domain using mesh or triangulation. These methods convert a PDE to its discrete counterpart, which is a system of algebraic equations with finite number of unknowns, and solve the system to obtain an approximate solution on the pre-selected grid points [3, 29, 91, 112]. These methods have been significantly advanced in the past decades, and they can handle complicated situations such as irregular domains. However, they severely suffer from the "curse of dimensionality" when applied to high-dimensional problems.

3.2.2 Neural network based methods for solving PDEs

Early attempts using shallow neural networks to solve PDEs can be seen in [24, 62, 63, 64]. Related, in recent years DNNs have demonstrated striking power in solving PDEs through various approaches [7, 9, 28, 86, 94, 104, 121, 124]. DNNs have demonstrated extraordinary potential in solving many high-dimensional nonlinear PDEs, which were considered computationally intractable using classical methods. For example, a variety of DNN based methods have been proposed based on the strong form [9, 24, 58, 80, 84, 87, 88, 94, 95], the variational form [28], and the weak form [7, 124] of PDEs. They are considered with adaptive collocation strategy [5], adversarial inference procedure [122], oscillatory solutions [14], and multiscale methods [15, 73, 117]. Improvements of these methods with adaptive activation functions [54], networks structures [37, 38, 49], boundary conditions [25, 79], structure probing [49], as well as their convergence [78, 103], are also studied. Further, some methods can also solve inverse problems, such as parameter identification.

For a class of high-dimensional PDEs with equivalent backward stochastic differential equations (SDEs) that arise from Feynman-Kac theory, deep learning methods have been used leveraging the Feynman-Kac theorem [8, 27, 33, 43, 44, 45, 50, 52, 90]. These methods are shown to be good even in very high dimensions (¿200) [44, 50, 90], however, they are limited to solving the special type of evolution equations whose generator function has a corresponding SDE.

For evolution PDEs, parameter evolution algorithms [4, 12, 26] have also been considered. These methods parameterize the PDE solution as a neural network [12, 26] or an adaptively chosen ansatz as discussed in [4]. In these methods, the parameters are evolved forward in time through a time marching scheme, where at each step a linear system [12, 26] or a constrained optimization problem [4] needs to be solved.

3.2.3 Learning solution operator of PDEs

The aforementioned methods aim at solving a *specific* instance of a given PDE, and they need to be rerun from scratch when any part of the problem configuration (e.g., initial value, boundary value, problem domain) changes. In contrast, the solution operator of a PDE directly maps a problem configuration to its corresponding solution. To this end, several methods have been proposed to approximate Green's functions for some linear PDEs [10, 11, 71, 111], as solutions to such PDEs have explicit expression based on their Green's functions. However, this approach only applies to a small class of linear PDEs whose solution can be represented using Green's functions. Moreover, Green's functions have singularities and it requires special care to approximate them using neural networks. For example, rational functions are used as activation functions of DNNs to address singularities in [10]. In [11], the singularities are represented with the help of fundamental solutions.

For general nonlinear PDEs, DNNs have been used for operator approximation and meta-learning for PDEs [41, 67, 76, 77, 82, 97, 118, 119]. For example, the work [41] considers solving parametric PDEs in low-dimension ($d \leq 3$ for the examples in the paper). Their method requires discretization of the PDE system and needs to be supplied by many full-order solutions for different combinations of time discretization points and parameter selections for their network training. Their method applies proper orthogonal decomposition to these solutions to obtain a reduced-order basis to construct solutions for new problems. Another approach is DeepONets [76, 77, 118] which seek to approximate solution mappings by using a special neural network consisting of a "branch" and "trunk" network. FNOs

[67, 119] use Fourier transforms to map a neural network to a low dimensional space and then back to the solution. In addition, several works that apply spatial discretization of the problem or transform domains and use convolutional neural networks (CNNs) [42, 96, 125] or graph neural networks (GNNs) [61, 68, 75]. Interested readers may also refer to generalizations and extensions of these methods in [13, 19, 20, 31, 60, 68, 77, 82, 87]. A key similarity of all these methods is they require certain domain discretization and often a large number of labeled pairs of IVP initial conditions (or PDE parameters) and the corresponding solution, obtained through other methods, for training. This limits their applicability to high-dimensional problems where such training data is unavailable, or the mesh is prohibitive to generate due to curse-of-dimensionality.

3.2.4 Differences between the proposed approach and existing ones

Different from all existing approaches, in this chapter we propose to approximate solution operators of evolution PDEs through controlling the parameters of the DNN in a relevant parameter space induced by DNNs' architecture. Unlike the existing solution operator approximation methods (e.g., DeepONet [76] and FNO [67]) which seek to directly approximate the infinite-dimensional operator, our approach is based on the relation between evolving solutions and their projected trajectories in the parameter space. This leads us to convert the problem of finding a solution operator over an infinite-dimensional function space into a control vector field optimization problem over a finite-dimensional parameter space. As a result, the problem of solving an evolution PDE in continuous space is reduced to numerically solving a system of ODEs, which can be done accurately with very low computational complexity. Moreover, our approach does not require spatial discretization of the problem domain in any problem nor does it need any basis function representation throughout problem formulation and computation. We provide mathematical insights into the relevant parameter submanifold and its tangent spaces and establish their connection to the finite-dimensional parameter space. These new insights led us to the proposed approach which approximates solution operators of PDEs by controlling network parameters in the parameter space. These new features also enable our approach to solve evolution PDEs in high-dimensional cases. This is a significant advantage over existing operator learning methods such as DeepONet or FNOs as their spatial discretization schemes, which are used to generate the training data, hinder their application to high-dimensional cases.

3.3 Proposed Method

As discussed, we parameterize solutions u of our PDE as a DNN, which is denoted by u_θ with parameters θ, i.e., u_θ is a parametric function determined by the value of its finite-dimensional parameters θ, and u_θ is used to approximate u. We identify this finite-dimensional parameter space by restricting ourselves to a subset $\Theta \subset \mathbb{R}^m$ such that all the parameters θ we care about belong to Θ and we define the submanifold $\mathcal{M}$ of functions by

$$\mathcal{M} := \{u_\theta : \Omega \to \mathbb{R} \mid \theta \in \Theta\}. \tag{3.1}$$

As we can see, u_θ defines a mapping from the parameter space Θ to the submanifold $\mathcal{M}$ of the infinite-dimensional function space. We call $\mathcal{M}$ the *parameter submanifold* determined by u_θ.

To find the solution operator, we propose to build a control vector field V in the parameter space Θ where θ resides. Then the solution operator can be implemented as a fast numerical solver of the ODE defined by V. More precisely, to approximate a time-evolving function $u^*(\cdot, t)$, e.g., the solution of an evolution PDE, over time horizon $[0, T]$ using the DNN u_θ, we need to find a trajectory $\{\theta(t) \in \Theta \,|\, 0 \le t \le T\}$ in the parameter space Θ so that $u_{\theta(t)}(\cdot)$ is close to $u^*(\cdot, t)$ in the function space for every $t \in [0, T]$. For example, if we consider $L^2(\Omega)$ as the function space, by closeness we mean $\|u_{\theta(t)} - u^*(\cdot, t)\|_{L^2(\Omega)}$ is small for all t (hereafter we denote $\|\cdot\|_p = \|\cdot\|_{L^p(\Omega)}$ for notation simplicity). Notice that $\{u_{\theta(t)} \,|\, 0 \le t \le T\}$ is a trajectory on $\mathcal{M}$, whereas $u^*(\cdot, t)$ is a trajectory in the full space $L^2(\Omega)$. To see how this relates to a solution operator imagine we first find the parameters $\theta(0)$ such that $u_{\theta(0)}$ approximates some starting function g, then we follow the control vector field V to obtain a trajectory $\{\theta(t) \,|\, 0 \le t \le T\}$ in Θ with low computational cost, which automatically induces a trajectory $u_{\theta(t)}$ to approximate the true solution u of some IVP with the initial value g. We make these ideas concrete in the following subsections.

3.3.1 Problem space and detailed methodology

Let Ω be an open bounded set in $\mathbb{R}^d$ and F a *nonlinear differential operator* of functions $u : \Omega \to \mathbb{R}$ with necessary regularity conditions, we consider the IVP of the evolution PDE defined by F with arbitrary initial value as follows:

$$\begin{cases} \partial_t u(x,t) = F[u](x,t), & x \in \Omega,\ t \in (0,T], \\ u(x,0) = g(x), & x \in \Omega, \end{cases} \tag{3.2}$$

where $T > 0$ is some prescribed terminal time, and $g : \mathbb{R}^d \to \mathbb{R}$ stands for an initial value. For ease of presentation, we assume zero Dirichlet boundary condition $u(x,t) = 0$ for all $x \in \bar{\Omega}$ and $t \in [0,T]$ (for compatibility we henceforth assume $g(x)$ has zero trace on $\partial\Omega$) throughout this chapter. We denote u^g as the solution to the IVP (3.2) with this initial g. The solution operator $\mathcal{S}_F$ of the IVP (3.2) is thus the mapping from the initial g to the solution u^g :

$$\mathcal{S}_F : C^2(\bar{\Omega}) \to C^{2,1}(\bar{\Omega} \times [0,T]), \quad \text{such that} \quad g \mapsto \mathcal{S}_F(g) := u^g, \tag{3.3}$$

where $C^2(\bar{\Omega}) := C(\bar{\Omega}) \cap C^2(\Omega)$ for short. As a reminder, our goal is to find a numerical approximation to $\mathcal{S}_F$. Namely, *we want to find a fast computational scheme $\mathcal{S}_F$ that takes any initial g as input and accurately estimates u^g with low computation complexity.*

For ease of presentation, we use autonomous, second-order nonlinear differential operators $F[u] = F(x, u, \nabla_x u, \nabla_x^2 u)$ as an example and take $\Omega = (0,1)^d$ in (3.2) to describe our main idea below. Extensions to general non-autonomous nonlinear differential operators and PDEs defined on an open bounded set $\Omega \subset \mathbb{R}^d$ with given boundary values will be discussed in Section 3.5.

We are unconcerned with the specific form of u_θ and only assume that $u_\theta(x) = u(x;\theta)$ is C^1 smooth with respect to θ. This is a mild condition satisfied by many common models: if

u_θ is a linear combination of basis functions and θ represents the combination coefficients, then u_θ is linear and hence smooth in θ; and if u_θ is a feedforward DNN then u_θ is smooth in θ as long as all activation functions are smooth. Suppose there exists a trajectory $\{\theta(t) \,|\, 0 \leq t \leq T\}$ in the parameter space Θ such that its corresponding $u_{\theta(t)}$ approximates the solution of the IVP, we must have

$$\begin{cases} \partial_t u_{\theta(t)}(x) = \nabla_\theta u(x;\theta(t)) \cdot \dot{\theta}(t) = F[u_{\theta(t)}](x), & \forall\, x \in \Omega,\ t \in (0,T], \\ u_{\theta(0)}(x) = g(x), & \forall\, x \in \Omega. \end{cases} \tag{3.4}$$

To compute $u_{\theta(t)}$, it is sufficient to find a control vector (velocity) field $V_F : \Theta \to \mathbb{R}^m$, in the sense of $\dot{\theta}(t) = V_F(\theta(t))$, that steers the trajectory $\theta(t)$ along the correct direction starting from the initial $\theta(0)$ satisfying $u_{\theta(0)}(x) = g(x)$.

This observation suggests for the evolution equation in (3.4) to hold, it suffices to find a vector field V_F such that

$$\nabla_\theta u_\theta \cdot V_F(\theta) = F[u_\theta] \tag{3.5}$$

for all $\theta \in \Theta$. It is important to note that V_F only depends on the nonlinear differential operator F of the original evolution PDE, but not any actual initial value g of the IVP. Once some V_F is found we can effectively approximate the solution of the IVP with any initial value g: we first set $\theta(0) = \theta^g$, where θ^g denotes the parameters such that u_{θ^g} fits g, then we numerically solve the following ODE in the parameter space Θ (which can be fast) using

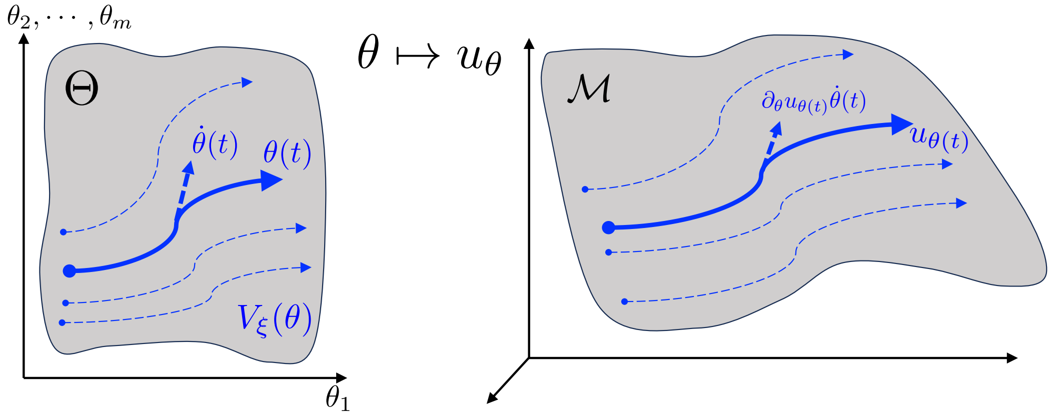

$\theta_2, \cdots, \theta_m$
Θ
$\dot{\theta}(t)$
$\theta(t)$
$V_\xi(\theta)$
θ_1
$\theta \mapsto u_\theta$
$\mathcal{M}$
$\partial_\theta u_{\theta(t)} \dot{\theta}(t)$
$u_{\theta(t)}$

Figure 3.1 Schematic plot of pulling back trajectories (solid and dashed blue curves) in $\mathcal{M} = \{u_\theta : \theta \in \Theta\}$ to trajectories in the parameter space Θ. Here each trajectory in $\mathcal{M}$ represents the reduced-order model (e.g., DNN) $u_{\theta(t)}(\cdot)$ approximating the PDE solution $u^*(t, \cdot)$ starting from a given initial, and it is pulled back to the trajectory $\theta(t)$ (we use $\theta(t) := \theta(t)$ as a trajectory here to avoid confusion with components $\theta_1, \ldots, \theta_m$) in Θ; and V_ξ is a DNN approximating the control vector field V_F in Θ.

the control vector field V_F:

$$\begin{cases} \dot{\theta}(t) = V_F(\theta(t)), \quad \forall\, t \in (0, T], \\ \theta(0) = \theta^g. \end{cases} \tag{3.6}$$

The solution trajectory $\{\theta(t) \,|\, 0 \le t \le T\}$ of the ODE (3.6) induces a path $\{u_{\theta(t)} \,|\, 0 \le t \le T\}$ in $\mathcal{M}$ as an approximation to the solution of the IVP. The computational cost is thus composed of two parts: finding the parameters θ^g of u_θ to fit g and numerically solving the ODE (3.6), both of which are substantially cheaper than solving the IVP (3.2).

The main question is how to get the control vector field V_F in (3.6)? As an explicit form of V_F is unknown, we choose to express V_F in a general parametric form V_ξ with parameters ξ to be determined. Specifically, we propose to set V_ξ as another DNN where ξ represents the set of learnable network parameters in V_ξ. A schematic plot of the pullback mechanism and the control vector field in Θ is provided in Figure 3.1. We call V_ξ the *neural control* field. We learn the parameters ξ by minimizing the following loss function:

$$\ell(\xi) := \int_\Theta \|\nabla_\theta u_\theta \cdot V_\xi(\theta) - F[u_\theta]\|_2^2 \, d\theta. \tag{3.7}$$

In practice, we approximate the integral in ℓ by Monte Carlo integration. We sample K points $\{\theta_k \,|\, k = 1, \ldots, K\}$ uniformly from Θ (here the subscript k in θ_k stands for the kth point among the K points sampled in Θ) and form the empirical loss function

$$\hat{\ell}(\xi) = K^{-1} \cdot \sum_{k=1}^{K} \|\nabla_\theta u_{\theta_k} \cdot V_\xi(\theta_k) - F[u_{\theta_k}]\|_2^2 \tag{3.8}$$

Then we minimize $\hat{\ell}(\xi)$ with respect to ξ, where the L^2 norm is also approximated by Monte Carlo integration on Ω. The training of V_ξ is summarized in Algorithm 1.

Algorithm 1 Training neural control V_ξ

```
Require: Reduced-order model structure $u_\theta$ and parameter set $\Theta$. Control vector field
    structure $V_\xi$. Error tolerance $\varepsilon$.
Ensure: Optimal control parameters $\xi$.
 1: Sample $\{\theta_k\}_{k=1}^K$ uniformly from $\Theta$ and $\{x_n\}_{n=1}^N$ from $\Omega$.
 2: Form empirical loss $\hat{\ell}(\xi)$ as in (3.30).
 3: Minimize $\hat{\ell}$ with respect to $\xi$ using any optimizer (e.g., ADAM or AdaGrad) until $\hat{\ell}(\xi) \leq$
    $\varepsilon$.
```

Once we trained the vector field V_ξ, we can implement the solution operator $\mathcal{S}_F$ in the following two steps: we first find a $\theta(0)$ such that $u_{\theta(0)}$ fits g, i.e., find $\theta(0)$ that minimizes $\|u_\theta - g\|_2$. This can be done by sampling $\{x_n\}_{n=1}^N$ from Ω and minimizing the empirical squared L^2 norm $(1/N) \cdot \sum_{n=1}^N |u_\theta(x_n) - g(x_n)|^2$ with respect to θ. Then we solve the ODE (3.6) using any numerical ODE solver (e.g., Euler, 4th order Runge-Kutta, predictor-corrector) with $\theta(0)$ as the initial value. Both steps can be done efficiently and the total computational cost is substantially lower than that of solving the original IVP (3.2) again. We summarize how neural control solves IVPs in Algorithm 2. Further details on the practical implementation of Algorithm 1 and 2 are discussed in Section 3.4.

Algorithm 2 Implementation of solution operator $\mathcal{S}_F$ of the IVP (3.2) using trained control V_ξ

```
Require: Initial value $g$ and tolerance $\varepsilon_0$. Reduced-order model $u_\theta$ and trained neural control $V_\xi$.
Ensure: Trajectory $\hat{\theta}_t$ such that $u_{\hat{\theta}_t}$ approximate the solution $\mathcal{S}_F[g]$ of the IVP (3.2).
1: Compute initial parameters $\theta(0)$ such that $\|u_{\theta(0)} - g\|_2 \le \varepsilon_0$.
2: Use any ODE solver to compute $\hat{\theta}_t$ to solve (3.6) with approximate field $V_\xi$ and initial $\theta(0)$.
```

3.3.2 Error analysis

In this subsection, we develop an error estimate of the proposed method. We first focus on the error due to projection onto the tangent space $T_{u_\theta}\mathcal{M}$ in the L^2 space in Section 3.3.2.1. Then we establish the solution approximation error for linear and semilinear parabolic PDEs in Section 3.3.2.2. For ease of discussion, we again assume zero Dirichlet boundary condition $u(x,t) = 0$ for all $x \in \bar{\Omega}$ and $t \in [0,T]$, and we let $\Omega = (0,1)^d \subset \mathbb{R}^d$ be the unit open cube in $\mathbb{R}^d$ and Θ some open bounded set in $\mathbb{R}^m$ (note that our analysis below applies as long as Ω is open and bounded). We let $F[u] := F(u, \nabla u, \nabla^2 u)$ be a nonlinear differential operator with necessary regularity conditions to be specified later and allows for a unique solution to the PDE for each initial. Additional requirements on the regularity of u_θ will be given when needed.

3.3.2.1 Approximation error of control vector field

We first investigate the main source of error when using a reduced-order model to approximate the time-evolving solution of the given PDE. We show that this error is due to the imperfect representation of $F[u_\theta]$ using $\nabla_\theta u_\theta$ in (3.5). Specifically, due to the approximation of reduced-order models, $T_{u_\theta}\mathcal{M}$ is only a finite-dimensional subspace of L^2, and thus we can only approximate the projection of $F[u_\theta]$ onto this tangent space. We will need the following assumptions on the regularity of u_θ and F.

Assumption 3. *The reduced-order model $u_\theta(\cdot) \in C^3(\Omega) \cap C(\bar{\Omega})$ for every $\theta \in \bar{\Theta}$ and $u(x;\cdot) \in C^2(\Theta) \cap C(\bar{\Theta})$. Moreover, there exists $L > 0$ such that for all $\theta \in \bar{\Theta}$*

$$F[u_\theta] \in \mathcal{F}^L := \{f \in C^1(\Omega) \cap C(\bar{\Omega}) : \|f\|_\infty \leq L,\ \|\nabla f\|_\infty \leq L\}. \tag{3.9}$$

Assumption 3 provides some sufficient regularity conditions on the reduced-order model u_θ and boundedness of $F[u_\theta]$ and its gradient to be used in our error estimates. Notice that we consider F as second-order differential operator here and therefore the assumption $u_\theta \in C^3(\Omega)$ ensures that $u_\theta(x), \nabla u_\theta(x), \nabla^2 u_\theta(x)$ are all sufficiently smooth. The regularity condition on F in Assumption 3 requires that the mapping $F[u_\theta](x)$ is a C^1 function and have magnitudes and gradients bounded by L over $\bar{\Omega}$. These assumptions are generally mild as we will use reduced-order models smooth in (x, θ), e.g., a DNN with smooth activation functions, and the operator F is sufficiently regular.

Assumption 4. *For any $\bar{\varepsilon} > 0$, there exist a reduced-order model u_θ and a bounded open* *set $\Theta \subset \mathbb{R}^m$, such that for every $\theta \in \bar{\Theta}$ there exists a vector $\alpha_\theta \in \mathbb{R}^m$ satisfying*

$$\|\alpha_\theta \cdot \nabla_\theta u_\theta - F[u_\theta]\|_2 \le \bar{\varepsilon}.$$

Remark 1. *Assumption 4 provides an upper bound on the error when projecting $F[u_\theta]$ onto the tangent space $T_{u_\theta}\mathcal{M}$, which is spanned by the functions in $\nabla_\theta u_\theta$. This error bound is determined by the choice of u_θ and the parameter set Θ. This can be seen as a first move, we shall show in chapter 4 a way to extend this assumption with some basic requirements on u_θ but for the more general treatment of this subject presented in this chapter we will keep u_θ as a more general model and simply assume Assumption 4 holds.*

We provide an example of a neural network with a special structure to justify the reasonableness of Assumption 4.

Example 1. *Let $\bar{\varepsilon} > 0$ and $\{\varphi_j\}_{j=1}^\infty$ be a complete smooth orthonormal basis (e.g., generalized Fourier basis) for $L^2(\Omega)$. Suppose there exist $C > 0$, $\gamma > 1$, and $C_0 > 0$ such that for all $u \in C^3(\Omega) \cap C(\bar{\Omega})$ and $\|u\|_2^2 \le C_0$ we have*

$$F[u] \in \mathcal{G}^{C,\gamma} := \{f \in C^1(\Omega) \cap C(\bar{\Omega}) : |\langle f, \varphi_j\rangle|^2 \le C j^{-\gamma},\ \forall\, j \ge 1\}. \tag{3.10}$$

Then there exists $m = m(\bar{\varepsilon}, C, \gamma) \in \mathbb{N}$ such that $\sum_{j=m+1}^\infty C j^{-\gamma} < \bar{\varepsilon}^2$. Consider $u_\theta = \theta \cdot \varphi = \sum_{j=1}^m \theta_j \varphi_j$. We denote $f_\theta := F[u_\theta]$ for short. Then $\nabla_\theta u_\theta = \varphi = (\varphi_1, \dots, \varphi_m)$ and for $\alpha^{f_\theta} = (\alpha_1^{f_\theta}, \dots, \alpha_m^{f_\theta})$ with $\alpha_j^{f_\theta} := \langle f_\theta, \varphi_j\rangle$, there is

$$\|\alpha^{f_\theta} \cdot \nabla_\theta u_\theta - F[u_\theta]\|_2^2 = \Big\| \sum_{j=1}^m \alpha_j^{f_\theta} \varphi_j - f_\theta \Big\|_2^2 = \sum_{j=m+1}^\infty |\langle f_\theta, \varphi_j\rangle|^2 \le \bar{\varepsilon}^2.$$

Therefore, the reduced-order model $u_\theta = \theta \cdot \varphi$ *with* $\Theta = \{\alpha \in \mathbb{R}^m : |\alpha|^2 < C_0\}$ *and* $\alpha_\theta = \alpha^{f_\theta}$ *satisfy Assumption 4.*

This example can be modified to use a more general form of u_θ*, such as a proper DNN. To see this, we first repeat the procedure above but with* $\bar{\varepsilon}$ *replaced by* $\bar{\varepsilon}/2$*. Then the universal approximation theorem [47, 123] and the continuity of DNNs in its parameters imply that there exist DNNs* $\{\hat{\varphi}_j : 1 \le j \le m\}$*, whose network parameters are collectively denoted by* $\eta \in \mathbb{R}^{m'}$*, satisfy* $\|\hat{\varphi}_j - \varphi_j\|_\infty \le \bar{\varepsilon}/(2\sqrt{mC_0|\Omega|})$ *and hence* $\|\hat{\varphi}_j - \varphi_j\|_2 \le \bar{\varepsilon}/(2\sqrt{mC_0})$ *for all* η *in an open set* $H \subset \mathbb{R}^{m'}$*. Consider the DNN* $u_\theta = c \cdot \hat{\varphi}$ *with parameters* $\theta = (c, \eta) \in \mathbb{R}^n$ *where* $n = m + m'$*. Then* $\nabla_c u_\theta(x) = (\hat{\varphi}_1, \dots, \hat{\varphi}_m)$*. Using the example above, we know for any* $f_\theta := F[u_\theta] \in \mathcal{G}^{C,\gamma}$*, there exists* $\alpha^{f_\theta} \in \mathbb{R}^m$ *such that* $\|\alpha^{f_\theta} \cdot \varphi - F[u_\theta]\|_2 \le \bar{\varepsilon}/2$*. Therefore, we use* $(\alpha^{f_\theta}, 0)$ *which concatenates* α^{f_θ} *and* $0 \in \mathbb{R}^{m'}$ *as the combination coefficients of* $\nabla_\theta u_\theta$ *to obtain*

$$\begin{aligned}
\|(\alpha^{f_\theta}, 0) \cdot \nabla_\theta u_\theta - F[u_\theta]\|_2 &= \|\alpha^{f_\theta} \cdot \nabla_c u_\theta - F[u_\theta]\|_2 \\
&\le \|\alpha^{f_\theta} \cdot \hat{\varphi} - \alpha^{f_\theta} \cdot \varphi\|_2 + \|\alpha^{f_\theta} \cdot \varphi - F[u_\theta]\|_\infty \\
&\le \sum_{j=1}^{m} |\alpha_j^{f_\theta}| \|\hat{\varphi}_j - \varphi_j\|_2 + \frac{\bar{\varepsilon}}{2} \\
&\le \sqrt{mC_0} \cdot \frac{\bar{\varepsilon}}{2\sqrt{mC_0}} + \frac{\bar{\varepsilon}}{2} \\
&= \bar{\varepsilon}.
\end{aligned}$$

Therefore, the DNN $u_\theta = c \cdot \hat{\varphi}$ *with* $\Theta = \{(c, \eta) : |c_j|^2 < C_0,\ \eta \in H\}$ *and* $\alpha_\theta = (\alpha^{f_\theta}, 0)$ *satisfy Assumption 4.*

Before proving the main proposition of this section we will need the following lemma.

Lemma 5. *Suppose Assumption 3 and 4 are satisfied. For all $\varepsilon > \bar{\varepsilon}$ there exists $v : \bar{\Theta} \to \mathbb{R}^m$ such that v is bounded over $\bar{\Theta}$ and the value of v at θ, denoted by v_θ, satisfies*

$$\|v_\theta \cdot \nabla_\theta u_\theta - F[u_\theta]\|_2 \leq \varepsilon, \qquad \forall\, \theta \in \bar{\Theta}.$$

Proof. Let $\varepsilon > \bar{\varepsilon}$ and $\delta \in (0, \varepsilon - \bar{\varepsilon})$. By Assumption 4, for all $\theta \in \Theta$ there exists $\alpha_\theta \in \mathbb{R}^m$ coefficient such that

$$\|\alpha_\theta \nabla_\theta u_\theta - F[u_\theta]\|_2 \leq \bar{\varepsilon}.$$

As $F[u_\theta]$ and $\nabla_\theta u_\theta$ are continuous in θ and Ω is bounded, we associate to each θ and coefficient α_θ the open set u_θ containing θ, small enough, such that for all $\theta' \in u_\theta$ we have

$$\|\alpha_\theta \nabla_\theta u_{\theta'} - \alpha_\theta \nabla_\theta u_\theta\|_2 + \|F[u_\theta] - F[u_{\theta'}]\|_2 \leq \delta \tag{3.11}$$

and hence

$$\|\alpha_\theta \cdot \nabla_\theta u_{\theta'} - F[u_{\theta'}]\|_2 \leq \|\alpha_\theta \nabla_\theta u_{\theta'} - \alpha_\theta \nabla_\theta u_\theta\|_2 + \|\alpha_\theta \nabla_\theta u_\theta - F[u_\theta]\|_2 + \|F[u_\theta] - F[u_{\theta'}]\|_2 \leq \delta + \bar{\varepsilon}. \tag{3.12}$$

Therefore $\cup_{\theta \in \bar{\Theta}} u_\theta$ is an open cover of $\bar{\Theta}$. As $\bar{\Theta}$ is compact this open cover has a finite subcover $\cup_{i=1}^N U_{\theta_i}$ for particular θ_i's. Define $v : \bar{\Theta} \to \mathbb{R}^m$ such that $v_\theta := v(\theta) = \alpha_{\theta_i}$ if $\theta \in U_{\theta_i}$ (if θ is in the intersection of multiple U_{θ_i}'s we choose a single α_{θ_i} arbitrarily). We see from this construction that v_θ is uniformly bounded over $\bar{\Theta}$ as the range of v_θ is finite. From (3.12) we have

$$\|v_\theta \cdot \nabla_\theta u_\theta - F[u_\theta]\|_2 \le \delta + \bar{\varepsilon} \le \varepsilon.$$

□

With Assumptions 3 and 4, and Lemma 5 we can prove the existence of an accurate neural control field V_ξ parameterized as a neural network, as shown in the next proposition.

Proposition 1. *Suppose Assumption 3 and 4 hold. Then for any $\varepsilon > 0$, there exists a differentiable vector field parameterized as a neural network $V_\xi : \bar{\Theta} \to \mathbb{R}^m$ with parameters ξ, such that*

$$\|V_\xi(\theta) \cdot \nabla_\theta u_\theta - F[u_\theta]\|_2 \le \varepsilon,$$

for all $\theta \in \bar{\Theta}$.

Proof. We first show that there exists a differentiable vector-valued function $V : \bar{\Theta} \to \mathbb{R}^d$ such that

$$\|V(\theta) \cdot \nabla_\theta u_\theta - F[u_\theta]\|_2 \le \frac{\varepsilon}{2} \tag{3.13}$$

for all $\theta \in \bar{\Theta}$. To this end, we choose $\bar{\varepsilon}_0 \in (0, \varepsilon/2)$ and $\bar{\varepsilon} \in (\bar{\varepsilon}_0, \varepsilon/2)$, then by Assumption 4 and Lemma 5 we know that there exist a reduced-order model u_θ, a bounded open set $\Theta \subset \mathbb{R}^m$, and $M_v > 0$ such that there is a vector-valued function $\theta \mapsto v_\theta$, where for any $\theta \in \bar{\Theta}$, we have $|v_\theta| < M_v$ and

$$\|v_\theta \cdot \nabla_\theta u_\theta - F[u_\theta]\|_2 \le \bar{\varepsilon}.$$

Note that v_θ is not necessarily differentiable with respect to θ. To obtain a differentiable vector field $V(\theta)$, for each $\theta \in \bar{\Theta}$, we define the function ψ_θ by

$$\psi_\theta(w) := \|w \cdot \nabla_\theta u_\theta - F[u_\theta]\|_2^2 = w^\top G(\theta) w - 2w^\top p(\theta) + q(\theta),$$

where

$$G(\theta) := \int_\Omega \nabla_\theta u_\theta(x) \nabla_\theta u_\theta(x)^\top \, dx, \quad p(\theta) := \int_\Omega \nabla_\theta u_\theta(x) F[u_\theta](x) \, dx, \quad q(\theta) := \int_\Omega F[u_\theta](x)^2 \, dx. \tag{3.14}$$

Then we know

$$\psi_\theta^* := \psi_\theta(v_\theta) = \|v_\theta \cdot \nabla_\theta u_\theta - F[u_\theta]\|_2^2 \leq \bar{\varepsilon}^2. \tag{3.15}$$

It is also clear that $G(\theta)$ is symmetric and positive semi-definite. Moreover, due to the compactness of $\bar{\Omega}$ and $\bar{\Theta}$, as well as that $\nabla_\theta u \in C(\bar{\Omega} \times \bar{\Theta})$, we know there exists $\lambda_G > 0$ such that

$$\|G(\theta)\|_2 \leq \lambda_G$$

for all $\theta \in \bar{\Theta}$. Therefore, ψ_θ is a convex function and the Lipschitz constant of $\nabla \psi_\theta$ is uniformly upper bounded by λ_G over $\bar{\Theta}$. Now for any $w \in \mathbb{R}^m$, $h > 0$, and $K \in \mathbb{N}$ (we reuse the letter K as the iteration counter instead of the number of sampling points in this proof), we define

$$\mathcal{O}_\theta^{K,h}(w) := w_K, \quad \text{where} \quad w_k = w_{k-1} - h \nabla \psi_\theta(w_{k-1}), \quad w_0 = w, \quad k = 1, \dots, K.$$

Namely, $\mathcal{O}_\theta^{K,h}$ is the oracle of executing the gradient descent optimization scheme on ψ_θ with step size $h > 0$ for K iterations.

Next, we slightly modify the standard convergence result of gradient descent in convex optimization [85, Theorem 2.1.14] and obtain Lemma 8 in Appendix A. Notice that ψ_θ is convex, differentiable, and $\nabla\psi_\theta$ is Lipschitz continuous with Lipschitz constant upper bounded by λ_G. Therefore, applying Lemma 8 with $y = v_\theta$, $f = \psi_\theta$, and the gradient descent scheme for K iterations (K to be determined soon) with initial 0 and any fixed step size $h \in (0, 1/\lambda_G)$ to ψ_θ directly yields an error bound for $\psi_\theta(\mathcal{O}_\theta^{K,h}(0))$:

$$\psi_\theta(\mathcal{O}_\theta^{K,h}(0)) - \psi_\theta^* \le \frac{|0 - v_\theta|^2}{2Kh}. \tag{3.16}$$

Combining this with the bound $|v_\theta| < M_v$, we choose any

$$K \ge \frac{M_v^2}{2h((\varepsilon/2)^2 - \bar{\varepsilon}^2)},$$

and there is

$$\psi_\theta(\mathcal{O}_\theta^{K,h}(0)) - \psi_\theta^* \le \frac{M_v^2}{2Kh} \le \left(\frac{\varepsilon}{2}\right)^2 - \bar{\varepsilon}^2. \tag{3.17}$$

Notice that $\mathcal{O}_\theta^{K,h}$ is a differentiable vector-valued function of θ because K and h are fixed. Therefore, combining (3.15) and (3.17) yields

$$0 \le \psi_\theta(\mathcal{O}_\theta^{K,h}(0)) = (\psi_\theta(\mathcal{O}_\theta^{K,h}(0)) - \psi_\theta^*) + \psi_\theta^* \le (\varepsilon/2)^2 - \bar{\varepsilon}^2 + \bar{\varepsilon}^2 = (\varepsilon/2)^2.$$

As this inequality holds $\forall\theta \in \bar{\Theta}$, we set $V(\theta) = \mathcal{O}_\theta^{K,h}(0)$ which is a differentiable function of θ satisfying (3.13).

By the universal approximation theorem of neural networks [39] (see also the discussion in the introduction), we know there exists a differentiable vector-valued function parameterized as a neural network V_ξ with parameters ξ such that

$$|V_\xi(\theta) - V(\theta)|_\infty \le \varepsilon/(2B)$$

for all $\theta \in \bar{\Theta}$, where $B := \max_{\theta\in\bar{\Theta}} \|\nabla_\theta u_\theta\|_2 < \infty$ and $|\cdot|_\infty$ stands for the ∞-norm of vectors. Hence we know

$$\|V_\xi(\theta)\cdot\nabla_\theta u_\theta - F[u_\theta]\|_2 \le \|V_\xi(\theta)\cdot\nabla_\theta u_\theta - V(\theta)\cdot\nabla_\theta u_\theta\|_2 + \|V(\theta)\cdot\nabla_\theta u_\theta - F[u_\theta]\|_2 \le B\cdot\frac{\varepsilon}{2B} + \frac{\varepsilon}{2} = \varepsilon.$$

This completes the proof. $\square$

Remark 2. *It is important to note the geometry of $\mathcal{M}$, especially its dimensionality, is complex and highly dependent on the structure of u_θ and the parameter space Θ. In particular, we can show that the tangent space $T_{u_\theta}\mathcal{M} = span(\nabla_\theta u_\theta)$ at any $u_\theta \in \mathcal{M}$ is in the L^2 space, where $\nabla_\theta u_\theta = (\partial_{\theta_1} u_\theta, \dots, \partial_{\theta_m} u_\theta)$ for $\theta = (\theta_1, \dots, \theta_m)$. (Here we use discrete indices $1, \dots, m$ as subscripts of θ to indicate its components for notation simplicity. This is to be distinguished from the subscript t in $\theta(t)$ which stands for time of the trajectory $\theta(t)$.) However,* $\dim(T_{u_\theta}\mathcal{M})$ *may vary across different u_θ on $\mathcal{M}$. For example, consider the reduced-order model u_θ parameterized as a DNN as in* (1.2)*: when $w = 0$, we have $\theta = (0, b, \cdots)$ and hence $\partial_{W_l} u_\theta = 0$ and $\partial_{b_l} u_\theta = 0$ for all $l = 1, \dots, L$. In this case, the m components of $\nabla_\theta u_\theta$ are* not *linearly independent, and* $\dim(T_{u_\theta}\mathcal{M}) < m$ *for such θ's. This distinguishes our parameter submanifold from existing ones, such as [4], which assumes that the tangent space is always of full dimension m at any point of the submanifold. In our case, however, challenges and complications in dealing with the parameter submanifold $\mathcal{M}$ can be avoided if we made such an assumption, but it will lead to incorrect analysis and error estimation, which poses a* *major technical challenge for the proposed framework. Specifically, we note that the rank of $G(\theta)$ varies across Θ, and therefore the pseudoinverse $G(\theta)^+$ may be discontinuous. A major theoretical merit of Proposition 1 is that we can still ensure the existence of a differentiable control vector field in such case.*

3.3.2.2 Error analysis in solving (semi-)linear parabolic PDEs

Now we are ready to provide error bounds of our method in solving a large class of linear and semilinear parabolic PDEs. This class of PDEs covers many types of reaction-diffusion equations, such as heat equations, Fisher's equation or the Allen-Cahn equation. The differential operator F in linear and semilinear parabolic PDEs has the form

$$F[u] = \nabla \cdot (A\nabla u) + b \cdot \nabla u + f(u)$$

where $A : \Omega \to \mathbb{R}^{d\times d}$ and $b : \Omega \to \mathbb{R}^d$ are continuous, $f : \mathbb{R} \to \mathbb{R}$ is L_f-Lipschitz and acts on $u(x)$ for each x. Moreover we assume that there exist $\lambda \geq 0$ and $B \geq 0$ such that

$$z^\top A(x)z \geq \lambda |z|^2, \quad \forall\, z \in \mathbb{R}^d,\ x \in \Omega, \tag{3.18}$$

and

$$\|\nabla \cdot b\|_\infty \leq B. \tag{3.19}$$

Furthermore, due to the smoothness of V_ξ and compactness of $\bar{\Theta}$, we know there exist $M_V > 0$ and $L_V > 0$ such that

$$\max_{\theta\in\bar{\Theta}} |V_\xi(\theta)| \leq M_V \qquad \text{and} \qquad \max_{\theta\in\bar{\Theta}} |\nabla_\theta V_\xi(\theta)| \leq L_V. \tag{3.20}$$

Theorem 3. *Suppose Assumptions 3 and 4 hold. Then there exist control field V_ξ such that* *for any u^* satisfying the evolution PDE in* (3.2) *there is*

$$\|u_{\theta(t)}(\cdot) - u^*(\cdot,t)\|_2 \le e^{(L_f+B/2-\lambda/C_p)t}(\varepsilon_0 + \varepsilon t) \tag{3.21}$$

for all t as long as $\theta(t) \in \bar{\Theta}$, where $\theta(t)$ is solved from the ODE (3.6) *with V_ξ and initial $\theta(0)$ satisfying $\|u_{\theta(0)}(\cdot) - u^*(\cdot,0)\|_2 \le \varepsilon_0$. Here C_p is a constant depending only on Ω.*

Proof. We denote the residual

$$r(x,t) := \nabla_\theta u_{\theta(t)}(x) \cdot V_\xi(\theta(t)) - F[u_{\theta(t)}](x).$$

Then by Proposition 1 we have $\|r(\cdot,t)\|_2 \le \varepsilon$ for all t. Furthermore, we denote

$$\delta(x,t) := u_{\theta(t)}(x) - u^*(x,t)$$

for all $(x,t) \in \bar{\Omega} \times [0,T]$ and $D(t) := \|\delta(\cdot,t)\|_2$, then there is

$$D'(t) = \Big\langle \frac{\delta(\cdot,t)}{\|\delta(\cdot,t)\|_2}, \partial_t \delta(\cdot,t) \Big\rangle. \tag{3.22}$$

Here we use the convention that $\delta(\cdot,t)/\|\delta(\cdot,t)\|_2 = 0$ if $\delta(\cdot,t) = 0$ a.e. By the definition of δ, we have

$$\begin{aligned}
\partial_t \delta(x,t) &= \partial_t u_{\theta(t)}(x) - \partial_t u^*(x,t) \\
&= \nabla_\theta u_{\theta(t)}(x) \cdot \dot{\theta}(t) - F[u^*](x,t) \\
&= \nabla_\theta u_{\theta(t)}(x) \cdot V_\xi(\theta(t)) - F[u^*](x,t) \\
&= F[u_{\theta(t)}](x) - F[u^*](x,t) + r(x,t) \\
&= \nabla \cdot (A(x) \nabla \delta(x,t)) + b(x) \cdot \nabla \delta(x,t) + f(u_{\theta(t)}(x)) - f(u^*(x,t)) + r(x,t).
\end{aligned}$$

Therefore, we have

$$\begin{aligned}\langle\delta(\cdot,t),\partial_t\delta(\cdot,t)\rangle &= \int_\Omega \delta(x,t)\left(\nabla\cdot(A(x)\nabla\delta(x,t)) + b(x)\cdot\nabla\delta(x,t)\right)\,dx \\ &\quad + \int_\Omega \delta(x,t)(f(u_{\theta(t)}(x)) - f(u^*(x,t)) + r(x,t))\,dx \\ &=: I(t) + J(t).\end{aligned} \tag{3.23}$$

Because $u_{\theta(t)}(\cdot)|_{\partial\Omega} = u^*(\cdot,t)|_{\partial\Omega} = 0$, we know $\delta(\cdot,t)|_{\partial\Omega} = 0$. Thus, we have

$$\begin{aligned}I(t) &= \int_\Omega \delta(x,t)\left(\nabla\cdot(A(x)\nabla\delta(x,t)) + b(x)\cdot\nabla\delta(x,t)\right)\,dx \\ &= -\int_\Omega \nabla\delta(x,t)^\top A(x)\nabla\delta(x,t)\,dx - \frac{1}{2}\int_\Omega(\nabla\cdot b(x))\delta(x,t)^2\,dx \\ &\le -\lambda\int_\Omega|\nabla\delta(x,t)|^2\,dx - \frac{1}{2}\int_\Omega(\nabla\cdot b(x))\delta(x,t)^2\,dx \\ &\le -\frac{\lambda}{C_p}\int_\Omega|\delta(x,t)|^2\,dx + \frac{B}{2}\int_\Omega|\delta(x,t)|^2\,dx,\end{aligned} \tag{3.24}$$

where the first equality is just by the definition of $I(t)$, the second equality is obtained by integrating by parts on both terms and using $\delta(\cdot,t)|_{\partial\Omega} = 0$, the first inequality is due to (3.18), and the last inequality is due to the Poincare's inequality

$$\|\delta(\cdot,t)\|_2 \le C_p\|\nabla\delta(\cdot,t)\|_2$$

as $\delta(\cdot,t)|_{\partial\Omega} = 0$ for all t (here C_p the Poincare's constant depending on Ω only) and the bound (3.19). We can also obtain

$$\begin{aligned} J(t) &= \int_\Omega \delta(x,t)(f(u_{\theta(t)}(x)) - f(u^*(x,t)) - r(x,t))\, dx \\ &\leq \int_\Omega |\delta(x,t)| \cdot |f(u_{\theta(t)}(x)) - f(u^*(x,t)) - r(x,t)|\, dx \\ &\leq \int_\Omega |\delta(x,t)| \cdot (L_f|\delta(x,t)| + |r(x,t)|)\, dx \\ &\leq L_f\|\delta(x,t)\|_2^2 + \|r(\cdot,t)\|_2\|\delta(\cdot,t)\|_2 \\ &\leq L_f\|\delta(x,t)\|_2^2 + \varepsilon\|\delta(\cdot,t)\|_2, \end{aligned} \tag{3.25}$$

where the first identity is by the definition of $J(t)$, the second inequality is due to the Lipschitz condition of f. Combining (3.22), (3.23), (3.24) and (3.25), we obtain

$$D'(t) \leq \Big(L_f + \frac{B}{2} - \frac{\lambda}{C_p}\Big)D(t) + \varepsilon.$$

By Grönwall's inequality we deduce that

$$D(t) \leq e^{(L_f + B/2 - \lambda/C_p)t}(D(0) + \varepsilon t).$$

Recall that

$$D(0) = \|\delta(\cdot,0)\|_2 = \|u_{\theta(0)}(\cdot) - u^*(\cdot,0)\|_2 = \|u_{\theta(0)}(\cdot) - g(\cdot)\|_2 \leq \varepsilon_0,$$

we thus have

$$\|u(\cdot,\theta(t)) - u(\cdot,t)\|_2 = D(t) \leq e^{(L_f + B/2 - \lambda/C_p)t}(\varepsilon_0 + \varepsilon t)$$

for all time t, which completes the proof. □

The error estimate in Theorem 3.5 indicates that the approximation error is determined by three factors: (i) the approximation error ε_0 of the reduced order model to the initial value g, (ii) the local approximation error ε of the projection of $F[u_\theta]$ onto the tangent space of $\mathcal{M}$ at u_θ; and (iii) the irregularity of the differential operator F itself. While the error from (iii) is determined by the given PDE, we can make an effort to suppress (i) and (ii) in practice by robust architecture of u_θ and the training of V_ξ. We note the error estimate provided in Theorem 3.5 is an upper bound of the approximation error.

Remark 3. *While we assumed f to be globally Lipschitz, the result in Theorem 3 still holds locally with local Lipschitz condition of f. For example, in the case of the Allen-Cahn example, we know if our initial function is bounded by 1 the true trajectories will remain bounded allowing the results of Theorem 3 to apply.*

Corollary 1. *Suppose the conditions in Theorem 3 hold. Let $\hat{\theta}_t$ be the numerical solution to the ODE* (3.6) *obtained by using the Euler scheme with step size $h > 0$. Then*

$$\|u_{\hat{\theta}_t}(\cdot) - u^*(\cdot, t)\|_2 \leq \frac{L_V M_V |\Omega| h}{2}(e^{L_V t} - 1) + e^{(L_f - B/2 + \eta/C_p)t}(\varepsilon_0 + \bar{\varepsilon} t) \tag{3.26}$$

for all t as long as $\theta(t) \in \bar{\Theta}$.

Proof. Given the estimate provided in Theorem 3, we only need to show

$$\|u_{\hat{\theta}_t}(\cdot) - u_{\theta(t)}(\cdot)\|_2 \leq \frac{L_V M_V |\Omega| h}{2}(e^{L_V t} - 1), \tag{3.27}$$

since combined with (3.21) it yields the claimed estimate (3.26). To show (3.27), we notice that

$$\ddot{\theta}_t = \frac{d}{dt} V_\xi(\theta(t)) = \nabla_\theta V_\xi(\theta(t)) \cdot \dot{\theta}(t) = \nabla_\theta V_\xi(\theta(t)) \cdot V_\xi(\theta(t)).$$

Therefore we have

$$|\ddot{\theta}_t| = |\nabla_\theta V_\xi(\theta(t)) \cdot V_\xi(\theta(t))| \le L_V M_V$$

where L_V and M_V are defined in (3.20). Hence, by the standard results for the Euler's method [6, pp. 346]), we know the numerical solution $\hat{\theta}_t$ satisfies

$$|\hat{\theta}_t - \theta(t)| \le \frac{hM_V}{2}\left(e^{L_V t} - 1\right) \tag{3.28}$$

for all t. Therefore, we obtain

$$\begin{aligned}\|u_{\hat{\theta}_t} - u_{\theta(t)}\|_2 &= \Big(\int_\Omega |u_{\hat{\theta}_t}(x) - u_{\theta(t)}(x)|^2\,dx\Big)^{1/2} = \Big(\int_\Omega |\nabla_\theta u_{\tilde{\theta}_t}(x)\cdot(\hat{\theta}_t - \theta(t))|^2\,dx\Big)^{1/2}\\ &\le L_V|\Omega||\hat{\theta}_t - \theta(t)| \le \frac{L_V M_V|\Omega|h}{2}(e^{L_V t} - 1),\end{aligned}$$

where the second equality is due to the fact that u_θ is C^1 in θ and hence the mean value theorem applies to u_θ (here $\tilde{\theta}_t$ is some point on the line segment between $\hat{\theta}_t$ and $\theta(t)$). $\square$

The proof above can be modified if a different numerical ODE solver is employed. In that case one can obtain improved upper bound and order in step size h in (3.28).

3.4 Numerical Results

3.4.1 Implementation

In Section 3.3.1, we have showed that the neural control field V_ξ is parameterized as a deep network, and its parameters ξ can be learned by solving

$$\min_\xi \left\{ \ell(\xi) := \int_\Theta \|V_\xi(\theta) \cdot \nabla_\theta u_\theta - F[u_\theta]\|_2^2 d\theta \right\}.$$

The first-order optimality condition of this minimization problem is given by $G(\theta)V_\xi(\theta) = p(\theta)$ where $G(\theta)$ and $p(\theta)$ are defined in (3.14). The objective function $\ell(\xi)$ above shares the same minimizers as the following one:

$$\bar{\ell}(\xi) := \int_\Theta |G(\theta)V_\xi(\theta) - p(\theta)|^2 \, d\theta. \tag{3.29}$$

In our numerical experiments, we use $\bar{\ell}$ defined in (3.29), as we can train to towards the optimal solution $V_\xi = G^+(\theta)p(\theta)$ as the optimal value which seems to produce lower error empirically.

Moreover, we know the minimum loss value of (3.29) is 0, which contrasts to (3.7) where the minimum loss value is often unknown.

In practice, as the dimension of θ and Ω could be large, we have to approximate (3.29) using techniques such as Monte-Carlo integration. This leads to the approximate forms

$$\tilde{G}(\theta) = \frac{1}{N_x} \sum_{i=1}^{N_x} \nabla_\theta u_\theta(x_i) \nabla_\theta u_\theta(x_i)^\top, \quad \tilde{p}(\theta) = \frac{1}{N_x} \sum_{i=1}^{N_x} \nabla_\theta u_\theta(x_i) F[u_\theta](x_i),$$

where x_i, $i = 1, \ldots, N_x$ are sampled from Ω. By also drawing samples from Θ, we arrive at our empirical loss function defined by

$$\ell_1(\xi) := \frac{1}{N_\theta}\sum_{j=1}^{N_\theta} |\tilde{G}(\theta_j)\cdot V_\xi(\theta_j) - \tilde{p}(\theta_j)|^2. \tag{3.30}$$

To improve the training of V_ξ, we also augment the loss function ℓ_1 in (3.30) with an additional term following a data-driven approach. Specifically, we follow the methods in [12, 26] to generate multiple sample trajectories starting from randomly sampled initial values $\{\theta(0)^{(i)} : i \in [M]\}$ in Θ. For the ith trajectory, a sequence of directions $\{v_j^{(i)} : j = 0, 1 \ldots, N_t\}$ are solved from linear systems $\tilde{G}(\theta_j^{(i)})v_j^{(i)} = \tilde{p}(\theta_j^{(i)})$ and the discrete-time points on the trajectory are obtained by $\theta_{j+1}^{(i)} = \theta_j^{(i)} + hv_j^{(i)}$ for $j = 0, 1, \ldots, N_t - 1$. We add the augment loss term

$$\ell_2(\xi) := \frac{1}{N_t M}\sum_{i=1}^{M}\sum_{j=1}^{N_t} |V_\xi(\theta_j^{(i)}) - v_j^{(i)}|^2. \tag{3.31}$$

Combining with (3.30), we obtain our final loss function

$$\ell_{\text{total}}(\xi) = \ell_1(\xi) + \zeta\ell_2(\xi), \tag{3.32}$$

where ζ is a weight parameter. In our experience for parabolic linear PDEs using only ℓ_1 is sufficient to generate a good result. For the nonlinear case adding ℓ_2 substantially improves training results empirically as network parameters may move far away from those we sampled near the initial parameters.

3.4.2 Experimental setting

To demonstrate the performance of the proposed method, we test it on three different PDEs: a 10-dimensional (10D) transport equation, a 10D heat equation, and a 2D Allen-Cahn equation. Both of the transport equation and heat equation are linear PDEs, while the Allen-Cahn is a highly nonlinear PDE. In fact, we also tested 10D Allen-Cahn equation but only present the result of the 2D one here. This is because the true solution of Allen-Cahn equation does not have closed-form, and we have to employ a classical finite difference method, which does not scale to 10D case, to produce a reference solution for comparison. In contrast, we have closed-form solutions of the IVPs with transport and heat equations and hence we can use them as the true solution for direct comparison. In our tests, we employ the following structure of our reduced-order model

$$u_\theta(x) = \alpha(x) z_L(x, \theta) \tag{3.33}$$

for the heat equation and Allen-Cahn equation. We use the following network structure

$$u_\theta(x) = z_L(\beta(x), \theta) \tag{3.34}$$

for the transport equation. In (3.33), $\alpha(x)$ is a distance function of $\partial\Omega$ such that it satisfies the zero boundary condition, and in (3.34) $\beta(x)$ is a function chosen to satisfy a periodic boundary condition as in [26]. This aligns with our choice of u_θ in (3.33) and (3.34) as the IVP with heat and Allen-Cahn equations have zero boundary value whereas the IVP with transport equation has periodic boundary value in our experiments. In both (3.33) and (3.34), z_L is the neural network and is defined by

$$z_L = w_L z_{L-1}, \quad z_l = z_{l-1} + \sigma(W_l z_{l-1} + b_l), \quad l = 1, \ldots, L-1 \tag{3.35}$$

and $z_0 = \sigma(W_0 x + b_0)$. Here σ is a user-chosen activation function (we use tanh or ReLU in our experiments) $W_l \in \mathbb{R}^{d' \times d'}$ are the weight matrices and $b_l \in \mathbb{R}^{d'}$ are the bias vectors, and $W_0 \in \mathbb{R}^{d' \times d}$ and $w_L \in \mathbb{R}^{1 \times d}$, all of these matrices and vectors make up the parameters vector θ. Networks such as in (3.35) are often called *residual neural networks* (ResNet), and have been shown performing better than basic feed forward networks in function approximation [108]. The values of L and d' in our experiments are shown in Table 3.1. They are selected manually to balance the depth L and width d' so that u_θ does not have too many neurons but still remains expressive. We use a similar structure for the vector field V_ξ, but adjust the layers to be $\eta_l = \eta_{l-1} + \mathrm{GeLU}(\bar{U}_l \theta + \bar{b}_l) \tanh(U_l \eta_{l-1} + b_l)$. Here $\mathrm{GeLU}(x) = x\Phi(x)$ where $\Phi(x)$ is the standard Gaussian cumulative distribution function. This is a slight modification of the network architecture proposed in [107] for improved effectiveness in training by gradient descent. We selected this network structure by starting with a ResNet with small width and depth and ReLU activation, then we gradually increased the width and depth until the improvement in the final loss value became insignificant. Finally, we attempted a few different activation functions and network architectures for this width and depth and selected the aforementioned structure which appeared to perform slightly better. This process was by no means exhaustive.

Other network architectures can be used as well. The width and depth of our network are reported in Table 3.1. Information about the number of trajectories used for (3.31) is also collected in Table 3.1. For all of the experiments, we set the weight $\zeta = 0.1$ in (3.32) to reflect the scale difference of the two loss terms and use the standard ADAM optimizer with learning rate 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$. We terminate the training process when the empirical loss $\ell_{\text{total}}(\xi) < 0.1$ or when the percent decrease of the empirical loss is less than 0.1% averaged over the past 100 steps. Once V_ξ is learned, we use the 4th-order Runge-Kutta method with a step size of $T/200$ (T is determined from the problem) to solve $\theta(t)$ from the ODE in (3.6) in Algorithm 2 and compare the corresponding u_θ with the reference solutions. All the implementations and experiments are performed using PyTorch in Python 3.9 in Windows 10 OS on a desktop computer with an AMD Ryzen 7 3800X 8-Core Processor at 3.90 GHz, 16 GB of system memory, and an Nvidia GeForce RTX 2080 Super GPU with 16 GB of graphics memory. Total computational time is split between three unique activities: (i) the generation of N_θ samples for ℓ_1 in (3.30); (ii) the generation of the M trajectories for ℓ_2 in (3.31); and (iii) the training of the network V_ξ. Parts (i) and (ii) can be parallelized offline to speed up the process. We discuss the specific time cost of the implementation of our method in the examples below.

We also provide a few remarks on the sampling strategy in Θ. While one can draw θ uniformly from Θ, adding samples θ corresponding to some example solutions to the PDE may further improve training efficiency. In practice, we use both uniformly sampled θ's and those close to the θ's corresponding to some randomly chosen initial functions. These initial functions are only used to help the loss function weigh more on the regions that are potentially more important than others in Θ; but they are not among the randomly chosen initial functions used for any testing. Details on samplings are given below.

Table 3.1 Problem settings, network structures, and the number of training trajectories/samples in numerical experiments. Here M is the number of trajectories used from Θ and N_θ is the total number of samples from Θ.

Problem	**Dim.** d	u_θ **Width/Depth**	V_ξ **Width/Depth**	M	N_θ
Transport Equation	10	12/4	1,500/4	0	160,000
Heat Equation	10	12/5	2,000/10	600	200,000
Allen-Cahn Equation	2	10/3	2,000/5	200	200,000

3.4.3 Numerical results on transport equation

We first consider the initial value problem defined by a 10D transport equation with periodic boundary conditions as follows:

$$\begin{cases} \partial_t u(x,t) = -\mathbf{1}\cdot\nabla_x u(x,t), & \forall\, x\in\Omega,\ t\in[0,T], \\ u(x,0) = g(x), & \forall\, x\in\bar{\Omega}, \end{cases} \tag{3.36}$$

where $\Omega=(0,1)^{10}$, $T=1$, $\mathbf{1}$ is the vector whose components are all ones, and the boundary value $u(x,t)=0$ for all $x\in\partial\Omega$ and $t\in[0,T]$. This IVP has the true solution $u^*(x,t) = g(x-\mathbf{1}\cdot t)$. We obtain the solution operator of the IVP (3.36), we use (3.34) as the reduced-order model u_θ. Although our error analysis requires certain regularity on initial and solution of PDEs, we test on the case where both are only Lipschitz continuous but not differentiable for this transport equation. To this end, we set the activation of u_θ to ReLU. Further, define $\beta(x) = (\cos(2\pi(x-b)), \sin(2\pi(x-b)))^\top$ where $b\in\mathbb{R}^{10}$ is a trainable parameter with sin and cos acting component-wise to x. This means that the first hidden layer uses $W_0\in\mathbb{R}^{12\times 20}$. For this example, we shall set $\Theta=[-1,1]^m$ where m are the number of parameters in u_θ. Then we train the neural control vector field V_ξ by minimizing (3.7) with the number of sampled θ drawn uniformly from Θ shown in Table 3.1. We note that this equation performed equally well with or without the loss ℓ_2 in (3.31). As such we need not generate any trajectories and this is reflected in Table 3.1.

After the control V_ξ is obtained, we test the performance of V_ξ on a variety of initial values g by uniformly sampling $\theta(0) \in \Theta$. We emphasize that the corresponding $\theta(0)$'s to these initial values are not used in the training process. We show three approximate solutions for three random initials in Figure 3.2. For the first random initial g_1 determined by the random $\theta(0)$, we plot the corresponding true solution $u^*(\cdot, t)$, the approximate solution $u_{\theta(t)}(\cdot)$ obtained by Algorithm 2, and their pointwise absolute difference $|u_{\theta(t)}(x) - u^*(x,t)|$ from row 1 to row 3 in Figure 3.2 respectively for $t = 0, 0.15, 0.5, 0.85, 1$. The plots for the second and third g_2 and g_3 random initials are shown in rows 4–6 and 7–9 in Figure 3.2 respectively. From Figure 3.2, we can see that the reduced-order model $u_{\theta(t)}$ with $\theta(t)$ controlled by the trained vector field V_ξ closely approximates the true solution $u^*(\cdot, t)$ with low absolute errors (note that the scale of the error is different from that of $u^*(\cdot, t)$ and $u_{\theta(t)}(\cdot)$). Figure 3.3a and 3.3b plots the mean of the absolute error $\|u^*(\cdot, t) - u_{\theta(t)}(\cdot)\|_2^2$, and the relative error $\|u^*(\cdot, t) - u_{\theta(t)}(\cdot)\|_2^2 / \|u^*(\cdot, t)\|_2^2$ respectively over 100 randomly chosen initials, while the standard deviation is shaded in. We see mean errors $< 1\%$ even though the initial functions considered are not smooth. This suggests that the proposed model can generalize to the case where the initial and solution of the PDE are not sufficiently smooth.

We now discuss the computational cost of the method. In our tests, it took 1.78 hours to generate $\tilde{G}$ and $\tilde{p}$ from the samples in Θ used for training. Once generated, the training of V_ξ (i.e., minimizing the loss function ℓ_{total} in (3.32)) took 5 minutes to complete. Testing each initial condition by solving (3.6) using a 4th-order Runge-Kutta (RK4) solver with step size 0.005 took an average of 2.1 seconds per initial. We note that no time is needed in this case to fit an initial, as $\theta(0)$ is chosen randomly.

The proposed method has evident improvement on computational cost over existing methods that only solve specific instances of the PDEs. In this test, we compare the computational cost with PINN [94] and a time marching (TM) [26] method. We use the same structure of u_θ for PINN and time marching as used by our method. We sample 10,000 points $(x, t) \in (0, 1)^{10} \times [0, 1]$ for PINN and 10,000 points $x \in (0, 1)^{10}$ for each step of the time marching method. We train PINN using its default parameters until convergence. For TM, we use RK4 with the same step size 0.005 and its default linear system solver for each step. We follow all other implementation steps of both PINN and TM as described in their original papers. For a single initial g, PINN, TM, and the proposed method took 116.5s, 16.7s, and 2.1s respectively to obtain the solution. This significant time reduction is due to the fact that the proposed method has learned the control field in the parameter space and thus can compute the solution of the PDE by solving an ODE which has very low computation complexity. The improvement is more significant for higher-order PDEs because PINN and TM require more time to compute the differential operator whereas the computation complexity of the proposed method remains the same.

The proposed method is capable of approximating solution operators of high-dimensional

PDEs whereas existing methods cannot. This is because existing solution operator learning methods, such as DeepONet, require spatial discretization, and thus the network size and sampling amounts grow exponentially fast in problem dimension. For example, for a one-dimensional ($d = 1$) evolution PDE, DeepONet [76] requires 100 sample solutions (which must be generated by another numerical method) each evaluated at 10^4 grid points in the (x, t) domain in $\mathbb{R} \times \mathbb{R}_+$. Thus the size of their trunk network alone is already 10 times larger than our V_ξ in the 10-dimensional case. When the problem dimension d becomes over 3, DeepONet will be infeasible computationally. In addition, our method does not require sample solutions which could be unavailable or difficult to obtain in practice.

3.4.4 Heat equation

Next we consider an initial value problem with heat equation in 10D:

$$\begin{cases} \partial_t u(x,t) = \Delta u(x,t), & \forall\, x \in \Omega, t \in [0,T] \\ u(x,0) = g(x), & \forall\, x \in \bar{\Omega}, \end{cases} \tag{3.37}$$

where $\Omega = (0,1)^{10}$ and the boundary value $u(x,t) = 0$ for all $x \in \partial\Omega$ and $t \in [0,T]$. As most of the initial conditions we consider have rapid evolution in a short time, we use $T = 0.01$ in this test. For neural network we use (3.33), with $\alpha(x) = \Pi_{i=1}^{10} 4(x_i - x_i^2)$ and tanh activation.

In order to have a class of analytical examples to compare against, we use the base

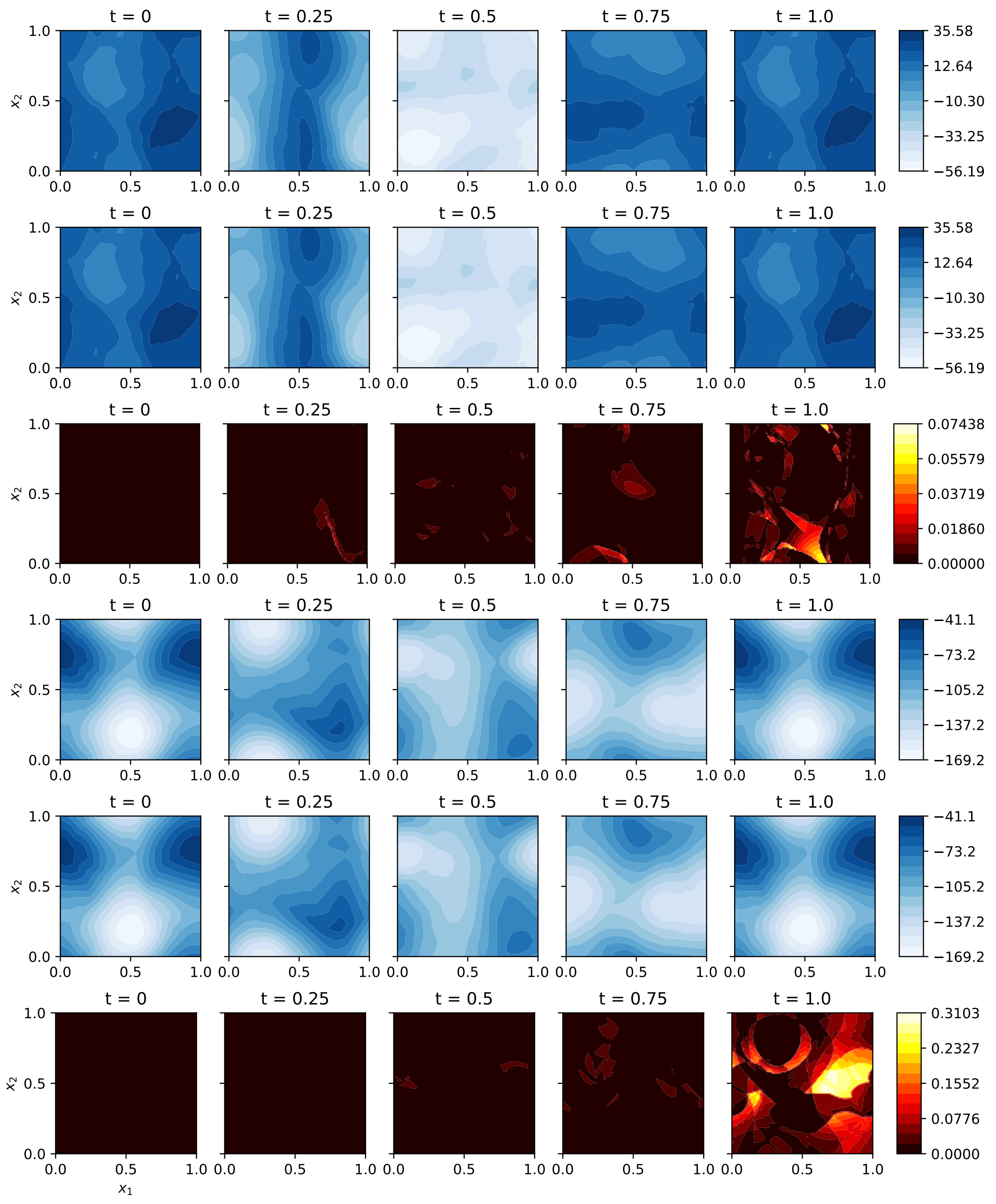


Figure 3.2 (Transport equation). Comparison between true solution $u^*(\cdot, t)$, the approximation $u_{\theta(t)}(\cdot)$ and their pointwise absolute difference $|u_{\theta(t)}(x) - u^*(x,t)|$ for times $t = 0, 0.15, 0.5, 0.85, 1$ for IVPs with the first initial (rows 1–3), second initial (rows 4–6) and third initial (rows 7–9) given by u_θ with θ randomly drawn from $[-1,1]^m$.

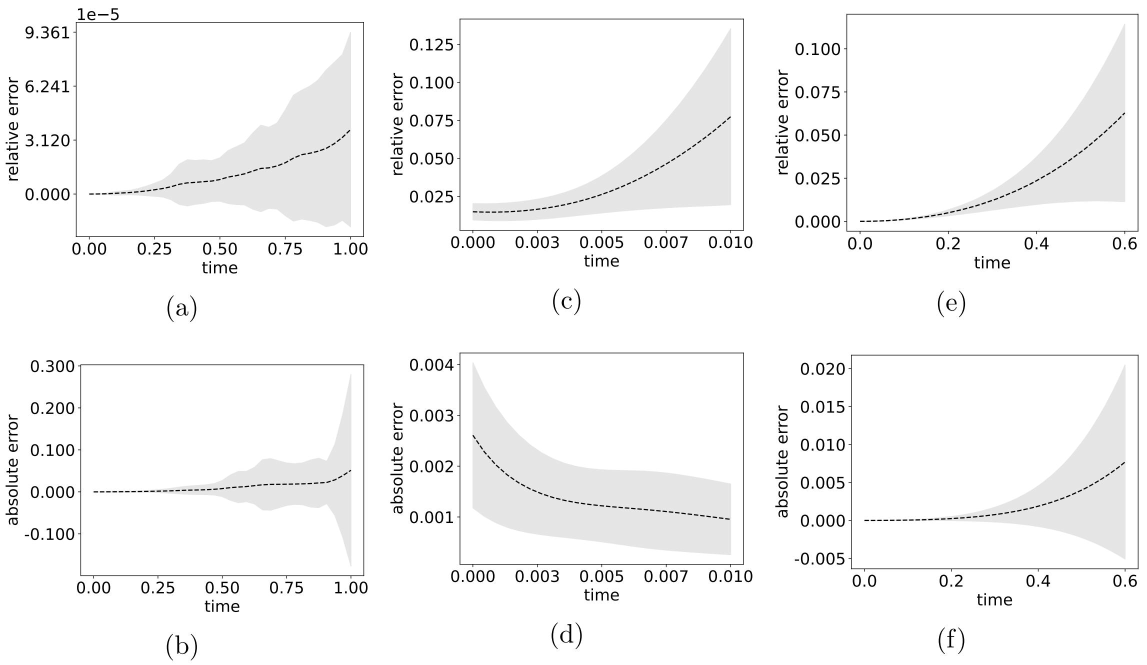


Figure 3.3 Comparison of the mean relative error $\|u^*(\cdot,t) - u_{\theta(t)}(\cdot)\|_2^2 / \|u^*(\cdot,t)\|_2^2$ (top) and mean absolute $\|u^*(\cdot,t) - u_{\theta(t)}(\cdot)\|_2^2$ (bottom) versus time t for 100 different initial conditions of the transport (a)-(b), heat (c)-(d), and Allen-Cahn (e)-(f) equations. Shaded areas indicate the standard deviation over the 100 results.

functions

$$
\begin{aligned}
g_1(x) &= \Pi_{i=1}^{10} \sin(\pi x_i), \\
g_2(x) &= \sin(2\pi x_1)\Pi_{i=1}^{10} \sin(\pi x_i), \\
g_3(x) &= \sin(2\pi x_2)\Pi_{i\neq 2}^{10} \sin(\pi x_i) \\
g_4(x) &= \sin(2\pi x_1)\sin(2\pi x_2)\Pi_{i=3}^{10} \sin(\pi x_i).
\end{aligned}
\tag{3.38}
$$

to generate a class of initial conditions $\mathcal{G} := \{\sum_{i=1}^{4} c_i g_i \ : \ c_i \in [-1,1]\}$. To train our method, we drew 600 samples from $\mathcal{G}$ and found a corresponding $\theta(0)^{(j)}$ for each sample. We set the parameter space to be $\Theta := \{\theta(0)^{(j)} + \delta \ : \ |\delta| \leq 3, \ j = 1, \ldots, 600\}$. We then uniformly sampled 200,000 points from this set Θ and generated paths for (3.30) from the 600 centers to train V_ξ. We then tested the method on a new set of 100 initials randomly drawn from

$\mathcal{G}$ by following the method outlined in Algorithm 2. We randomly select three from the test set containing the 100 initials and plot the result using our method in Figure 3.4. In addition, Figure 3.3c and 3.3d show the mean and standard deviations of the relative and absolute errors versus time t. We notice that the relative error increases while absolute error decreases: this is because the true solution $u^*(t,\cdot)$ gradually vanishes in time and hence it is easy to cause large relative error even when the absolute error is small.

In this test, it took 2.64 hours to generate $\tilde{G}$ and $\tilde{p}$ for (3.30) and 1.33 hours to generate the trajectories for (3.31). This time cost is significantly higher than the transport equation as the heat equation requires the computation of the Laplacian which is second-order. Once the samples were generated, training V_ξ took approximately 10 minutes. For testing, it took an average of 25 seconds to train a $\theta(0)$ to a sampled g and an average of 2.6 seconds to then solve (3.6) using a 4th order Runge-Kutta solver with step size 0.0001. This amounts to less than 30 seconds in time per initial for the testing stage.

3.4.5 Allen-Cahn equation

In this test, we consider the IVP with nonlinear Allen-Cahn equation given by

$$\begin{cases} \partial_t u(x,t) = \epsilon \Delta u(x,t) + \frac{3}{2}\left(u(x,t) - u(x,t)^3\right), & \forall\, x \in \Omega, t \in (0,T] \\ u(x,0) = g(x), & \forall\, x \in \bar{\Omega}, \end{cases} \tag{3.39}$$

where $\Omega = (-1,1)^2$, $\epsilon = 0.0001$, and the boundary value $u(x,t) = 0$ for all $x \in \partial\Omega$ and $t \in [0,T]$. As the Allen-Cahn PDE does not have an analytical solution to compare against, we resort to the classical implicit-explicit scheme (see e.g. [109]) with a 100×100 grid and

2000 time points to generate a reference solution for comparison in 2D case only, despite that our method can be applied to higher dimensional case. In this test, we use (3.33) with $\alpha(x) = (1 - x_1^2)(1 - x_2^2)$ as our neural network. We let $T_i : \mathbb{R} \to \mathbb{R}$ represent the ith order Chebyshev polynomial. We generate a class of initial conditions

$$\mathcal{G} := \left\{ (1 - x_1^2)(1 - x_2^2) \sum_{k=1}^{m} c_k T_{i_k}(x_1) T_{j_k}(x_2) \ : \ i_k, j_k \in \{0, \ldots, 6\},\ m \le 36,\ |c_k| \le 1 \right\}. \tag{3.40}$$

We see that $\mathcal{G}$ is a space of all combinations of Chebyshev polynomials up to degree 6 multiplied by a boundary function. This set is chosen to represent a diverse spread of initials that can be approximated by our neural network from (3.33). We drew 200 samples from $\mathcal{G}$ and found a corresponding $\theta(0)^{(j)}$ for each sample. Then, as in the case of heat equations above, we generated the parameter set $\Theta := \{\theta(0)^{(j)} + \delta \ : \ |\delta| \le 3,\ j = 1, \ldots, 200\}$. We sampled from Θ uniformly and generated paths from the 200 centers to train V_ξ. We again tested the method on a new set of 100 initials from $\mathcal{G}$. The results of the proposed method at time $t = 0, 0.15, 0.3, 0.45, 0.6$ for three random initials are shown in Figure 3.5. In Figure 3.3e and 3.3f we again plot the mean relative and absolute errors versus time, which demonstrate promising approximation performance of our method.

Figure 3.3e shows some challenges in the relative error as time advances. This is because the solution to the Allen-Cahn equation for this initial value has fast-increasing derivatives as time progresses, which poses a challenge to all numerical methods including ours in solving Allen-Cahn equations in general. Specifically, such large derivatives force the parameters θ of the neural network to blow up quickly, and hence the trajectory $\theta(t)$ may rapidly escape from the prescribed Θ over which we trained the vector field V_ξ. This is a challenge that remains to be overcome by using more adaptive training methods and sampling strategies.

For this experiment, generating $\tilde{G}$ and $\tilde{p}$ for (3.30) took 1.04 hours while the generation of the trajectories for (3.31) took only 15 minutes. The much lower dimension of this problem compared to the transport and heat equation examples accounted for the speed up in the generation of samples. Similar to the transport equation, training V_ξ took only 7 minutes. For testing, it took an average of 21 seconds to train a $\theta(0)$ to a sampled g and an average of 2.1 seconds to then solve (3.6) using a 4th order Runge-Kutta solver with step size 0.002. This amounts to less than 24 seconds in total time per initial for the testing stage.

3.5 Variations and Generalizations

In this section, we briefly discuss modifications of the proposed approach so that it can be applied to some other problems involving evolution PDEs. In particular, we consider the following two cases: general time-dependent PDEs and initial value problems with time-varying boundary conditions.

Applications to general time-dependent PDEs

Our approach can be readily applied to a large variety of time-dependent PDEs. The reason is that these PDEs can be converted to the exact form of (3.2) for which our method is designed. To avoid overloading the bracket notation, we temporarily use $F(u)$ and $F(t,u)$ to represent $F[u]$ and $F_t[u]$ (differential operator that explicitly depends on time t). We first note that one can convert any non-autonomous evolution PDE into an autonomous one:

$$\partial_t u = F(t,u) \quad \Longleftrightarrow \quad \partial_t \tilde{u} = \tilde{F}(\tilde{u}), \quad \text{where} \;\; \tilde{u} := [t; u], \quad \tilde{F}(\tilde{u}) := \big[1; F(u)\big], \tag{3.41}$$

and $[\cdot\,;\,\cdot]$ means to stack the two arguments vertically to form a single one. We can also consider PDEs involving higher order time derivatives and convert them to first-order PDE systems by noticing equivalency as follows:

$$\partial_{tt} u = F(u) \quad \Longleftrightarrow \quad \partial_t \tilde{u} = \tilde{F}(\tilde{u}), \quad \text{where} \;\; \tilde{u} := [u; v], \quad \tilde{F}(\tilde{u}) := \big[v; F(u)\big]. \tag{3.42}$$

History-dependent PDEs can also be considered: denote $H_u(t) := \{u(\cdot, s) \,|\, 0 \le s \le t\}$ the trajectory recording the path of u up to time t and F a nonlinear operator on path H_u, then we can set $H_u(t)$ as an auxiliary variable and convert the problem $\partial_t u = F[H_u]$ to an autonomous evolution PDE of $[u; H_u]$.

Evolution PDEs with boundary conditions

We can also modify our method to solve IVPs with different boundary conditions. Let (g, ϕ) be the pair of initial and boundary values of the IVP. That is, $u(x,0)|_{\bar{\Omega}} = g$ and $u(x,t)|_{\partial\Omega\times[0,T]} = \phi$. In this case, we can parameterize $u_\theta(x) = \varphi_\eta(x) + \alpha(x)\psi_\zeta(x)$ with $\theta = (\eta, \zeta)$, where φ_η and ψ_ζ are two reduced-order models (e.g., neural nets) with parameters η and ζ, respectively, and $\alpha(x)$ is a prescribed smooth function such that $\alpha(x) > 0$ if $x \in \Omega$ and $\alpha(x) = 0$ if $x \in \partial\Omega$. Here φ_η is to fit the boundary value ϕ without interference from $\alpha\psi_\zeta$ as the latter vanishes on the boundary $\partial\Omega$.

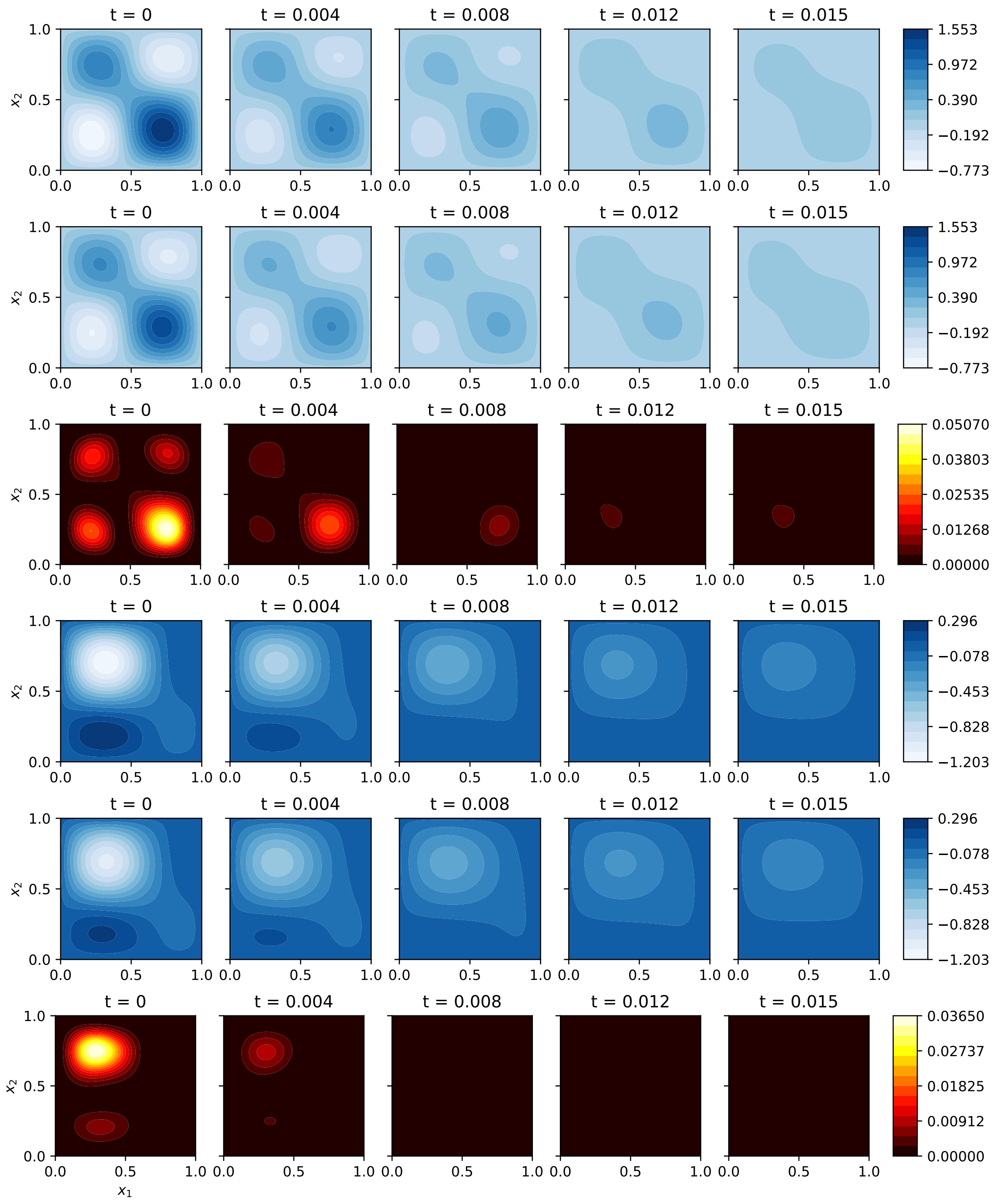


Figure 3.4 (Heat equation). Comparison between true solution $u^*(\cdot, t)$, the approximation $u_{\theta(t)}(\cdot)$ and their pointwise absolute difference $|u_{\theta(t)}(x) - u^*(x,t)|$ for times $t = 0, 0.004, 0.008, 0.012, 0.015$ for IVPs with the first (rows 1–3), second (rows 4–6) and third initial (rows 7–9) drawn from the set $\mathcal{G} := \{\sum_{i=1}^{4} c_i g_i \ : \ c_i \in [-1,1]\}$ where g_i is defined in (3.38).

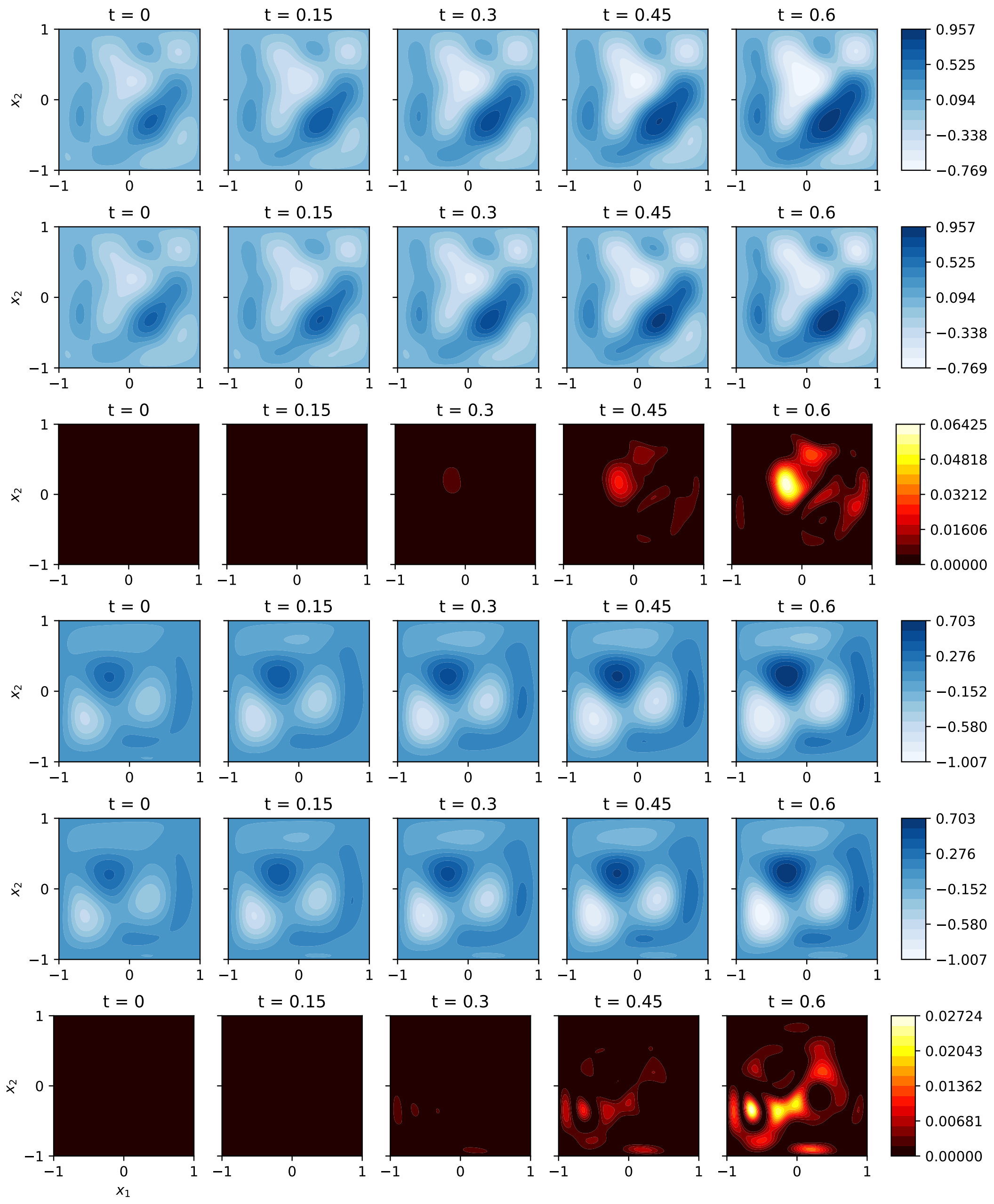


Figure 3.5 (Allen-Cahn equation). Comparison between true solution $u^*(\cdot,t)$, the approximation $u_{\theta(t)}(\cdot)$ and their pointwise absolute difference $|u_{\theta(t)}(x) - u^*(x,t)|$ for times $t = 0, 0.004, 0.008, 0.012, 0.015$ for IVPs with the first (rows 1–3), second (rows 4–6) and third initial (rows 7–9) drawn from the set $\mathcal{G}$ defined in (3.40).

PART 4
EXTENDING NEURAL CONTROL FOR HIGH-DIMENSIONAL PDES

4.1 Introduction

In this last part we will continue searching for ways to find solution operators for evolutional PDEs of the form

$$\begin{cases} \partial_t u(x,t) = F[u](x,t), & x \in \Omega,\ t \in (0,T], \\ \\ u(x,0) = g(x), & x \in \Omega. \end{cases} \tag{4.1}$$

Previously, we had approached approximating the solution operator by solving the nonlinear least squares problem defined as

$$\min_{\xi} \int_{\Theta} \int_{\Omega} |\nabla_\theta u_\theta(x) \cdot V_\xi(\theta) - F[u_\theta](x)|^2 \, dx d\theta. \tag{4.2}$$

This approach turns out to be a natural first choice sharing much of the reasoning used in many deep learning papers such as PINNs [94] or the approach we used in Lyapunov-Net in terms of the negative orbital requirement. Namely, the general idea followed by the least squares problem is that we collect together some relevant penalty terms that we want to be zero and add those to a loss function which we seek to minimize. While the previous chapter shows that in certain cases the total computational cost for solving this least-squares problem is substantially lower than solving the IVP (4.1) directly over and over again, there remain hurdles to using this approach. Chief amongst these, the computational cost in solving the nonlinear least squares problem (4.2) is high due to the relatively high dimension m of Θ. Further, it requires intuition and careful tuning in sampling the θ from Θ to achieve reasonable solution quality. These issues may hinder the practical applications of this least squares approach. As such, we shall develop an extension of this method that builds upon the main ideas developed in the previous chapter but is simpler to implement and use.

The new contributions of this part to the existing literature can be summarized as follows:

1. We develop improve on the Neural Control framework by introducing a new training approach which utilizes Neural-ODE to train the DNNs.

2. We extend the theoretical analysis from the previous chapter to establish error bounds for the proposed method when more general semi-linear PDEs, as well as show new results on approximation capabilities of our previous theory.

3. We conduct a new series of numerical experiments and compare these to the least-squares approach of the previous chapter to show some improvements.

4.2 Proposed Method

We now develop the new approach to estimate ξ which is substantially more efficient and accurate, even without any intuition, in training V_ξ than the alternative of using (4.2).

4.2.1 Control Theory Approach

Consider $\theta(t)$ as a controllable trajectory in the parameter space and recall that when approximating the solution operator it suffices to find a proper control vector field V_ξ with parameters ξ such that

$$\nabla_\theta u_{\theta(t)}(x) \cdot V_\xi(\theta(t)) = \partial_t(u_{\theta(t)}(x)) = F[u_{\theta(t)}](x), \qquad \forall\, x \in \Omega,\ t \in (0,T] \tag{4.3}$$

and the initial $\theta(0)$ to satisfy $u_{\theta(0)} = g$. Realize, enforcing (4.3) for many trajectories $\theta(t)$ is equivalent to minimizing the following running cost

$$r(\theta(t);\xi) := \|\nabla_\theta u_{\theta(t)} \cdot V_\xi(\theta) - F[u_{\theta(t)}]\|^2_{L^2(\Omega)} \tag{4.4}$$

for any trajectory $\theta(t)$ in Θ. Realize a problem of the least-squares approach is that relevant trajectories can be unevenly distributed in Θ for the corresponding $u_{\theta(t)}$ to be a meaningful solution to (4.1), this is a major issue leading to the inefficient sampling of Θ in (4.2). Therefore, we propose the following strategy to learn V_ξ effectively: first, define the auxiliary variable $\gamma(t) := [\theta(t); s(t)] \in \mathbb{R}^{m+1}$ (we use the MATLAB syntax $[\cdot;\cdot]$ to stack vectors as one column vector) such that $\gamma(0) = [\theta(0); 0]$ and

$$\dot\gamma(t) = [\dot\theta(t);\ \dot s(t)] = [V_\xi(\theta(t));\ r(\theta(t);\xi)].$$

Define $\ell(\gamma;\xi) = [0_m; 1]^\top \gamma$, and we see $\ell(\gamma(T);\xi) = \int_0^T r(\theta(t);\xi)dt$. We then sample M initial values $\{g_i :\ i = 1,\dots,M\}$ and fit them by $\{u_{\theta_i(0)} :\ i = 1,\dots,M\}$ correspondingly, then solve the following terminal value control problem with control parameters ξ:

$$\begin{aligned} \min_\xi \quad & \hat\ell(\xi) := \frac{1}{M}\sum_{i=1}^{M} \ell(\gamma_i(T);\xi), \\ \text{subject to} \quad & \dot\gamma_i(t) = [V_\xi(\theta_i(t));\ r(\theta_i(t);\xi)], \quad \gamma_i(0) = [\theta_i(0);\ 0], \qquad i = 1,\dots,M, \end{aligned} \tag{4.5}$$

where each $\theta_i(t)$ starts from its own initial $\theta_i(0)$ fitted earlier. In practice, we shall be required to approximate the L^2 norm within $r(\theta;\xi)$ in (4.4) using Monte Carlo integration as usual.

With the formulation in (4.5), we now show how to calculate gradients of $\hat\ell$ using Neural ODE (NODE) [18]. It suffices to consider the case with $M = 1$. NODE allows us to calculate $\nabla_\xi \ell(\gamma(T), \xi)$ as follows: using the *adjoint sensitivity method*, we introduce the so-called adjoint parameter $a : [0, T] \to \mathbb{R}^{m+1}$ such that $a(T) = -\nabla_\gamma \ell(\gamma(T); \xi)$ and $\dot{a}(t)^\top = -a(t)^\top \nabla_\gamma [V_\xi(\theta(t)); r(\theta(t); \xi)]$. In practice, we compute $[\theta(t); r(\theta(t); \xi)]$ forward in time to T, and then compute $a(t)$ and integrate $a(t)^\top \nabla_\xi [V_\xi(\theta(t)); r(\theta(t); \xi)]$ simultaneously backward in time to calculate $\nabla_\xi \ell(\gamma(T), \xi) = -\int_T^0 a(t)^\top \nabla_\xi [V_\xi(\theta(t)); r(\theta(t); \xi)] dt$. The latter can be used in any standard first-order optimization algorithm, such as stochastic gradient descient method and ADAM [59] to minimize ℓ. This can be readily extended to the case with $M > 1$ and averaged loss $\hat{\ell}$ in (4.5). The training of V_ξ is summarized in Algorithm 3. In this work, we apply ADAM to minimize $\hat{\ell}$ and its parameter settings are provided in Section 4.3.

Algorithm 3 Training neural control V_ξ

```
Require: Reduced-order model structure $u_\theta$ and parameter set $\Theta$. Control vector field
    structure $V_\xi$. Error tolerance $\varepsilon$.
Ensure: Optimal control parameters $\xi$.
 1: Sample $\{\theta_k(0)\}_{k=1}^K$ uniformly from $\Theta$.
 2: Form loss $\hat{\ell}(\xi) = \frac{1}{M}\sum_{i=1}^M \int_0^T \ell(\theta_i(t); \xi) dt$ as in (4.5).
 3: Minimize $\hat{\ell}(\xi)$ with respect to $\xi$ (using Neural ODE to compute $\nabla_\xi \hat{\ell}(\xi)$).
```

Once the vector field V_ξ is trained, we can implement the solution operator $\mathcal{S}_F$ in the same way as in Algorithm 2.

4.2.2 Theoretical Advances and Error Analysis

In this section, we provide theoretical justifications on the solution approximation ability of the proposed method for a general class of second-order semi-linear PDEs. For ease of presentation, we first define Sobolev ball with radius L in $W^{k,\infty}(\Omega)$ as follows.

Definition 2 (Sobolev ball). *For $L > 0$ and $k \in \mathbb{N}$, we define the Sobolev ball of radius L as the set*

$$SB(\Omega, L, k) := \{g \in W^{k,\infty}(\Omega) : \quad \|g\|_{W^{k,\infty}(\Omega)} \leq L\}.$$

As mentioned in Section 4.2.1, we focus on IVPs with solutions compactly supported in an open bounded set $\Omega \subset \mathbb{R}^d$ for ease of discussion. More specifically, we consider the following IVP with an evolution PDE and arbitrary initial value g:

$$\begin{cases} \partial_t u(x,t) = F[u](x,t) & x \in \Omega,\ t \in (0,T) \\ u(x,0) = g(x) & x \in \Omega, \\ u(x,t) = 0 & x \in \partial\Omega,\ t \in [0,T] \end{cases} \tag{4.6}$$

where F is a kth order autonomous, (possibly) nonlinear differential operator. We note that non-autonomous PDEs can be converted to an equivalent autonomous one by augmenting spatial variable x with t in practice. For the remainder of this section, we require the following regularity on F.

Assumption 5. *Let F be a kth order differential operator and $p \in \mathbb{N}$. For all $L > 0$, there exists $M_{p,L,F} > 0$ such that for all $w, v \in SB(\Omega, L, k+p)$ we have*

$$\|F[w] - F[v]\|_{W^{p,\infty}(\Omega)} \leq M_{p,L,F}\|w - v\|_{W^{k+p,\infty}(\Omega)}$$

The requirements of Assumption 5 ensure that $F : W^{k+p,\infty}(\Omega) \to W^{p,\infty}(\Omega)$ is Lipschitz over $SB(\Omega, L, k+p)$ for the specified p. For many types of PDEs, this assumption is quite mild. For example, if the second-order linear/nonlinear differential operator F is C^p in $(x, u(x), \nabla u(x), \nabla^2 u(x))$ for all $x \in \Omega$, then F satisfies Assumption 5 with $k = 2$. With this new assumption we can replace Assumption 4 from the previous chapter with a different theorem.

Theorem 4. *Suppose F satisfies Assumption 5 for $k, p \in \mathbb{N}$. Let $\epsilon > 0$ and $L > 0$. Then there exists a feed-forward neural network u_θ and an open bounded set $\Theta_{u,F,L} \subset \mathbb{R}^m$ such that*

(i) For every $\theta \in \Theta_{u,F,L}$, there exists $\alpha_\theta \in \mathbb{R}^m$ such that

$$\|\alpha_\theta \nabla_\theta u_\theta - F[u_\theta]\|_{W^{1,\infty}(\Omega)} < \epsilon.$$

(ii) For every $g \in W^{2k+3,\infty}(\Omega) \cap SB(\Omega, L, 2k+2)$, there exists $\theta \in \Theta_{u,F,L}$ such that

$$\|u_\theta - g\|_{W^{1,\infty}(\Omega)} < \epsilon.$$

To prove Theorem 4 we will need a few results. First, we prove the following Lemma, which provides a meaningful connection between commonly-used general neural networks with fixed parameters and neural networks with time-evolving parameters. For ease of presentation, we restrict to standard feed-forward networks only, while the idea can be readily generalized to other networks (see Remark 4).

Lemma 6. *For any feed-forward neural network architecture $v_\eta(x,t)$ with parameters denoted by η, there exists a feed-forward neural network $u_{\theta(t)}(x)$ and a differentiable evolution of the parameters $\theta(t)$ such that $u_{\theta(t)}(x) = v_\eta(x,t)$ for all $x \in \mathbb{R}^d$ and $t \in \mathbb{R}$.*

Proof. We denote the feed-forward network $v_\eta(x,t) = w_l^\top h_{l-1}(x,t) + b_l$ with

$$h_\ell(x,t) = \sigma(W_\ell h_{\ell-1}(x,t) + b_\ell), \quad \ell = 1, \ \ldots, \ l-1,$$

where $h_0(x,t) = \sigma(W_0 x + w_0 t + b_0)$ and $\eta = (w_l, b_l, W_{l-1}, \ldots, W_0, w_0, b_0)$. Note that all components in η are constants independent of x and t. Similarly, denote $u_\theta(x) = \tilde{w}_l^\top \tilde{h}_{l-1}(x) + \tilde{b}_l$ with

$$\tilde{h}_\ell(x) = \sigma(\tilde{W}_\ell \tilde{h}_{\ell-1}(x) + \tilde{b}_\ell), \quad \ell = 1, \ \ldots, \ l-1,$$

where $\tilde{h}_0(x) = \sigma(\tilde{W}_0 x + \tilde{b}_0)$ and $\theta = (\tilde{w}_l, \tilde{b}_l, \tilde{W}_{l-1}, \ldots, \tilde{W}_0, \tilde{b}_0)$. Note that $\theta(t)$ can be time-evolving. Hence, we can set $\tilde{b}_0(t) = w_0 t + b_0$ for all t and all other components in $\theta(t)$ as constants identical to the corresponding ones in η, i.e., $\tilde{w}_l(t) \equiv w_l$, $\tilde{b}_L(t) \equiv b_L$, $\ldots$, $\tilde{b}_1(t) \equiv 1_0$. Then it is clear that $u_{\theta(t)}(x) = v_\eta(x,t)$ for all x and t, and $\theta(t)$ is differentiable. $\square$

Remark 4. *While we state Lemma 6 for feed-forward neural networks, it is easy to see that the result holds for general neural networks as well. To see this, note that for any neuron $y(x,t)$ we can rewrite it as $\tilde{y}_{\eta(t)}(x)$ as shown in Lemma 6. For any neuron $z(t,x,y(x,t))$ we can write it as $z(t,x,y(x,t)) = z(t,x,\tilde{y}_{\eta(t)}(x)) = \tilde{z}_{(\eta(t),\tilde{\eta}(t))}(x)$ for some neural network $\tilde{z}_{\eta,\tilde{\eta}}$ with input x and parameters $(\eta, \tilde{\eta})$ evolving in t. Applying this repeatedly justifies the claim.*

We will use the following statement of the universal approximation theorem for neural networks in our later proofs.

Lemma 7 (Theorem 5.1 [23]). *Let $\epsilon > 0$, $L > 0$, and $k \in \mathbb{N}$. There exists a fixed feed-forward neural network architecture w_η with* tanh *activation such that for all $f \in SB(\Omega, L, k+1)$ there exists an $\eta \in \mathbb{R}^m$ such that*

$$\|w_\eta - f\|_{W^{k,\infty}(\Omega)} < \epsilon.$$

Proof. While Theorem 5.1 in [23] is stated for $\Omega = [0,1]^d$ we can apply a transformation layer to generalize this for more arbitrary bounded Ω. Thus Lemma 7 is an immediate corollary to Theorem 5.1 in [23]. $\square$

We now advance to the proof.

Proof of Theroem 4. Let $\delta > 0$ and define $\Omega_\delta = \Omega \times [-\delta, \delta]$. Denote

$$\bar{L} := (1 + (1+\delta)M_{k,L,F})L + (1+\delta)\|F[0]\|_{W^{k+1,\infty}(\Omega)}.$$

For any $g \in W^{2k+3,\infty}(\Omega)$ with $\|g\|_{W^{2k+2,\infty}(\Omega)} \le L$, we define T_g such that $T_g(x,t) = g(x) + tF[g](x)$ for all $(x,t) \in \Omega_\delta$. Note that $T_g(\cdot, 0) \equiv g(\cdot)$ and $\partial_t T_g(\cdot, t) = F[g](\cdot)$ for all t. Furthhermore, there is

$$\begin{aligned}\|T_g\|_{W^{k+2,\infty}(\Omega_\delta)} &= \|g + tF[g]\|_{W^{k+2,\infty}(\Omega_\delta)} \\ &\le \|g\|_{W^{k+2,\infty}(\Omega_\delta)} + \|F[g]\|_{W^{k+1,\infty}(\Omega)} + \delta\|F[g]\|_{W^{k+2,\infty}(\Omega)} \\ &\le \|g\|_{W^{k+2,\infty}(\Omega_\delta)} + (1+\delta)\|F[g]\|_{W^{k+2,\infty}(\Omega)} \\ &=: \mathrm{I} + (1+\delta)\cdot \mathrm{II},\end{aligned} \tag{4.7}$$

where the first equality in (4.7) is due to the definition of T_g, the first inequality due to the triangle inequality in $W^{k+2,\infty}(\Omega_\delta)$, the split of time and spatial derivatives on T_g, and $|t| \le \delta$, and the second inequality due to $\|F[g]\|_{W^{k+1,\infty}(\Omega_\delta)} \le \|F[g]\|_{W^{k+2,\infty}(\Omega_\delta)}$. Since $g \in C^{2k+2}(\Omega)$, we have

$$\mathrm{I} := \|g\|_{W^{k+2,\infty}(\Omega_\delta)} \le \|g\|_{W^{2k+3,\infty}(\Omega)} \le L \tag{4.8}$$

given that g is constant in t. On the other hand, there is

$$\begin{aligned}\mathrm{II} := \|F[g]\|_{W^{k+2,\infty}(\Omega)} &\leq \|F[g]-F[0]\|_{W^{k+2,\infty}(\Omega)} + \|F[0]\|_{W^{k+2,\infty}(\Omega)} \\ &\leq M_{k,L,F}\|g\|_{W^{2k+2,\infty}(\Omega)} + \|F[0]\|_{W^{k+2,\infty}(\Omega)} \qquad (4.9) \\ &\leq M_{k,L,F}L + \|F[0]\|_{W^{k+2,\infty}(\Omega)}\end{aligned}$$

where the second inequality is due to the Assumption 5 on F. Combining (4.7), (4.8) and (4.9) yields $\|T_g\|_{W^{k+2,\infty}(\Omega_\delta)} \leq \bar{L}$. By Lemma 7, there exists a feedforward neural network architecture v_η such that for any T_g there are parameters $\eta = \eta_g$ for some η_g satisfying

$$\|v_{\eta_g} - T_g\|_{W^{k+1,\infty}(\Omega_\delta)} < \frac{\epsilon}{2(1+M_{k,L,F})}. \tag{4.10}$$

Define $w_g := v_{\eta_g} - T_g$, then $w_g \in W^{k+2,\infty}(\Omega_\delta)$ since v_{η_g} is smooth in (x,t) and $T_g(\cdot,t) = g(\cdot)+tF[g](\cdot) \in W^{k+2,\infty}(\Omega_\delta)$. As a result, for any multi-index $\alpha \in \mathbb{N}^{d+1}$ satisfying $|\alpha| \leq k+1$, there is $\partial^\alpha w_g \in W^{1,\infty}(\Omega_\delta)$, which also implies $\partial^\alpha w_g \in C(\Omega_\delta)$. Due to this continuity, we also have

$$\|\partial^\alpha w_g(\cdot,0)\|_{L^\infty(\Omega)} \leq \|\partial^\alpha w_g\|_{L^\infty(\Omega_\delta)}. \tag{4.11}$$

Since α is arbitrary, we thus have

$$\|w_g(\cdot,0)\|_{W^{p,\infty}(\Omega)} \leq \|w_g\|_{W^{p,\infty}(\Omega_\delta)} \tag{4.12}$$

$$\|\partial_t w_g(\cdot,0)\|_{W^{p-1,\infty}(\Omega)} \leq \|\partial_t w_g\|_{W^{p-1,\infty}(\Omega_\delta)} \tag{4.13}$$

for all $p = 1, \dots, k+1$. Therefore,we have

$$\begin{aligned}\|v_{\eta_g}(\cdot,0) - g(\cdot)\|_{W^{k+1,\infty}(\Omega)} &= \|w_g(\cdot,0)\|_{W^{k+1,\infty}(\Omega)} \le \|w_g\|_{W^{k+1,\infty}(\Omega_\delta)} \\ &= \|v_{\eta_g} - T_g\|_{W^{k+1,\infty}(\Omega_\delta)} < \frac{\epsilon}{2(1+M_{k,L,F})},\end{aligned} \tag{4.14}$$

where the two equalities are due to the definition of w_g, the first inequality due to (4.12) with $p = k+1$, and last inequality due to (4.10), Note that (4.14) further implies that

$$\|F[v_{\eta_g}](\cdot,0) - F[g](\cdot)\|_{W^{1,\infty}(\Omega)} \le M_{k,L,F}\|v_{\eta_g}(\cdot,0) - g(\cdot)\|_{W^{k+1,\infty}(\Omega)} < \frac{\epsilon}{2}. \tag{4.15}$$

Combining the results above, we have

$$\begin{aligned}\|\partial_t v_{\eta_g}(\cdot,0) - F[v_{\eta_g}](\cdot,0)\|_{W^{1,\infty}(\Omega)} &\le \|\partial_t v_{\eta_g}(\cdot,0) - F[g](\cdot)\|_{W^{1,\infty}(\Omega)} + \|F[g](\cdot) - F[v_{\eta_g}](\cdot,0)\|_{W^{1,\infty}(\Omega)} \\ &< \|\partial_t w_g(\cdot,0)\|_{W^{1,\infty}(\Omega)} + \frac{\epsilon}{2} \\ &\le \|\partial_t w_g\|_{W^{1,\infty}(\Omega_\delta)} + \frac{\epsilon}{2} \\ &\le \|v_{\eta_g} - T_g\|_{W^{k+1,\infty}(\Omega_\delta)} + \frac{\epsilon}{2} \\ &\le \epsilon.\end{aligned} \tag{4.16}$$

where the first inequality above is due to the triangle inequality, the second due to the definition of w_g, the property $\partial_t T_g(\cdot,0) = F[g](\cdot)$, and (4.15), the third due to (4.13) with $p = 2$, the fourth due to $\|v_{\eta_g} - T_g\|_{W^{k+1,\infty}(\Omega_\delta)}$ being the max of terms including $\|\partial_t(v_{\eta_g} - T_g)\|_{W^{1,\infty}(\Omega_\delta)}$, and the last inequality due to (4.14). Since v_η uses tanh activations and hence is smooth in η, we know both (4.14) and (4.16) hold for some open neighborhood U_g of η_g

in $\mathbb{R}^m$.

Following the ideas in the proof of Lemma 6 we define the sets $\hat{U}_g$ where for each $\eta = (w_l, \dots, w_0, b_0) \in U_g$ we have $\theta = (w_l, \dots, b_0) \in \hat{U}_g$, since U_g is open we can easily conclude that $\hat{U}_g$ is also open.

Now we set $\Gamma := \bigcup_{g \in W^{2k+3,\infty}(\Omega) \cap SB(\Omega, L, 2k+2)} \hat{U}_g$, which is open, and $\Theta_{u,L,F} = \Gamma \cap B_R(0)$ where $B_R(0) := \{\theta \in \mathbb{R}^m : \ |\theta| < R\}$ is a bounded open set for some $R > 0$. Therefore $\Theta_{u,L,F}$ is also bounded and open. Moreover by Lemma 6, for any $\theta \in \Theta_{u,L,F}$, there exists $\eta \in U_g$ for some g and a differentiable $\hat{\theta}(t)$, such that $u_{\hat{\theta}(t)}(\cdot) = v_\eta(\cdot, t)$ for all t with $\hat{\theta}(0) = \theta$. Therefore,

$$\partial_t v_\eta(\cdot, 0) = \partial_t u_{\hat{\theta}(t)}(\cdot)|_{t=0} = \nabla_\theta u_{\hat{\theta}(0)}(\cdot) \cdot \dot{\hat{\theta}}(0) = \nabla_\theta u_\theta(\cdot) \cdot \dot{\hat{\theta}}(0).$$

Let $\alpha_\theta = \dot{\hat{\theta}}(0)$. Given that (4.16) holds for v_η and $u_\theta(\cdot) = v_\eta(\cdot, 0)$, we know that

$$\|\nabla_\theta u_\theta \cdot \alpha_\theta - F[u_\theta]\|_{W^{1,\infty}(\Omega)} = \|\partial_t v_\eta(\cdot, 0) - F[v_\eta](\cdot, 0)\|_{W^{k+1,\infty}(\Omega_\delta)} < \epsilon,$$

which justifies the first claim.

(ii) For any $g \in W^{2k+2,\infty}(\Omega) \cap SB(\Omega, L, 2k+1)$, we know there exists $\eta_g \in U_g$, such that $v_{\eta_g}(\cdot, 0) = g(\cdot)$. Given the definition of $\Theta_{u,F,L}$ above, there exists $\theta \in \Theta_{u,F,L}$ and a differentiable curve $\hat{\theta}(t)$ such that $\hat{\theta}(0) = \theta$ and $u_{\hat{\theta}(t)}(\cdot) = v_{\eta_g}(\cdot, t)$ for all t. Therefore, given that (4.14) holds for v_{η_g}, we have

$$\|u_\theta - g\|_{W^{1,\infty}(\Omega)} = \|u_{\hat{\theta}(0)} - g\|_{W^{1,\infty}(\Omega)} = \|v_{\eta_g}(\cdot, 0) - g(\cdot)\|_{W^{1,\infty}(\Omega)} < \epsilon,$$

which proves the second claim. $\square$

Theorem 4 tells us that with a sufficiently large neural network, we can find a subset $\Theta_{u,F,L}$ of the parameter space such that the tangent spaces of $\Theta_{u,F,L}$ can approximate $F[u_\theta]$ arbitrarily well. Realize, this theorem allows us to derive a stronger result when compared to Assumption 4 from the previous chapter and only requires regularity on the operator F and the approximating power of neural networks. With Theorem 4 we can now state a similar existence result as in Proposition 1 of vector fields defined over $\Theta_{u,F,L}$ that have the approximation capabilities proved in Theorem 4.

Proposition 2. *Suppose Assumption 5 holds. Then for any $\varepsilon > 0$ and $L > \epsilon > 0$, there exists a differentiable vector field parameterized as a neural network $V_\xi : \bar{\Theta}_{u,F,L} \to \mathbb{R}^m$ with parameters ξ, such that*

$$\|V_\xi(\theta) \cdot \nabla_\theta u_\theta - F[u_\theta]\|_{H^1(\Omega)} \leq \varepsilon,$$

for all $\theta \in \bar{\Theta}_{u,F,L}$.

Due to the similarity of the proof of Proposition 2 to that of Proposition 1, we have moved the exposition to Appendix B. Now we provide a few mild conditions on the problem setting of the PDE and its solutions, as follows.

Assumption 6. *Let Ω be a bounded open set of $\mathbb{R}^d$. The solutions to* (4.1) *are compactly supported within Ω for all $t \in [0, T]$.*

Of particular importance, Assumption 6 allows us to be able to design u_θ so that $\nabla u_\theta(x) = 0$ and $u_\theta(x) = 0$ on the boundary and as such matches the true solutions on the boundaries as well (see e.g. [12, 26, 34] for more discussion on this point).

Assumption 7. *Let* (4.1) *be a second-order semi-linear PDE with* $F[u](x) := (\sigma^2(x)/2)\Delta u(x) + f(x, u(x), \nabla u(x))$ *for some (possibly) fully nonlinear function* f *such that the following inequalities hold for all* $x, y \in \Omega$, $u, v \in \mathbb{R}$ *and* $p, q \in \mathbb{R}^d$*:*

$$|f(x,u,p) - f(y,v,q)| \le L_f(|x-y| + |u-v| + |p-q|) \tag{4.17a}$$

$$|\partial_x f(x,u,p) - \partial_x f(y,v,q)| \le L_x(|x-y| + |u-v| + |p-q|) \tag{4.17b}$$

$$|\partial_u f(x,u,p) - \partial_u f(y,v,q)| \le L_u(|x-y| + |u-v| + |p-q|) \tag{4.17c}$$

$$|\partial_p f(x,u,p) - \partial_p f(y,v,q)| \le L_p(|x-y| + |u-v| + |p-q|) \tag{4.17d}$$

for some $L_f, L_x, L_u, L_p > 0$*. Here* $|\cdot|$ *denotes absolute value or standard Euclidean vector norm.*

It should be noted that F as defined in Assumption 7 satisfies the requirements of Assumption 5, and thus we can use the results of Proposition 2. Now we are ready to present the main theorem of the present chapter.

Theorem 5. *Suppose Assumptions 6 and 7 hold. Let* $\varepsilon_0 \ge \varepsilon > 0$ *be arbitrary. Then for any* $L > \varepsilon > 0$ *there exist a vector field* V_ξ *and a constant* $C > 0$ *depending only on* F, u_θ, *and* V_ξ, *such that for any* $u^* \in SB(\Omega, L, 2)$ *satisfying the evolution PDE in* (4.1), *Assumption 6 and* $u^*(\cdot, 0) \in SB(\Omega, L, 4)$ *there is*

$$\|u_{\theta(t)}(\cdot) - u^*(\cdot, t)\|_{H^1(\Omega)} \le \sqrt{2}(\varepsilon_0 + 2\varepsilon t)e^{4Ct} \tag{4.18}$$

for all t *as long as* $\theta(t) \in \bar{\Theta}_{u,F,L}$, *where* $\theta(t)$ *is solved from the ODE defined by* (3.6) *with* V_ξ

and initial $\theta(0)$ *satisfying* $\|u_{\theta(0)}(\cdot) - u^*(\cdot, 0)\|_{L^2(\Omega)} \le \varepsilon_0$.

Proof. Due to Assumptions 7 and 5, there exist a vector field V_ξ and a bounded open set $\Theta_{u,F,L}$ that satisfy the results of Proposition 2. Assume $\theta(t) \in \Theta_{u,F,L}$ is the solution to (3.6), and $\|u_{\theta(0)}(\cdot) - u^*(\cdot,0)\|_2 \le \varepsilon_0$ (such $\theta(t)$ is guaranteed to exist for $\varepsilon \le \varepsilon_0$), where u^* is the true solution of the PDE (4.1). As $u^* \in SB(\Omega, L, 2)$, there exists $M_f := \|\partial_u f\|_{L^\infty(\Omega \times [-L,L] \times [-L,L]^d)}$ because $u^* \in SB(\Omega, L, 2)$ which implies $\partial_u f(x, u^*(x,t), \nabla u^*(x,t)) \le M_f$ a.e. in Ω for every t. Similarly, by Assumption 7 we have $\partial_p f$ Lipschitz continuous jointly in its arguments and thus there is $\nabla \cdot \partial_p f(x, u^*(x,t), \nabla u^*(x,t)) \le M_f'$ a.e. in Ω for some $M_f' > 0$.

In what follows, we denote $\|\cdot\|_2 := \|\cdot\|_{L^2(\Omega)}$ and

$$u_t := u_{\theta(t)}, \quad v_t := u_t^*, \quad e_0(\cdot,t) := u_t(\cdot) - v_t(\cdot), \quad e_1(\cdot,t) := \nabla u_t(\cdot) - \nabla v_t(\cdot)$$

to simplify the notations hereafter. Define

$$E(t) = \frac{1}{2}\int_\Omega |u_t(x) - v_t(x)|^2\,dx + \frac{1}{2}\int_\Omega |\nabla u_t(x) - \nabla v_t(x)|^2\,dx = \frac{\|e_0(\cdot,t)\|_2^2}{2} + \frac{\|e_1(\cdot,t)\|_2^2}{2}.$$

Taking the time derivative of E, we find

$$\begin{aligned}\dot{E}(t) =& \ \langle e_0(\cdot,t), \partial_t e_0(\cdot,t)\rangle + \langle e_1(\cdot,t), \partial_t e_1(\cdot,t)\rangle \\ =& \ \langle u_t - v_t, F[u_t] - F[v_t]\rangle + \langle \nabla u_t - \nabla v_t, \nabla F[u_t] - \nabla F[v_t]\rangle + \langle u_t - v_t, \epsilon_t\rangle + \langle \nabla u_t - \nabla v_t, \nabla \epsilon_t\rangle \\ =:& \ \mathrm{I} + \mathrm{II} + \mathrm{III} + \mathrm{IV}, \end{aligned} \tag{4.19}$$

where $\langle\cdot,\cdot\rangle$ is the inner product in the $L^2(\Omega)$ space, $\epsilon_t(\cdot) := \nabla_\theta u_{\theta(t)}(\cdot) \cdot V_\xi(\theta(t)) - F[u_{\theta(t)}](\cdot) = \partial_t u_t(\cdot) - F[u_t](\cdot)$, and the terms III and IV result from $\partial_t u_t = F[u_t] + \epsilon_t$. We shall now proceed to estimate the four terms above.

For the term I, we first notice that

$$\begin{aligned}
\mathrm{I}_1 &:= \left\langle u_t - v_t, \frac{\sigma^2}{2}\left(\Delta u_t - \Delta v_t\right)\right\rangle \\
&= \int_\Omega \frac{\sigma^2(x)}{2} e_0(x,t)\Delta e_0(x,t)dx = \int_\Omega \frac{\sigma^2(x)}{2}\left(\nabla\cdot(e_0(x,t)\nabla e_0(x,t)) - |\nabla e_0(x,t)|^2\right)dx \\
&\le \int_{\partial\Omega} \frac{\sigma^2(x)}{2} e_0(x,t)\nabla e(x,t)\cdot\vec{n}(x)\,dS(x) - \int_\Omega (\sigma(x)\nabla\sigma(x))\cdot(e_0(x,t)\nabla e_0(x,t))dx \\
&\le M_\sigma \|e_0(\cdot,t)\|_2 \|\nabla e_0(t,\cdot)\|_2,
\end{aligned} \tag{4.20}$$

where $M_\sigma := \|\sigma\nabla\sigma\|_{L^\infty} < \infty$, $\vec{n}(x)$ is the unit outer normal at $x \in \partial\Omega$, and the first inequality is due to $\int_\Omega \frac{\sigma^2}{2}|\nabla e_0|^2 \ge 0$. Furthermore, we can show a bound for the quantity:

$$\mathrm{I}_2 := \langle e_0(\cdot,t), f(\cdot,u_t,\nabla u_t) - f(\cdot,v_t,\nabla v_t)\rangle \le L_f\|e_0(\cdot,t)\|_2(\|e_0(\cdot,t)\|_2 + \|\nabla e_0(\cdot,t)\|_2), \tag{4.21}$$

where we used the Lipschitz bound on f to arrive at the inequality. Realize $\mathrm{I} = \mathrm{I}_1 + \mathrm{I}_2$ and so

$$\mathrm{I} \le (M_\sigma + L_f)\|e_0(\cdot,t)\|_2\|\nabla e_0(t,\cdot)\|_2 + L_f\|e_0(\cdot,t)\|_2^2. \tag{4.22}$$

Next, consider the term

$$\begin{aligned}
\left\langle \nabla u_t - \nabla v_t, \nabla(\frac{\sigma^2}{2}\Delta u_t - \frac{\sigma^2}{2}\Delta v_t)\right\rangle = &\int_{\partial\Omega} \frac{\sigma^2(x)}{2}\Delta e_0(x,t)\nabla e_0(x,t)\cdot\vec{n}(x)\,dS(x) \\
&- \int_\Omega \frac{\sigma^2(x)}{2}(\Delta e_0(x,t))^2 dx \\
\le 0, &
\end{aligned} \tag{4.23}$$

where the inequality comes as $\nabla e_0(x,t) = 0$ on the boundary by Assumption 6. Now we define

$$\begin{aligned}
\mathrm{II}_2 &= \langle \nabla u_t - \nabla v_t, \nabla[f(\cdot, u_t, \nabla u_t) - f(\cdot, v_t, \nabla v_t)]\rangle \\
&= \langle \nabla e_0(\cdot, t), \partial_x f(\cdot, u_t, \nabla u_t) - \partial_x f(\cdot, v_t, \nabla v_t)\rangle \\
&\qquad + \langle \nabla e_0(\cdot, t), (\partial_u f(\cdot, u_t, \nabla u_t)\nabla u_t - \partial_u f(\cdot, v_t, \nabla v_t)\nabla v_t)\rangle \\
&\qquad + \langle \nabla e_0(\cdot, t), (\nabla^2 u_t \partial_p f(\cdot, u_t, \nabla u_t) - \nabla^2 v_t \partial_p f(\cdot, v_t, \nabla v_t))\rangle \\
&= \mathrm{II}_{2,1} + \mathrm{II}_{2,2} + \mathrm{II}_{2,3}.
\end{aligned} \tag{4.24}$$

We now estimate these three terms using our assumptions on the function f. First

$$\begin{aligned}
\mathrm{II}_{2,1} &\le L_x \int_\Omega \nabla e_0(x, t)(|u_t(x) - v_t(x)| + |\nabla u_t(x) - \nabla v_t(x)|)dx \\
&\le L_x(\|\nabla e_0(\cdot, t)\|_2 \|e_0(\cdot, t)\|_2 + \|\nabla e_0(\cdot, t)\|_2^2).
\end{aligned}$$

Since $\bar{\Theta}_{u,F,L}$ is compact and $\theta(t)$ is bounded, it is easy to see there exists a constant $M_u > 0$ such that $\|u_t\|_{W^{2,\infty}(\Omega)} \le M_u$ where M_u depends only on $\bar{\Theta}_{u,F,L}$ and the architecture of u_t. Therefore,

$$\begin{aligned}
\mathrm{II}_{2,2} &= \langle \nabla e_0(\cdot, t), (\partial_u f(\cdot, u_t, \nabla u_t) - \partial_u f(\cdot, v_t, \nabla v_t))\, \nabla u_t + \partial_u f(\cdot, v_t, \nabla v_t)\nabla e_0(\cdot, t)\rangle \\
&\le M_u L_u \|\nabla e_0(\cdot, t)\|_2 (\|e_0(\cdot, t)\|_2 + \|\nabla e_0(\cdot, t)\|_2) + M_f \|\nabla e_0(\cdot, t)\|_2^2.
\end{aligned}$$

Finally, we have

$$\begin{aligned}
\mathrm{II}_{2,3} &= \langle \nabla e_0(\cdot, t), \nabla^2 u_t \partial_p f(\cdot, u_t, \nabla u_t) - \nabla^2 v_t \partial_p f(\cdot, v_t, \nabla v_t)\rangle \\
&= \langle \nabla e_0(\cdot, t), \nabla^2 u_t(\partial_p f(\cdot, u_t, \nabla u_t) - \partial_p f(\cdot, v_t, \nabla v_t))\rangle \\
&\qquad + \langle \nabla e_0(\cdot, t), (\nabla^2 e_0(\cdot, t))\partial_p f(\cdot, v_t, \nabla v_t)\rangle.
\end{aligned}$$

Now we see

$$\langle \nabla e_0(\cdot,t), \nabla^2 u_t(\partial_p f(\cdot,u_t,\nabla u_t) - \partial_p f(\cdot,v_t,\nabla v_t))\rangle \le M_u L_p \|\nabla e_0(\cdot,t)\|_2(\|e_0(\cdot,t)\|_2 + \|\nabla e_0(\cdot,t)\|_2) \tag{4.25}$$

and

$$\begin{aligned}\langle \nabla e_0(\cdot,t), (\nabla^2 e_0(\cdot,t))\partial_p f(\cdot,v_t,\nabla v_t)\rangle &= \int_\Omega \partial_p f(x,v_t(x),\nabla v_t(x))^\top \nabla^2 e_0(x,t)\nabla e_0(x,t)dx \\ &= \int_\Omega \partial_p f(x,v_t(x),\nabla v_t(x))^\top \nabla\left(\frac{1}{2}|\nabla e_0(x,t)|^2\right)dx \\ &= -\frac{1}{2}\int_\Omega \nabla\cdot(\partial_p f(x,v_t(x),\nabla v_t(x)))|\nabla e_0(x,t)|^2 dx \\ &\le \frac{d}{2}M_f'\|\nabla e_0(\cdot,t)\|_2^2,\end{aligned} \tag{4.26}$$

where we used the fact that $\nabla e_0(\cdot,t) = 0$ on $\partial\Omega$ to get $\int_{\partial\Omega}\partial_p f|\nabla e_0|^2 dS(x) = 0$. Applying (4.25) and (4.26) in $\mathrm{II}_{2,3}$, we find

$$\mathrm{II}_{2,3} \le M_u L_p\|\nabla e_0(\cdot,t)\|_2\|e_0(\cdot,t)\|_2 + \left(M_u L_p + \frac{d}{2}M_f'\right)\|\nabla e_0(\cdot,t)\|_2^2.$$

Define

$$C := L_x + M_u L_u + M_u L_p + \max\left\{M_\sigma + L_f,\ M_f + \frac{d}{2}M_f'\right\},$$

and we can see

$$\begin{aligned}\mathrm{I} + \mathrm{II} &\le C(\|e_0(\cdot,t)\|_2^2 + \|e_0(\cdot,t)\|_2\|\nabla e_0(\cdot,t)\|_2 + \|\nabla e_0(\cdot,t)\|_2^2) \\ &\le 2C\left(\|e_0(\cdot,t)\|_2^2 + \|\nabla e_0(\cdot,t)\|_2^2\right).\end{aligned} \tag{4.27}$$

Now, using our bound on ϵ_t, we can bound III and IV:

$$\begin{aligned}\mathrm{III}+\mathrm{IV} &= \langle e_0(\cdot,t), \epsilon(\cdot,t)\rangle + \langle \nabla e_0(\cdot,t), \nabla\epsilon(\cdot,t)\rangle \\ &\le \bar{\varepsilon}(\|e_0(\cdot,t)\|_2 + \|\nabla e_0(\cdot,t)\|_2) \\ &\le \sqrt{2}\bar{\varepsilon}\sqrt{\|e_0(\cdot,t)\|_2^2 + \|\nabla e_0(\cdot,t)\|_2^2}.\end{aligned} \tag{4.28}$$

Applying (4.27) and (4.28) to (4.19) and dividing both sides by $E(t)$, we see

$$\frac{d}{dt}\sqrt{E(t)} = \frac{\dot{E}(t)}{\sqrt{E(t)}} \le 4C\sqrt{E(t)} + 2\bar{\varepsilon}.$$

Applying Grönwall's inequality, we conclude

$$\sqrt{E(t)} \le (\sqrt{E(0)} + 2\bar{\varepsilon}t)e^{4Ct} = (\varepsilon_0 + 2\bar{\varepsilon}t)e^{4Ct}.$$

The above inequality holds for any $u^*(\cdot,0) \in SB(\Omega, L, 4)$ by Proposition 2, and so the proof is completed. □

Remark 5. *Theorem 5 provides an upper bound on the error between $u_{\theta(t)}$ and $u^*(\cdot,t)$: it depends on the projection error ε and initial error ε_0 linearly, while it grows exponentially fast in t which is expected. Note that Theorem 5 extends Theorem 3.6 in [34] from PDEs that are linear in the first- and second-order terms to general semi-linear PDEs. This extension requires substantial efforts and new proof steps.*

4.3 Numerical Results

4.3.1 Implementation

Throughout all experiments, we select the following special form for our neural network $V_\xi : \mathbb{R}^m \to \mathbb{R}^m$,

$$V_\xi(\theta) = \phi^{(s)}_{\xi_s}(\theta) \left(\phi^{(r)}_{\xi_r}(\theta) + \phi^{(e)}_{\xi_e}(\theta) \odot \theta \right), \tag{4.29}$$

where $\xi = (\xi_s, \xi_r, \xi_e)$, $\odot$ is component wise multiplication, and $\phi^{(s)} : \Omega \to \mathbb{R}$ is a sigmoid feedforward network, $\phi^{(e)} : \Omega \to \mathbb{R}^d$ is a ReLU feedforward network, and $\phi^{(r)} : \Omega \to \mathbb{R}^d$ is a ReLU ResNet, each of depth L and constant width w as shown in Table 4.1. We decided to use the structure above in order to represent both linear functions through $\phi^{(r)}$ and exponential functions through $\phi^{(e)}$, while $\phi^{(s)}$ acts as a scaling network to handle the varying velocities the different regions of the parameter space may use. It appears to perform better than the other few generic networks we tested.

The loss function (4.5) records the total error accumulated along the trajectories. Sometimes it can be further augmented by an additional loss containing relative errors to target values obtained by, e.g., [26]. In [26], the goal is to solve a given PDE (not to approximate solution operator) but it provides potential target locations $\bar{\theta}(T)$ by solving a sequence of linear least squares problems from a given $\bar{\theta}(0)$. We leverage this method to generate an additional set $\mathcal{T} = \{\bar{\theta}_i(T) : \ i = 1, \ldots, |\mathcal{T}|\}$, and add the following augmentation term to the loss function (4.5):

$$\ell_{aug}(\xi) := \frac{1}{|\mathcal{T}|} \sum_{i=1}^{|\mathcal{T}|} \Big(\ell(\gamma_i(T); \xi) + \frac{\|u_{\theta_i(T)} - u_{\bar{\theta}_i(T)}\|_2^2}{\|u_{\bar{\theta}_i(T)}\|_2^2} \Big), \tag{4.30}$$

Table 4.1 The problem dimension d, the size of augument set $\mathcal{T}$, the mini-batch size K in training, and the width and depth of V_ξ used in each of the numerical experiments.

Experiment	d	$\lvert\mathcal{T}\rvert$	K	Width	Depth
Heat Equation	10	0	100	1000	5
Hyperbolic PDE	10	1000	100	600	5
HJB Equation	8	250	512	1000	5

where the first term in the sum is due to the same logic in (4.5) by starting from $\theta_i(0) = \bar{\theta}_i(0)$ for all i and the second term is to match the targets computed by [26]. Here $\theta_i(T)$ is the first m components of $\gamma_i(T)$. Since time-marching is very slow, we only add a small sized $\mathcal{T}$ for some of our experiments. Nevertheless, they help to further improve approximation of V_ξ in certain experiments. Finally, we note that the empirical estimations in the relative errors are on $L^2(\Omega)$ norm and thus are again approximated by Monte Carlo integration over Ω. Other parameters are also shown in Table 4.1.

For all the experiments, we use the standard ADAM optimizer [59] with learning rate 0.0005, $\beta_1 = 0.9$, $\beta_2 = 0.999$. We terminate the training process when the percent decrease of the loss is less than 0.1% averaged over the past 20 steps or 10,000 total iterations whichever comes first. Once V_ξ is learned, we use the adaptive Dormand-Prince (DOPRI) method to approximate $\theta(t)$ from the ODE in (3.6). All the implementations and experiments are performed using PyTorch in Python 3.9 in Windows 10 OS on a desktop computer with an AMD Ryzen 7 3800X 8-Core Processor at 3.90 GHz, 16 GB of system memory, and an Nvidia GeForce RTX 2080 Super GPU with 8 GB of graphics memory.

4.3.2 Comparison to Existing Methods

First, we consider an initial value problem corresponding to the 10D heat equation ($d = 10$):

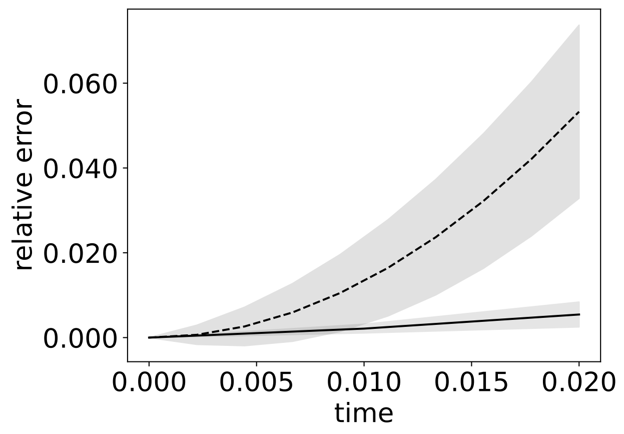
$$\partial_t u(x,t) = \Delta u(x,t), \quad \forall\, x \in \Omega, t \in [0,T] \tag{4.31}$$

with initial value $u(x,0) = g(x)$, where $\Omega = (-1,1)^{10} \subset \mathbb{R}^{10}$ and we use periodic boundary conditions. Note that this is beyond the restrictions we used in error analysis, which suggests that the analysis may be extended to more general cases. As most of the initial conditions result in rapid diffusion, we use $T = 0.1$ in this test. We define our neural network as follows:

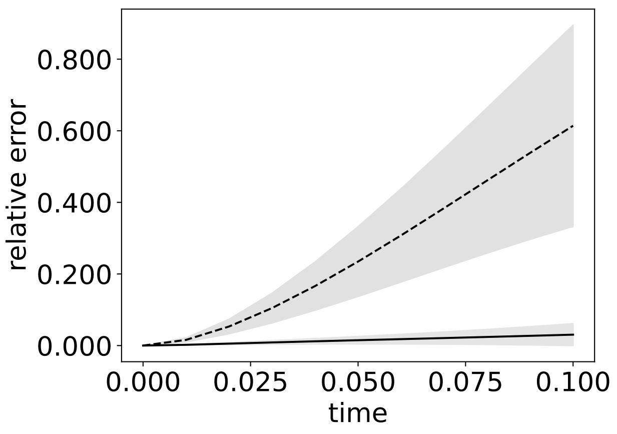
$$u_\theta(x) = \sum_{i=1}^{80} c_i \tanh\left(a_i^\top \sin\left(\pi(x-\beta)\right) - b_i\right). \tag{4.32}$$

We choose this network with trainable parameters $\theta = (\beta, a_i, b_i, c_i)$ with $i = 1, \dots, 80$, $a_i, \beta \in \mathbb{R}^d$ and $b_i, c_i \in \mathbb{R}$ which enforce the periodic boundary conditions as $\sin(\pi(1-p)) = \sin(\pi(-1-p))$ for any $p \in \mathbb{R}$ and the sin function above acts component-wisely. Hence, $\theta \in \mathbb{R}^m$ has dimension $m = 970$. We train the vector field V_ξ using the nonlinear least squares (NLS) method in [34] and the proposed method (Alg. 3) and test their performance on randomly generated initial conditions. Specifically, we generate a set of 100,000 points sampled uniformly from $\Theta := \{\theta \in \mathbb{R}^m : \ |\theta| \le 20\}$ and another 50,000 from $N(0_m, 0.5I_m)$, use them as the initials to train the control field V_ξ defined in (4.29) for both of the compared methods.

After the training is completed, we generate 100 random initials $g = u_{\theta(0)}$ where $\theta(0) \sim N(0_m, 0.5I_m)$. For each g, we compute the ODE (3.6) using the trained V_ξ of the comparison methods, obtain an approximate solution $u_{\theta(t)}$, and compute the relative error $\|u_{\theta(t)}(\cdot) -$



(a) $T = 0.02$



(b) $T = 0.10$

Figure 4.1 Comparison of NLS [34] method (black dashed line) and the proposed method (black solid line). While the previous method only allows accurate results for a small time scale $T = 0.02$, the new method allows more accuracy for longer time scales $T = 0.1$.

$u^*(\cdot,t)\|_2^2/\|u^*(\cdot,t)\|_2^2$ where the true solution is given by $u^*(x,t) = \int g(y)N(y-x, 2tI_d)\,dy$ [30], where N stands for the density of Gaussian. Then we show the average and standard deviation of the relative errors over these 100 initials versus time t in Figure 4.1.

Figure 4.1 shows that Alg. 3 significantly improves the approximation accuracy compared to the NLS method provided by [34]. Specifically, V_ξ obtained by NLS yields reasonably accurate solutions up to $t = 0.01$ with $< 2\%$ relative error, but the error grows unacceptably large to 60% on average at $t = 0.1$. On the other hand, the vector field V_ξ trained by Alg. 3 yields much more stable and accurate solution $u_{\theta(t)}$. The relative error maintains to be lower than 0.3% at $t = 0.01$ and 4% for the proposed method at $t = 0.1$. We also show the approximation $u_{\theta(t)}$ obtained by NLS and the proposed method for $t = 0.02$ in Figure 4.2. Due to space limitation, we only draw 5 different initial function g (as shown from the top to bottom rows in Figure 4.2). From left to right columns, they are the initial g (first column), the true solution u^* (second column), the result obtained by the proposed method (third column), and the approximation obtained by NLS (fourth column), where the last three are at $t = 0.02$.

The training time for NLS takes approximately 1.3 hours using NLS, but only about 15 minutes using Algorithm 3. The testing time for individual initial condition, i.e., running time of Alg. 2, takes an average of 0.84 second with a standard deviation of 0.33 second. We remark that this time can be significantly reduced when multiple initials are queried in batch. For example, it takes only 1.94 seconds to solve 100 different initials when they are batched. These results demonstrate the significant improvement of the proposed method in both accuracy and efficiency.

4.3.3 Hyperbolic PDE

Our next test is on the following 10D nonlinear hyperbolic PDE:

$$\partial_t u(x,t) = F[u](x,t) := 2\nabla \cdot \tanh(u(x,t)) \tag{4.33}$$

with initial value $u(x, 0) = g(x)$ and $\Omega = (-1, 1)^{10}$. We remark that this is a very challenging problem and its solutions can often develop shocks which makes most methods suffer. Therefore, we choose $T = 0.15$ as this is the earliest we observed shock waves in the examples we shall compare against. Then we enforce periodic boundary conditions by using the same network u_θ in (4.32) as above. Here, we sample $\theta_i(0)$ such that $a_i, \beta \sim N(0_d, I_d)$ and $b_i \sim N(0, 1)$ for $i = 1, \ldots, 80$, and $c = (c_1, c_2, \ldots, c_{80})^\top$ is sampled uniformly from the sphere of radius 1. For this method to be scalable, we have to use special initials $g = u_{\theta(0)}$

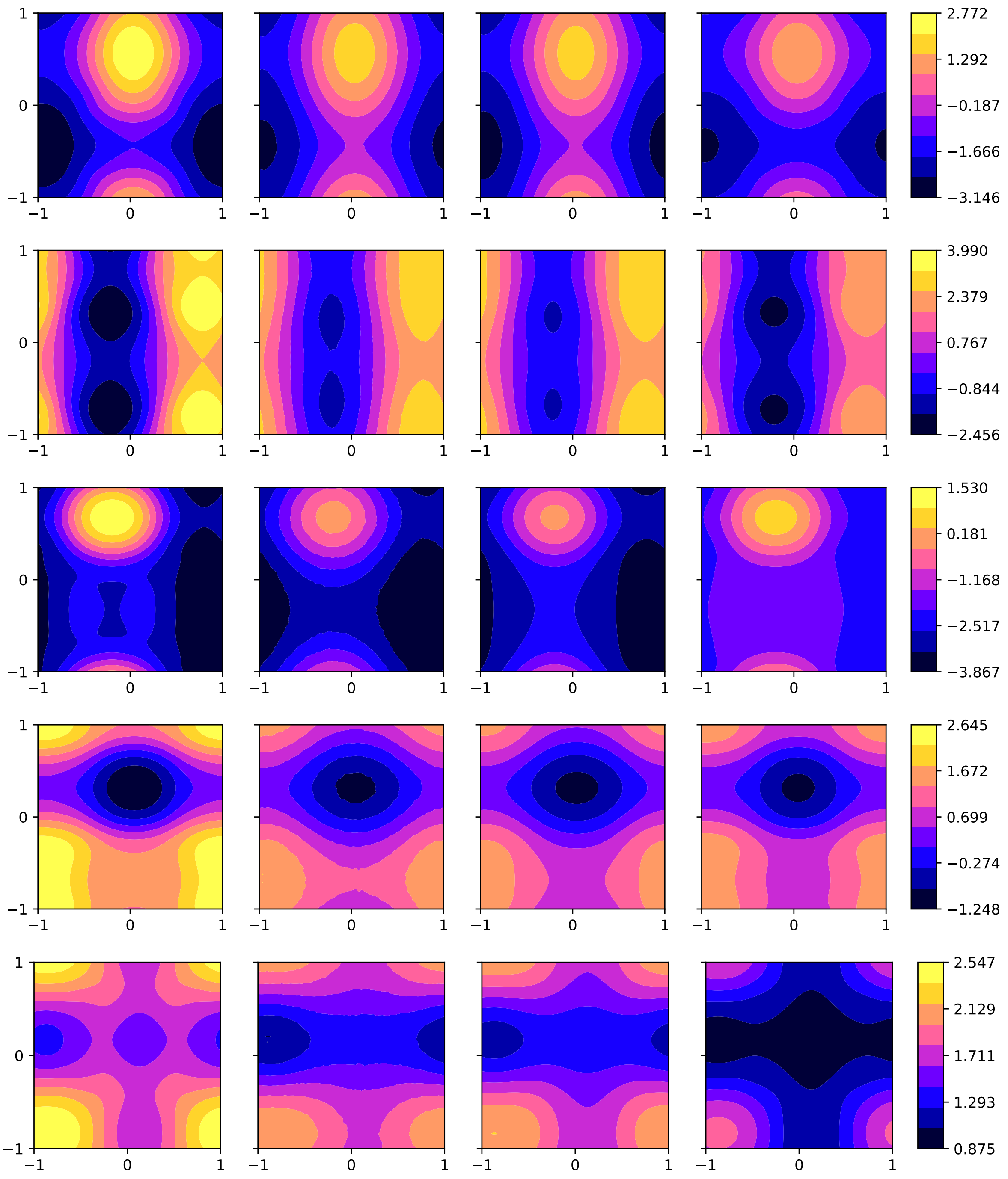


Figure 4.2 Comparison with NLS [34] on the Heat Equation (4.31). All plots show the (x_1, x_2) plane. (First column) initial g; (Second column): reference solution; (Third column) Proposed method; and (Fourth column) NLS [34]. In all cases, the proposed method demonstrates significant improvement on solution accuracy.

with $a_i(0) = (a_{i1}(0), 0, \dots, 0)^\top$ for $i = 1, \dots, 80$. For the θ's to be used in the terminal set $\mathcal{T}$, we make use of this form of our tested g's and sample β, b_i, and c as above but use $a_i(0) = (a_{i1}(0), 0, \dots, 0)^\top$ with $a_{i1} \sim N(0, 1)$.

The PDE (4.33) does not have analytic solutions. For comparison purpose, we use a dedicated solver [81], which is a numerical approximation using a first-order upwind scheme, with 4000 time steps and 1000 spatial steps to compute the reference solution. We use as initial conditions the g's defined above as their 1-d nature allows the computation of our reference solutions. We stress that the initials we use here for testing are not contained within the samples used for training. After training, we use the learned control V_ξ to generate solutions for all these initials. We plot mean and relative error of our method compared to the reference numerical solutions for 100 different random initials in Figure 4.3 (left). We observe a relative error of less than 4% on average, and more than 70% of results are below 8% relative error in our tests. These results suggest that our proposed method has great potential to be applied to more challenging PDEs.

4.3.4 Applications to Hamilton-Jacobi-Bellman equations

We consider the following stochastic control problem with control function α:

$$\min_{\alpha} \quad \frac{1}{2}\int_0^T |\alpha(X(t),t)|^2 dt + g(X(T)), \tag{4.34a}$$

$$\text{s.t.} \quad dX(t) = \alpha(X(t),t)dt + \sqrt{2\epsilon} dW, \qquad \text{where} \quad X(0) = x, \tag{4.34b}$$

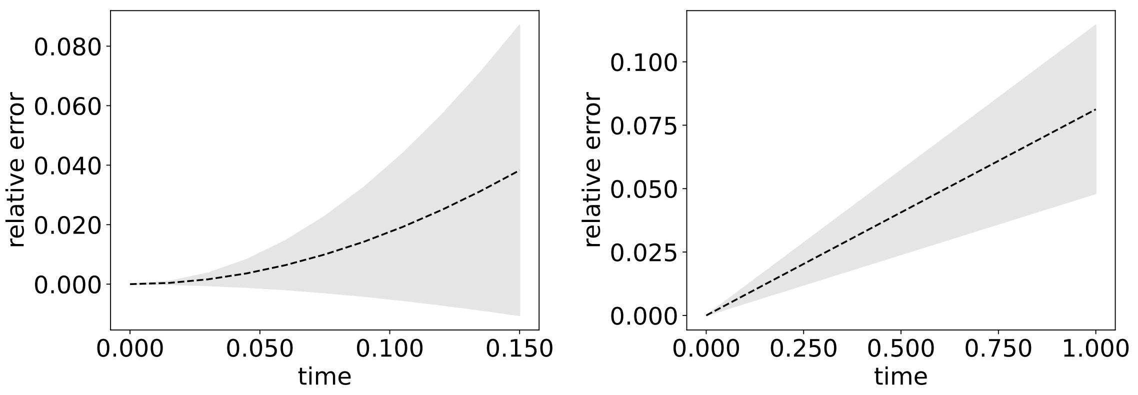


Figure 4.3 Mean relative error (dotted line) and standard deviation (grey) between the learned $u_{\theta(t)}$ and the reference solution u^* for 100 different random initial conditions. (Left) hyperbolic PDE (4.33); (Right) Hamiltonian-Jacobi-Bellman equation (4.35). Note that in either case, the tested initials are not included in the training datasets.

and the objective function consists of a running cost of α and a terminal cost g, and W is the standard Wiener process. We set problem dimension to $d = 8$, coefficient $\epsilon = 0.2$, and terminal time $T = 1$. Here $X(t) \in \mathbb{R}^d$ for all t. The corresponding Hamilton-Jacobi-Bellman (HJB) equation of this control problem is a second-order nonlinear PDE of the value function u:

$$\partial_t u(x,t) = -\epsilon \Delta u(x,t) + \frac{1}{2}|\nabla u(x,t)|^2, \quad \forall\, x \in \mathbb{R}^d,\ t \in [0,T] \tag{4.35}$$

with terminal cost (value) $u(x,T) = g(x)$. The optimal control $\alpha(x,t) = -\nabla u(x,t)$ for all x and t. This problem is a classic problem with many real-world applications [32]. Our goal is to find the solution operator that maps any terminal condition g to its corresponding solution. We note that (4.35) can be easily converted to an IVP with initial value g by changing t to $T - t$. We choose

$$u_\theta(x) = \sum_{i=1}^{50} w_i e^{-|a_i \odot (x-b_i)|^2/2},$$

where $\odot$ denotes component wise multiplication, and the parameters $\theta \in \mathbb{R}^m$ with $m = 850$ is the collection of $(w_i, a_i, b_i) \in \mathbb{R} \times \mathbb{R}^d \times \mathbb{R}^d$ for $i = 1, \ldots, 50$. We select this neural network as it is similar to those in the set $\mathcal{G}$ below (but they are not necessarily the actual solutions u^*) and satisfies $|u_\theta(x)| \to 0$ as $x \to \infty$ so long as $a_i \neq 0$. This is important as our solutions will be mainly concentrated in $[-3, 3]^d$ due to the solution property of the HJB equation and we want our neural network to reflect this. We sample g functions by setting them to $u_{\theta(0)}$ for $\theta(0)$ sampled from

$$\Theta := \{\theta : \ w_i \in (-1, 0), \ |a_i|_\infty \in (0.1, 2), \ |b_i|_\infty \in [-2, 2]\},$$

and use these to generate trajectories $\theta(t)$ for training. We sampled 250 points $\theta_i(0)'s$ from Θ uniformly and then used the time marching numerical method as described in [26] to generate a terminal dataset $\mathcal{T}$. As such we use the loss function (4.30) in our training. As $\Omega = \mathbb{R}^8$ we use importance sampling to generate the x points for our Monte-Carlo approximations of $\|\cdot\|_2$. Namely, we sample x from the distribution defined by the density $\rho(x; \theta) := \frac{1}{50} \sum_{i=1}^{50} N(x - b_i, \mathrm{diag}(a_i)^{-2})$.

After we train the V_ξ, we uniformly sample 100 terminal conditions g from the following set:

$$\mathcal{G} := \Big\{ \sum_{i=1}^{50} c_i e^{-|x-b_i|^2/\sigma_i^2} \ : \ c_i \in (-1, 0), \ \sigma_i^2 \in (0.5, 20), \ |b_i|_\infty \in [-2, 2] \Big\}.$$

We choose such $\mathcal{G}$ as it is an analogue to the Reproducing Kernel Hilbert Space (RKHS) and can represent a variety of functions by different combinations of functions in practice. For each g, we compute the solution using the Cole-Hopf transformation [30]:

$$u^*(x,t) = -2\epsilon \ln\left((4\epsilon\pi(T-t))^{-d/2} \int_{\mathbb{R}^d} \exp\left(-\frac{|x-y|^2}{4\epsilon(T-t)} - \frac{g(y)}{2\epsilon(T-t)} \right) dy \right).$$

We can approximate these true solutions using Monte-Carlo methods, which is done by using importance sampling to sample 20,000 points in $\mathbb{R}^8$ using the distribution $N(x, 2\epsilon(T-t))$. This allows us to approximate the integral above and compute u^* at each (x,t). We graph the average and standard deviation of the relative error of the solution by our approach for these 100 terminal costs in Figure 4.3 (right) with reversed time $t \leftarrow T-t$. This plot shows that the average relative error is about 7.5% and the standard deviation is less than 3%.

For demonstration, we show the performance of control $\hat{\alpha}(\cdot,t) := -\nabla u_{\theta(t)}(\cdot)$ obtained above. Due to space limitation, we select 5 terminal cost g's such that they vary in x_1, x_2 making the minimum locations clear. For each g, we sample 50 X_0's uniformly from $[-1.5, 1.5]^2 \times [-1,1]^6$ and generate their trajectories by solving the stochastic differential equation with $\hat{\alpha}$ in (4.34b) using the Euler-Maruyama method with step size 0.001. The initial $X(0)$'s (red dots) and the terminal locations $X(T)$'s (green triangle) averaged over 1000 runs for these terminals are shown from top to bottom rows in Figure 4.4. The gray-level contours show the corresponding terminal cost g and the locations of X are shown in the (x_1,x_2), (x_3,x_4), (x_5,x_6), and (x_7,x_8) planes in the four columns of Figure 4.4. From these plots, one can see properly controlled particle locations by using the solution obtained by the proposed method.

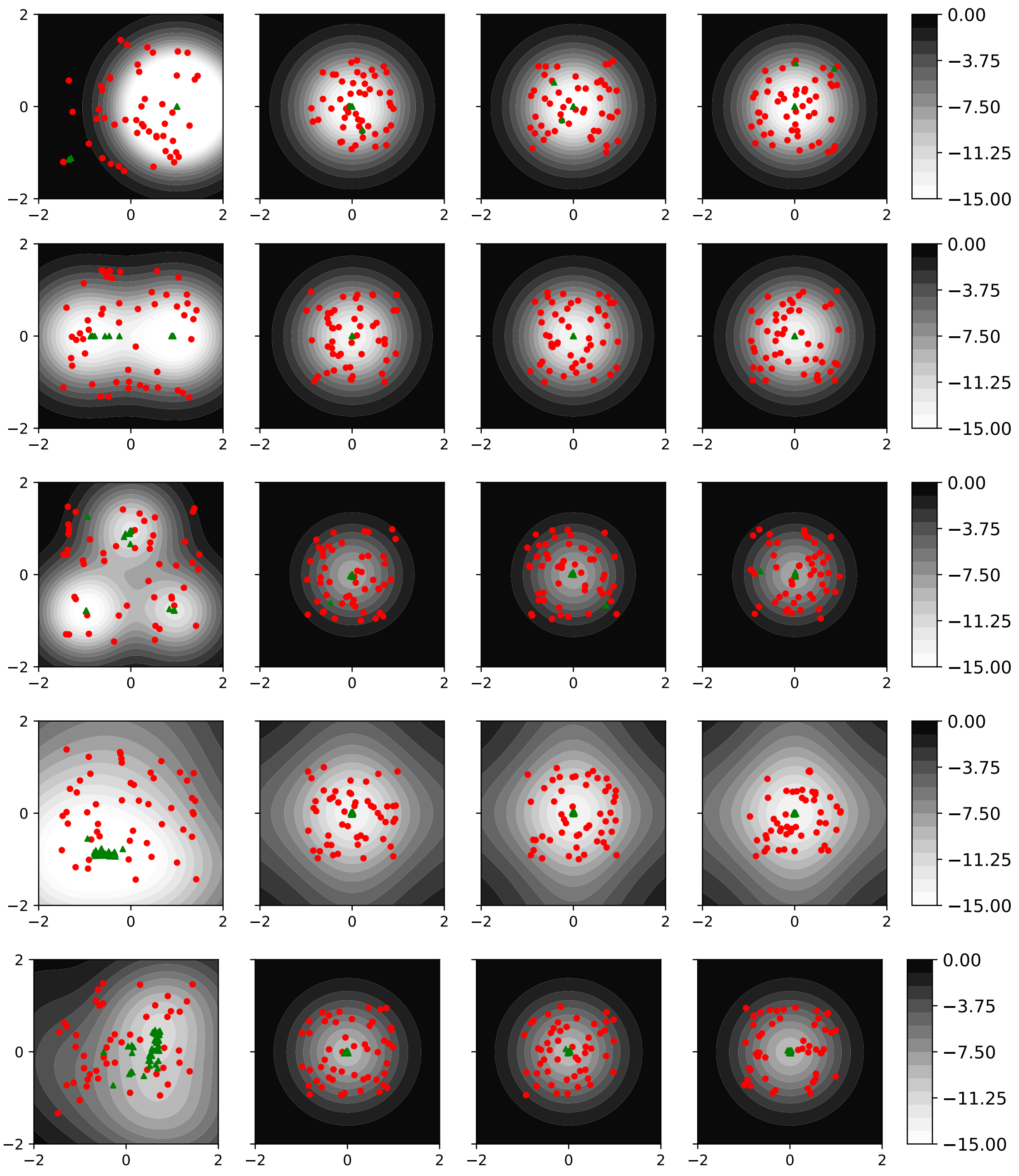


Figure 4.4 The evolution of 50 sampled points $X(0)$ (red circles) to time $X(1)$ (green triangles) in the (first column) (x_1, x_2) plane; (second column) (x_3, x_4) plane; (third column) (x_5, x_6) plane; and (fourth column) (x_7, x_8) plane. The background images show the expected minimum points of the terminal cost for the five randomly chosen initials. We see the solution $-\nabla u_{\theta(t)}$ provides correct control for all cases. Note that the induced control may not be able to steer the far-away initials to the minimum since the running cost penalizes large movements that the terminal gains do not compensate for. This phenomenon becomes less (more) likely as the terminal cost g is scaled larger (smaller).

APPENDICES

A Proof of Inequality in Proposition 1

In the proof of Proposition 1, we needed (3.16). This equation can be obtained by applying the lemma below, whose proof is a slight modification of the proof of [85, Theorem 2.1.14].

Lemma 8. *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a differentiable convex function and ∇f is L-Lipschitz continuous for some $L > 0$. Define the gradient descent iterates by*

$$x_i = x_{i-1} - h\nabla f(x_{i-1})$$

with $x_0 \in \mathbb{R}^d$. Let $y \in \mathbb{R}^d$ and $0 < h < \frac{1}{L}$, then for any $k \geq 1$ there is

$$f(x_k) - f(y) \leq \frac{|x_0 - y|^2}{2kh}.$$

Proof. Following the standard steps in the proof of [85, Theorem 2.1.14] and using $0 < h < 1/L$ we can derive the bound

$$f(x_i) - f(x_{i-1}) \leq -h\left(1 - \frac{1}{2}hL\right)|\nabla f(x_{i-1})|^2 \leq -\frac{h}{2}|\nabla f(x_{i-1})|^2. \tag{36}$$

Since f is convex, there is

$$f(x) \leq f(y) + \nabla f(x)^\top (x - y), \quad \forall x \in \mathbb{R}^d.$$

Combining this with $x = x_{i-1}$ and (36), we derive

$$\begin{aligned}
f(x_i) - f(y) &\leq \nabla f(x_{i-1})^\top (x_{i-1} - y) - \frac{h}{2}|\nabla f(x_{i-1})|^2 \\
&= \frac{1}{2h}\left(2h\nabla f(x_{i-1})^\top (x_{i-1} - y) - h^2|\nabla f(x_{i-1})|^2 + |x_{i-1} - y|^2 - |x_{i-1} - y|^2\right) \\
&= \frac{1}{2h}\left(|x_{i-1} - y|^2 - |x_{i-1} - h\nabla f(x_{i-1}) - y|^2\right) \\
&= \frac{1}{2h}\left(|x_{i-1} - y|^2 - |x_i - y|^2\right).
\end{aligned}$$

We can now bound the telescoping sum

$$\sum_{i=1}^{k}(f(x_i) - f(y)) \leq \frac{1}{2h}\sum_{i=1}^{k}(|x_{i-1} - y|^2 - |x_i - y|^2) \leq \frac{1}{2h}|x_0 - y|^2.$$

By (36) we know $f(x_k) \leq f(x_{k-1}) \leq \cdots \leq f(x_0)$ and therefore

$$f(x_k) - f(y) \leq \frac{1}{k}\sum_{i=1}^{k}(f(x_i) - f(y)) \leq \frac{|x_0 - y|^2}{2hk}.$$

□

B Proof of Proposition 2

First we sight the following corollary to Theorem 4 which follows Lemma 5.

Lemma 9. *Suppose Assumption 5 is satisfied. For all $\varepsilon > 0$ there exists $v : \bar{\Theta}_{u,F,L} \to \mathbb{R}^m$ such that v is bounded over $\bar{\Theta}_{u,F,L}$ and the value of v at θ, denoted by v_θ, satisfies*

$$\|v_\theta \cdot \nabla_\theta u_\theta - F[u_\theta]\|_{H^2} \leq \varepsilon, \qquad \forall\, \theta \in \bar{\Theta}.$$

Lemma 9 is an immediate result by combining Theorem 4 above and Lemma 5. Hence we omit the proof here. With this lemma in hand, we can prove our main result of this section.

Proof of Proposition 2. In what follows, we will follow the proof of Proposition 3.4 in [34] to extend that result from the L^2 to the H^1 norm. We first show that there exists a differentiable vector-valued function $V : \bar{\Theta}_{u,F,L} \to \mathbb{R}^d$ such that

$$\|V(\theta) \cdot \nabla_\theta u_\theta - F[u_\theta]\|_{H^2} \leq \frac{\varepsilon}{2} \tag{37}$$

for all $\theta \in \bar{\Theta}_{u,F,L}$. After which the claim about neural networks follows immediately from the proof of Proposition 1. To this end, we choose $\bar{\varepsilon}_0 \in (0, \varepsilon/2)$ and $\bar{\varepsilon} \in (\bar{\varepsilon}_0, \varepsilon/2)$, then by Theroem 4 and Lemma 9 we know that there exists a neural network u_θ, a bounded open set $\bar{\Theta}_{u,F,L} \subset \mathbb{R}^m$, and $M_v > 0$ such that there is a vector-valued function $\theta \mapsto v_\theta$, where for any $\theta \in \bar{\Theta}_{u,F,L}$, we have $|v_\theta| < M_v$ and

$$\|v_\theta \cdot \nabla_\theta u_\theta - F[u_\theta]\|_{H^2} \leq \bar{\varepsilon}.$$

Note that v_θ is not necessarily differentiable with respect to θ. To obtain a differentiable vector field $V(\theta)$, for each $\theta \in \bar{\Theta}_{u,F,L}$, we define the function ψ_θ by

$$\psi_\theta(w) := \|w \cdot \nabla_\theta u_\theta - F[u_\theta]\|_{H^2}^2 = w^\top G(\theta) w - 2w^\top p(\theta) + q(\theta),$$

where

$$\begin{aligned}
G(\theta) &= \sum_{i=1}^{d} \int_\Omega \nabla_\theta \partial_{x_i} u_\theta(x) \nabla_\theta \partial_{x_i} u_\theta(x)^\top dx + \int_\Omega \nabla_\theta u_\theta(x) \nabla_\theta u_\theta(x)^\top dx \\
p(\theta) &= \sum_{i=1}^{d} \int_\Omega \nabla_\theta \partial_{x_i} u_\theta(x) F[u_\theta](x) dx + \int_\Omega \nabla_\theta u_\theta(x) \nabla_\theta F[u_\theta](x) dx \\
q(\theta) &= \int_\Omega F[u_\theta](x) dx,
\end{aligned} \tag{38}$$

we are using the convention that $\nabla_\theta u_\theta \in \mathbb{R}^d$ is a column vector. Then we know

$$\psi_\theta^* := \psi_\theta(v_\theta) = \|v_\theta \cdot \nabla_\theta u_\theta - F[u_\theta]\|_{H^2}^2 \leq \bar{\varepsilon}^2. \tag{39}$$

It is also clear that $G(\theta)$ is symmetric and positive semi-definite. Moreover, due to the compactness of $\bar{\Omega}$ and $\bar{\Theta}_{u,F,L}$, as well as that $\nabla_\theta u \in C(\bar{\Omega} \times \bar{\Theta})$, we know there exists $\lambda_G > 0$ such that

$$\|G(\theta)\|_2 \leq \lambda_G$$

for all $\theta \in \bar{\Theta}_{u,F,L}$ with respect to the spectral norm. Therefore, ψ_θ is a convex function and the Lipschitz constant of $\nabla\psi_\theta$ is uniformly upper bounded by λ_G over $\bar{\Theta}_{u,F,L}$. Now for any $w \in \mathbb{R}^m$, $h > 0$, and $K \in \mathbb{N}$ (we reuse the letter K as the iteration counter instead of the number of sampling points in this proof), we define

$$\mathcal{O}_\theta^{K,h}(w) := w_K, \quad \text{where} \quad w_k = w_{k-1} - h\nabla\psi_\theta(w_{k-1}), \quad w_0 = w, \quad k = 1, \dots, K.$$

Namely, $\mathcal{O}_\theta^{K,h}$ is the oracle of executing the gradient descent optimization scheme on ψ_θ with step size $h > 0$ for K iterations.

Now we note that ψ_θ is convex, differentiable, and $\nabla\psi_\theta$ is Lipschitz continuous with Lipschitz constant upper bounded by λ_G. With these in mind, we can directly use the process from Proposition 1 to arrive at the inequality

$$\psi_\theta(\mathcal{O}_\theta^{K,h}(0)) - \psi_\theta^* \leq \left(\frac{\varepsilon}{2}\right)^2 - \bar{\varepsilon}^2. \tag{40}$$

Here we have set K to be some fixed number such that

$$K \geq \frac{M_v^2}{2h((\varepsilon/2)^2 - \bar{\varepsilon}^2)}.$$

Notice that $\mathcal{O}_\theta^{K,h}$ is a differentiable vector-valued function of θ because K and h are fixed. Therefore, combining (39) and (40) yields

$$0 \leq \psi_\theta(\mathcal{O}_\theta^{K,h}(0)) = (\psi_\theta(\mathcal{O}_\theta^{K,h}(0)) - \psi_\theta^*) + \psi_\theta^* \leq (\varepsilon/2)^2 - \bar{\varepsilon}^2 + \bar{\varepsilon}^2 = (\varepsilon/2)^2.$$

As this inequality holds $\forall\theta \in \bar{\Theta}$, we set $V(\theta) = \mathcal{O}_\theta^{K,h}(0)$ which is a differentiable function of θ satisfying (37). This completes the proof. $\square$

REFERENCES

[1] A. Abate, D. Ahmed, M. Giacobbe, and A. Peruffo. Formal synthesis of lyapunov neural networks. *IEEE Control Systems Letters*, 5(3):773–778, 2021.

[2] S. Amari. A theory of adaptive pattern classifiers. *IEEE Transactions on Electronic Computers*, EC-16(3):299–307, 1967.

[3] W. F. Ames. *Numerical methods for partial differential equations.* Academic press, 2014.

[4] W. Anderson and M. Farazmand. Evolution of nonlinear reduced-order solutions for pdes with conserved quantities. *SIAM Journal on Scientific Computing*, 44(1):A176–A197, 2022.

[5] C. Anitescu, E. Atroshchenko, N. Alajlan, and T. Rabczuk. Artificial neural network methods for the solution of second order boundary value problems. *Computers, Materials & Continua*, 59(1):345–359, 2019.

[6] K. Atkinson. *An Introduction to Numerical Analysis (2nd ed.).* John Wiley I& Sons, 1989.

[7] G. Bao, X. Ye, Y. Zang, and H. Zhou. Numerical solution of inverse problems by weak adversarial networks. *Inverse Problems*, 36(11):115003, 2020.

[8] C. Beck, W. E, and A. Jentzen. Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations. *Journal of Nonlinear Science*, pages 1–57, 2017.

[9] J. Berg and K. Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018.

[10] N. Boullé, C. Earls, and A. Townsend. Data-driven discovery of Green's functions with human-understandable deep learning. *Scientific Reports*, 12:4824, 03 2022.

[11] N. Boullé, S. Kim, T. Shi, and A. Townsend. Learning Green's functions associated with time-dependent partial differential equations. *Journal of Machine Learning Research*, 23:1–34, 08 2022.

[12] J. Bruna, B. Peherstorfer, and E. Vanden-Eijnden. Neural galerkin schemes with active learning for high-dimensional evolution equations. *Journal of Computational Physics*, 496:112588, 2024.

[13] S. Cai, Z. Wang, L. Lu, T. A. Zaki, and G. E. Karniadakis. Deepm&mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks. *arXiv preprint arXiv:2009.12935*, 2020.

[14] W. Cai, X. Li, and L. Liu. A phase shift deep neural network for high frequency approximation and wave problems. *SIAM Journal on Scientific Computing*, 42(5):A3285–A3312, 2020.

[15] W. Cai and Z.-Q. J. Xu. Multi-scale deep neural networks for solving high dimensional pdes. *arXiv preprint arXiv:1910.11710*, 2019.

[16] Y.-C. Chang, N. Roohi, and S. Gao. Neural lyapunov control. *Advances in neural information processing systems*, 32, 2019.

[17] L. Chen and C. Wu. A note on the expressive power of deep rectified linear unit

networks in high-dimensional spaces. *Mathematical Methods in The Applied Sciences*, 42:3400–3404, 2019.

[18] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[19] Y. Chen, B. Dong, and J. Xu. Meta-mgnet: Meta multigrid networks for solving parameterized partial differential equations. *arXiv preprint arXiv:2010.14088*, 2020.

[20] P. Clark Di Leoni, C. Meneveau, G. Karniadakis, and T. Zaki. Deep operator neural networks (DeepONets) for prediction of instability waves in high-speed boundary layers. *Bulletin of the American Physical Society*, 2020.

[21] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli. Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, 92(3):88, 2022.

[22] C. Dawson, Z. Qin, S. Gao, and C. Fan. Safe nonlinear control using robust neural lyapunov-barrier functions. *arXiv preprint arXiv:2109.06697*, 2021.

[23] T. De Ryck, S. Lanthaler, and S. Mishra. On the approximation of functions by tanh neural networks. *Neural Networks*, 143:732–750, 2021.

[24] M. Dissanayake and N. Phan-Thien. Neural-network-based approximations for solving partial differential equations. *Communications in Numerical Methods in Engineering*, 10(3):195–201, 1994.

[25] S. Dong and N. Ni. A method for representing periodic functions and enforcing exactly periodic boundary conditions with deep neural networks. *arXiv preprint*

arXiv:2007.07442, 2020.

[26] Y. Du and T. A. Zaki. Evolutional deep neural network. *Phys. Rev. E*, 104:045303, Oct 2021.

[27] W. E, J. Han, and A. Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *arXiv preprint arXiv:1706.04702*, 5(4):349–380, 2017.

[28] W. E and B. Yu. The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

[29] G. Evans, J. Blackledge, and P. Yardley. *Numerical methods for partial differential equations*. Springer Science & Business Media, 2012.

[30] L. C. Evans. *Partial differential equations*, volume 19 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 1998.

[31] Y. Fan, C. O. Bohorquez, and L. Ying. Bcr-net: A neural network based on the nonstandard wavelet form. *Journal of Computational Physics*, 384:1–15, 2019.

[32] W. Fleming and R. Rishel. *Deterministic and Stochastic Optimal Control*. Springer, 1975.

[33] M. Fujii, A. Takahashi, and M. Takahashi. Asymptotic expansion as prior knowledge in deep learning method for high dimensional bsdes. *Asia-Pacific Financial Markets*, pages 1–18, 2017.

[34] N. Gaby, X. Ye, and H. Zhou. Neural control of parametric solutions for high-

dimensional evolution pdes. *SIAM Journal on Scientific Computing (accepted)*, 2023.

[35] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[36] L. Grüne. Computing lyapunov functions using deep neural networks. *arXiv preprint arXiv:2005.08965*, 2020.

[37] Y. Gu, C. Wang, and H. Yang. Structure probing neural network deflation. *arXiv preprint arXiv:2007.03609*, 2020.

[38] Y. Gu, H. Yang, and C. Zhou. Selectnet: Self-paced learning for high-dimensional partial differential equations. *arXiv preprint arXiv:2001.04860*, 2020.

[39] I. Guhring, G. Kutyniok, and P. Peterson. Error bounds for approximations with deep relu neural networks in $w^{s,p}$ norms. *Analysis and Applications*, 18:803–859, 2020.

[40] I. Gühring and M. Raslan. Approximation rates for neural networks with encodable weights in smoothness spaces. *Neural Networks*, 134:107–130, 11 2020.

[41] M. Guo and J. S. Hesthaven. Data-driven reduced order modeling for time-dependent problems. *Computer Methods in Applied Mechanics and Engineering*, 345:75–99, 2019.

[42] X. Guo, W. Li, and F. Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 481–490, New York, NY, USA, 2016. Association for Computing Machinery.

[43] J. Han, A. Jentzen, and W. E. Overcoming the curse of dimensionality: Solving

high-dimensional partial differential equations using deep learning. *arXiv preprint arXiv:1707.02568*, pages 1–13, 2017.

[44] J. Han, A. Jentzen, and W. E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

[45] J. Han, J. Lu, and M. Zhou. Solving high-dimensional eigenvalue problems using deep neural networks: A diffusion monte carlo like approach. *Journal of Computational Physics*, 423:109792, 2020.

[46] Y. Han, J. Yoo, H. H. Kim, H. J. Shin, K. Sung, and J. C. Ye. Deep learning with domain adaptation for accelerated projection-reconstruction mr. *Magnetic resonance in medicine*, 80(3):1189–1205, 2018.

[47] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[48] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[49] J. Huang, H. Wang, and H. Yang. Int-deep: A deep learning initialized iterative method for nonlinear problems. *Journal of Computational Physics*, 419:109675, 2020.

[50] C. Huré, H. Pham, and X. Warin. Deep backward schemes for high-dimensional nonlinear pdes. *Mathematics of Computation*, 89(324):1547–1579, 2020.

[51] T. Huster, C.-Y. J. Chiang, and R. Chadha. Limitations of the lipschitz constant as a defense against adversarial examples. In *ECML PKDD 2018 Workshops*, pages 16–29.

Springer International Publishing, 2019.

[52] M. Hutzenthaler, A. Jentzen, T. Kruse, and T. A. Nguyen. A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations. *SN partial differential equations and applications*, 1(2):1–34, 2020.

[53] G. K. Ingo Guhring and P. Peterson. A compendium of comparison function results. *Mathematics of Control, Signals, and Systems*, 26:339–374, 2014.

[54] A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020.

[55] C. Johnson. *Numerical solution of partial differential equations by the finite element method*. Courier Corporation, 2012.

[56] H. Khalil. *Nonlinear Systems, 3rd Ed.* Prentice Hall, New Jersey, 2001.

[57] S. M. Khansari-Zadeh and A. Billard. Learning control lyapunov function to ensure stability of dynamical system-based robot reaching motions. *Robotics and Autonomous Systems*, 62(6):752–765, 2014.

[58] E. Kharazmi, Z. Zhang, and G. E. Karniadakis. hp-vpinns: Variational physics-informed neural networks with domain decomposition. *arXiv preprint arXiv:2003.05385*, 2020.

[59] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations,*

ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015.

[60] N. Kovachki, S. Lanthaler, and S. Mishra. On universal approximation and error bounds for Fourier neural operators. *Journal of Machine Learning Research*, 22:Art–No, 2021.

[61] N. B. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. M. Stuart, and A. Anandkumar. Neural operator: Learning maps between function spaces with applications to pdes. *J. Mach. Learn. Res.*, 24(89):1–97, 2023.

[62] M. Kumar and N. Yadav. Multilayer perceptrons and radial basis function neural network methods for the solution of differential equations: a survey. *Computers & Mathematics with Applications*, 62(10):3796–3811, 2011.

[63] I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

[64] H. Lee and I. S. Kang. Neural algorithm for solving differential equations. *Journal of Computational Physics*, 91(1):110–131, 1990.

[65] F. Lewis, S. Jagannathan, and A. Yesildirak. *Neural network control of robot manipulators and non-linear systems.* CRC press, 2020.

[66] B. Li, S. Tang, and H. Yu. Better approximations of high dimensional smooth functions by deep neural networks with rectified power units. *Communications in Computational Physics*, 27:379–411, 02 2020.

[67] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[68] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020.

[69] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*, 33(12):6999–7019, 2021.

[70] S. Liang and R. Srikant. Why deep neural networks for function approximation? In *International Conference on Learning Representations (ICLR)*, 2017.

[71] G. Lin, F. Chen, P. Hu, X. Chen, J. Chen, J. Wang, and Z. Shi. Bi-greennet: Learning Green's functions by boundary integral network. *arXiv preprint arXiv:2204.13247*, 2022.

[72] J. Liu, Y. Meng, M. Fitzsimmons, and R. Zhou. Physics-informed neural network lyapunov functions: Pde characterization, learning, and verification. *arXiv preprint arXiv:2312.09131*, 2023.

[73] Z. Liu, W. Cai, and Z.-Q. J. Xu. Multi-scale deep neural network (mscalednn) for solving poisson-boltzmann equation in complex domains. *arXiv preprint arXiv:2007.11207*, 2020.

[74] Y. Long and M. Bayoumi. Feedback stabilization: Control lyapunov functions modelled

by neural networks. In *Proceedings of 32nd IEEE Conference on Decision and Control*, pages 2812–2814. IEEE, 1993.

[75] W. Lötzsch, S. Ohler, and J. S. Otterbach. Learning the solution operator of boundary value problems using graph neural networks. *arXiv preprint arXiv:2206.14092*, 2022.

[76] L. Lu, P. Jin, and G. E. Karniadakis. DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

[77] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. Deepxde: A deep learning library for solving differential equations. *arXiv preprint arXiv:1907.04502*, 2019.

[78] T. Luo and H. Yang. Two-layer neural networks for partial differential equations: Optimization and generalization theory. *arXiv preprint arXiv:2006.15733*, 2020.

[79] L. Lyu, K. Wu, R. Du, and J. Chen. Enforcing exact boundary and initial conditions in the deep mixed residual method. *arXiv preprint arXiv:2008.01491*, 2020.

[80] M. Magill, F. Qureshi, and H. de Haan. Neural networks trained to solve differential equations learn general representations. In *Advances in Neural Information Processing Systems*, pages 4071–4081, 2018.

[81] Z. Mao, A. D. Jagtap, and G. E. Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.

[82] Z. Mao, L. Lu, O. Marxen, T. A. Zaki, and G. E. Karniadakis. DeepM&Mnet for hypersonics: Predicting the coupled flow and finite-rate chemistry behind a normal shock

using neural-network approximation of operators. *arXiv preprint arXiv:2011.03349*, 2020.

[83] S. Mukhopadhyay and F. Zhang. An algorithm for computing robust forward invariant sets of two dimensional nonlinear systems. *Asian Journal of Control*, 2020.

[84] M. A. Nabian and H. Meidani. A deep neural network surrogate for high-dimensional random partial differential equations. *arXiv preprint arXiv:1806.02957*, 2018.

[85] Y. Nesterov. Introductory lectures on convex programming. *Lecture Notes*, pages 119–120, 1998.

[86] N. Nüsken and L. Richter. Solving high-dimensional Hamilton–Jacobi–Bellman pdes using neural networks: perspectives from the theory of controlled diffusions and measures on path space. *Partial Differential Equations and Applications*, 2(4):1–48, 2021.

[87] G. Pang, M. D'Elia, M. Parks, and G. E. Karniadakis. nPINNs: nonlocal physics-informed neural networks for a parametrized nonlocal universal laplacian operator. algorithms and applications. *arXiv preprint arXiv:2004.04276*, 2020.

[88] G. Pang, L. Lu, and G. E. Karniadakis. fpinns: Fractional physics-informed neural networks. *SIAM Journal on Scientific Computing*, 41(4):A2603–A2626, 2019.

[89] P. Petersen and F. Voigtlaender. Optimal approximation of piecewise smooth functions using deep relu neural networks. *Neural Networks*, 108:296–330, 2018.

[90] H. Pham, X. Warin, and M. Germain. Neural networks-based backward scheme for fully nonlinear pdes. *SN Partial Differ. Equ. Appl.*, 2(1), 2021.

[91] A. Quarteroni and A. Valli. *Numerical approximation of partial differential equations*,

volume 23. Springer Science & Business Media, 2008.

[92] M. Raissi and G. E. Karniadakis. Machine learning of linear differential equations using gaussian processes. *arXiv preprint arXiv:1701.02440*, 2017.

[93] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.

[94] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[95] A. A. Ramabathiran and P. Ramachandran. Spinn: Sparse, physics-based, and partially interpretable neural networks for pdes. *Journal of Computational Physics*, 445:110600, 2021.

[96] B. Raonić, R. Molinaro, T. Rohner, S. Mishra, and E. de Bezenac. Convolutional neural operators. *arXiv preprint arXiv:2302.01178*, 2023.

[97] F. Regazzoni, L. Dedè, and A. Quarteroni. Machine learning for fast and reliable solution of time-dependent differential equations. *Journal of Computational Physics*, 397:108852, 2019.

[98] S. M. Richards, F. Berkenkamp, and A. Krause. The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems. In *Conference on Robot Learning*, pages 466–476. PMLR, 2018.

[99] A. Robey, H. Hu, L. Lindemann, H. Zhang, D. V. Dimarogonas, S. Tu, and N. Matni.

Learning control barrier functions from expert demonstrations. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 3717–3724, 2020.

[100] G. Serpen. Empirical approximation for lyapunov functions with artificial neural nets. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 735–740. IEEE, 2005.

[101] Z. Shen, H. Yang, and S. Zhang. Deep network approximation characterized by number of neurons. *Commun. Comput. Phys.*, 28(5):1768–1811, 2020.

[102] Z. Shen, H. Yang, and S. Zhang. Neural network approximation: Three hidden layers are enough. *Neural networks : the official journal of the International Neural Network Society*, 141:160–173, 2021.

[103] Y. Shin, J. Darbon, and G. E. Karniadakis. On the convergence and generalization of physics informed neural networks. *arXiv preprint arXiv:2004.01806*, 2020.

[104] J. Sirignano and K. Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018.

[105] Y. Song and S. Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

[106] E. D. Sontag. Feedback stabilization using two-hidden-layer nets. In *1991 American control conference*, pages 815–820. IEEE, 1991.

[107] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[108] P. Tabuada and B. Gharesifard. Universal approximation power of deep residual neural networks through the lens of control. *IEEE Transactions on Automatic Control*, pages 1–14, 2022.

[109] T. Tang and J. Yang. Implicit-explicit scheme for the allen-cahn equation preserves the maximum principle. *Journal of Computational Mathematics*, 34:471–481, 09 2016.

[110] U. Tanielian, M. Sangnier, and G. Biau. Approximating lipschitz continuous functions with groupsort neural networks. *arXiv preprint arXiv:2006.05254*, 2021.

[111] Y. Teng, X. Zhang, Z. Wang, and L. Ju. Learning Green's functions of linear reaction-diffusion equations with application to fast numerical solver. In *Proceedings of Mathematical and Scientific Machine Learning*, volume 190 of *Proceedings of Machine Learning Research*, pages 1–16. PMLR, 15–17 Aug 2022.

[112] J. W. Thomas. *Numerical partial differential equations: conservation laws and elliptic equations*, volume 33. Springer Science & Business Media, 2013.

[113] J. W. Thomas. *Numerical partial differential equations: finite difference methods*, volume 22. Springer Science & Business Media, 2013.

[114] H. Tsukamoto, S.-J. Chung, and J.-J. Slotine. Learning-based adaptive control using contraction theory. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 2533–2538, 2021.

[115] H. Tsukamoto, S.-J. Chung, and J.-J. E. Slotine. Contraction theory for nonlinear stability analysis and learning-based control: A tutorial overview. *Annual Reviews in Control*, 52:135–169, 2021.

[116] A. Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[117] B. Wang, W. Zhang, and W. Cai. Multi-scale deep neural network (mscalednn) methods for oscillatory stokes flows in complex domains. *arXiv preprint arXiv:2009.12729*, 2020.

[118] S. Wang, H. Wang, and P. Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Science advances*, 7(40):eabi8605, 2021.

[119] G. Wen, Z. Li, K. Azizzadenesheli, A. Anandkumar, and S. M. Benson. U-fno—an enhanced Fourier neural operator-based deep-learning model for multiphase flow. *Advances in Water Resources*, 163:104180, 2022.

[120] W. Xiao, R. Hasani, X. Li, and D. Rus. Barriernet: A safety-guaranteed layer for neural networks. *arXiv preprint arXiv:2111.11277*, 2021.

[121] L. Yang, D. Zhang, and G. E. Karniadakis. Physics-informed generative adversarial networks for stochastic differential equations. *SIAM Journal on Scientific Computing*, 42(1):A292–A317, 2020.

[122] Y. Yang and P. Perdikaris. Adversarial uncertainty quantification in physics-informed neural networks. *Journal of Computational Physics*, 394:136–152, 2019.

[123] D. Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94:103–114, 2017.

[124] Y. Zang, G. Bao, X. Ye, and H. Zhou. Weak adversarial networks for high-dimensional

partial differential equations. *Journal of Computational Physics*, page 109409, 2020.

[125] Y. Zhu and N. Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447, 2018.