Chapter 6

Theoretical foundations of physics-informed neural networks and deep neural operators

A brief review

Yeonjong Shin[a,*], Zhongqiang Zhang[b], and George Em Karniadakis[c]
[a]*Department of Mathematics, North Carolina State University, Raleigh, NC, United States,* [b]*Department of Mathematical Sciences, Worcester Polytechnic Institute, Worcester, MA, United States,* [c]*Division of Applied Mathematics, Brown University, Providence, RI, United States*
Corresponding author: e-mail address: yeonjong_shin@ncsu.edu

Contents

Abstract
This chapter presents a brief review of the theoretical foundations of physics-informed neural networks (PINNs) and deep neural operators. PINN is one of the most popular deep learning approaches for solving both forward and inverse problems of partial differential equations (PDEs). It provides seamless ways of embedding laws of physics into deep neural networks (DNNs) by leveraging auto-differentiation. At the same time, operator learning emerged as a new learning paradigm for learning nonlinear operators, particularly ones relevant to PDEs. Deep Operator Network (DeepONet) is one of the first pioneering models whose architecture is inspired by the universal approximation theorem. DeepONets can generate reliable real-time responses when they are pretrained with a large amount of data pairs of inputs and outputs. Topics to be covered include mathematical formulations, approximation error estimates, approximation theory of DNNs, and training/optimization methods.

Keywords
Physics-informed neural networks, Deep operator networks, Deep learning, Optimization, Approximation theory, Partial differential equations

MSC Codes
65-00, 90-00, 41-00

1 Introduction

In this chapter, we present a review of the theoretical development of physics-informed neural networks (PINNs, e.g. in Lagaris et al., 1998, and Raissi et al., 2019) and deep operator networks (DeepONets, e.g., in Lu et al., 2021a) for partial differential equations.

In physics-informed neural networks, partial differential equations including forward and inverse problems are reformulated as minimization of least-squares of the residuals of the equations and their constraints at random points (often uniformly distributed). Since the neural networks are nonconvex and nonlinear in their trainable parameters, we often use stochastic gradient descent methods to find approximate solutions to the minimization problems. Advances in PINNs include choices of loss functions, distribution of sampling points, architectures of neural networks, efficient training methods, and many techniques addressing issues in various applications.

Compared to least-squares finite element methods (Bochev and Gunzburger, 1998; Bramble and Schatz, 1970) and least-squares collocations, the approximators are networks and thus are highly nonlinear and nonconvex. Correspondingly, different formulations and solvers for the least-squares problems have been applied and some error estimates of PINNs have been established.

While PINNs may need no data or little data, DeepONets require a large amount of data but DeepONets learn maps/operators from a function input to a function output. In the vanilla DeepONets, a regression problem is formulated as the empirical L^2-norm of the differences among the output of neural networks at input data and corresponding output data. Developments of DeepONets include but are not limited to those in architectures of neural networks, efficient training methods and error estimates.

In Section 2, we briefly introduce a special class of neural networks-feedforward neural networks, although many other networks work as well in most of the settings in this work. In Section 3, we introduce an abstract problem and present some classical forward and inverse problems to solve with PINNs. We also introduce the strong formulations and weak formulations used in PINNs. Various aspects of PINNs are addressed with various techniques to solve many types of problems. In Section 4, we discuss the error estimates of PINNs for conditionally stable problems. The estimates are based on the conditional stability of problems. We present some developments of training methods in Section 5 for the loss functions, which may not be limited to loss functions in PINNs. In Section 6, we present a special neural network where the weights and biases are small. This network is motivated by expressing functions with discontinuity or large derivatives, such as in hyperbolic problems. In Section 7, we briefly discuss PINNs when observational data are available, e.g. in inverse problems. In Section 8, we present error estimates for DeepONets for continuous operators and three examples of solution operators arising from linear and nonlinear advection-diffusion-reaction equations.

2 Neural networks

Many neural network architectures can be used for scientific machine learning, such as radial basis function networks, convolution neural networks, residual neural networks, recurrent Neural networks, U-nets, etc. Let's focus on feed-forward neural networks. A L-layer feed-forward neural network (NN) u_{NN} is a function $\mathbb{R}^{d_{\mathrm{in}}} \mapsto \mathbb{R}^{d_{\mathrm{out}}}$ defined by

$$u_{\mathrm{NN}}(x;\theta) = W^L u^{L-1}(x) + b^L, \tag{2.1}$$

where u^{L-1} is recursively computed by

$$u^\ell(x) = \phi(W^\ell u^{\ell-1}(x) + b^\ell), \quad 1 \le \ell < L, \qquad u^0(x) = x.$$

Here ϕ is a nonlinear activation function that applies element-wise. Some popular activation functions include the hyperbolic tangent (tanh) and the rectified linear unit (ReLU). L is a positive integer greater than or equal to 2, referred to as the depth. For notational convenience, the input and output dimensions are often denoted by $n_0 = d_{\mathrm{in}}$ and $n_L = d_{\mathrm{out}}$, respectively and these are assumed to be given for every learning task. The weight matrix and bias vector of the ℓ-th hidden layer are $W^\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ and $b^\ell \in \mathbb{R}^{n_\ell}$, respectively. The collection θ of all the weight matrices and bias vectors determines u_{NN} and is called the network parameter. The vector $\vec{n}_L = (n_1, \ldots, n_{L-1}) \in \mathbb{N}^{L-1}$ is referred to as the architecture, which determines the size of θ. To explicitly express the dependency of the architecture, the network parameter is often denoted by $\theta(\vec{n}_L)$, whose size and magnitude are given by

$$|\vec{n}_L| = \sum_{\ell=1}^{L} n_\ell(n_{\ell-1}+1), \quad |\theta|_\infty = \max_{1\le\ell\le L} \max\{\|W^\ell\|_{\max}, \|b^\ell\|_{\max}\}.$$

A class of L-layer NNs is then given by

$$V_L(M) = \left\{u_{\mathrm{NN}}(\cdot;\theta(\vec{n}_L)) : |\theta(\vec{n}_L)|_\infty \le M, \vec{n}_L \in \mathbb{N}^{L-1}\right\}. \tag{2.2}$$

The class V_L does not form a linear space as it is not closed under addition, i.e., for any $u, v \in V_L$, $u+v \notin V_L$. However, it has been well known that V_L is universal. For example, for any continuous function f in a compact domain $\Omega \subset \mathbb{R}^{d_{\mathrm{in}}}$ and any $\epsilon > 0$, there exist M and $\vec{n}_L$ such that $\|f(x) - u_{\mathrm{NN}}(x,\theta(\vec{n}_L))\| < \epsilon$ for any $x \in \Omega$.

To provide a mathematical explanation of the empirical success of NNs, many works have focused on quantifying the approximation ability of NNs using the number of nonzero network parameters, the magnitudes, the depth, and the width, to name a few. The works of Petersen and Voigtlaender (2018) proved universal approximation theorems of rectified linear units (ReLU) networks, showing the advantage of the depth. On the other hand, the class of two-layer neural networks has been shown to effectively approximate certain functions, and such a class is now known as the Barron class.

3 Mathematical formulations

Consider the following problem

$$\begin{aligned} \mathcal{D}[u](x) &= f(x) \quad x \in \Omega \\ \mathcal{B}[u](x) &= g(x) \quad x \in \Gamma, \end{aligned} \tag{3.1}$$

where $\mathcal{D} : X \to Y$ $\mathcal{B} : X \to Z$ are appropriate differential operators. Here X, Y and Z are Banach spaces. Here $\Omega \subset \mathbb{R}^d$ is a computational (compact) domain and Γ is a subset of $\mathbb{R}^d$. This abstract problem can include many interesting problems. Below, we list some typical benchmark problems when developing deep learning methods for partial differential equations (PDEs).

Example 3.1 (Poisson equation with Dirichlet boundary condition, forward problem). Let $\mathcal{D} = -\sum_{j=1}^{d} \partial_{x_i}^2 := \Delta$ and $\mathcal{B} = \mathrm{Id}$ (identity operator). Let $\Omega = (0,1)^2$, $\Gamma = \partial\Omega = \{x = (x_1, x_2) \in \Omega | x{=}(0, x_2) \text{ or } (1, x_2), \text{ or } (x_1, 0), \text{ or } (x_1, 1)\}$. The problem is usually written as

$$-\Delta u(x) = f(x), \quad x \in \Omega; \qquad u(x) = g(x), \quad x \in \Gamma = \partial\Omega. \tag{3.2}$$

Here f and g are given, we seek the solution u on the Ω.

Example 3.2 (Data assimilation of Poisson's equation, inverse problem). Let $\Omega \subset \mathbb{R}^d$ be a Lipschitz domain. Let $\Omega_0 \subsetneq \Omega$ be open. Consider the following problem

$$-\Delta u = f, \quad x \in \Omega; \qquad u = g, \quad x \in \Omega_0.$$

Here f and g are given, we seek the solution u on Ω and $\partial\Omega$. This problem is often called a *continuation* problem.

Example 3.3 (Cauchy problem of Poisson's equation, inverse problem). Let $\Omega \subset \mathbb{R}^d$ be a Lipschitz domain. Let $\Gamma_D \subsetneq \partial\Omega$ be an open subset with a positive area. Consider the following Cauchy problem

$$-\Delta u = f, \quad x \in \Omega; \qquad u = g, \quad x \in \Gamma_D; \qquad \nabla u \cdot \mathbf{n} = \varphi, \quad x \in \Gamma_D.$$

In this problem, we only know partial data on the boundary and thus we seek the solution as well as the data on the whole boundary. This problem is often called a Cauchy problem for the Poisson equation.

Example 3.4 (Coefficient inverse problem). Let $D \subset \mathbb{R}^n$ be a smooth bounded domain. Consider the initial boundary value problem on $\Omega = D \times (0, T)$ (T>0),

$$\partial_t^2 u = \nabla \cdot (p(x)\nabla u), \quad (x,t) \in D \times (0,T);$$
$$u(x,0) = g_0(x),\ \partial_t u(x,0) = 0,\ x \in D;\ u = g(x,t),\ (x,t) \in S_T = \partial D \times (0,T).$$

Here the coefficient $p(x) \in C^1(\bar{D})$ and $p(x) \geq p_0 > 0$. Here the functions g_0 and g are known, while The goal is to find the coefficient $p(x)$ in D. We also know the normal derivative of u $\nabla u \cdot \mathbf{n} = h(x,t)$ on S_T, where $\mathbf{n}$ is the outward normal vector at the cylindrical surface S_T.

3.1 Stability

We assume that the problem (3.1) is conditionally stable. We use the following definition of conditional stability, see e.g. in Burman and Oksanen (2018) and Dahmen et al. (2023).

Definition 3.5 (Conditional stability). Let X and Y be Banach spaces. Let $\mathcal{D} : X \mapsto Y$ and $\mathcal{B} : X \mapsto Z$ be linear maps. Assume that there exists a linear operator $L : X \to H$ where H is a Banach space such that there is a positive function $\mathsf{N} : \hat{X} \to (0, \infty)$[1] and a nondecreasing function $\rho_C : [0, \infty] \to [0, \infty]$ with $\lim_{t\searrow 0} \rho_C(t) = 0$ for any $C > 0$, such that for $v = v_1 - v_2 \in X$ ($v_1, v_2 \in X$) with $\|Lv\|_H \leq C$, it holds that

$$\mathsf{N}(v_1 - v_2) \leq \rho_C(\|\mathcal{D}v_1 - \mathcal{D}v_2\|_Y + \|\mathcal{B}v_1 - \mathcal{B}v_2\|_Z). \tag{3.3}$$

Here N is usually a norm or a seminorm on a Banach space V which $\widehat{X}$ can be embedded into. We then call the problem (3.1) conditionally stable.

1 Here $\widehat{X}$ is a Banach space, which often coincides with $\widehat{X}$.

Here L is often called regularization. When N is a norm, the problem is well-posed in the Hadamard sense: the solution exists and is unique and stable with respect to inputs. In this case, the problem is called unconditionally stable.

A special case of the conditional stability is linear stability. In *linear stability*, N is a norm and ρ_C is a linear function. For example, in Example 3.1, we can derive the following stability[2]

$$\|u\|_{L^2(\Omega)} \le C(\|\Delta u\|_{H^{-3/2}(\Omega)} + \|u\|_{L^2(\partial\Omega)}) \le C(\|\Delta u\|_{L^2(\Omega)} + \|u\|_{L^2(\partial\Omega)}). \tag{3.4}$$

Here H^r is the Sobolev-Hilbert space with elements and their r-th weak derivatives being square integrable.

The conditional stability for the Problem 3.2 is stated as follows.

Theorem 3.6 (Conditional stability for the Poisson continuation problem, Burman and Oksanen, 2018). *Let $f \in L^2(\Omega)$ and let $u \in H^1(\Omega)$ such that the equation in the Problem 3.2 holds. Let $g \in H^1(\Omega_0)$. Then for every open simply connected set $E \subset \Omega$ such that $\operatorname{dist}(E, \partial D) > 0$, there holds*

$$\|u\|_{H^1(E)} \le C\left(\|u\|_{H^1(\Omega)}\right)^{1-r} \omega\left(\|f\|_{H^{-1}(\Omega)} + \|g\|_{L^2(\Omega_0)}\right)^r.$$

Here $r \in (0,1)$, depending on the set E. On the domain Ω, we have

$$\|u\|_{H^1(\Omega)} \le C\left(\|u\|_{H^1(\Omega)}\right)^r \left[\log((\|f\|_{H^{-1}(\Omega)} + \|g\|_{L^2(\Omega_0)})\right]^{-r}.$$

The conditional stability of the Problem 3.3 can be found in Burman and Oksanen (2018). The inverse problem in Example 3.4 is Lipschitz stable, i.e., ρ_C is stable, see e.g., Klibanov and Yamamoto (2006).

Many problems have proven to be conditionally stable in literature, such as classical partial differential equations, integral equations and a large class of inverse problems.

Remark 3.7. Here the conditional stability is described for linear problems. From the definition, it can be further extended to nonlinear problems, where ρ_C may depend on v_1 and v_2.

Solving the problem (3.1) can be formulated as an optimization problem using the residual minimization

$$\inf_{v \in \widehat{X}} \mathcal{J}(v), \tag{3.5}$$

where the loss functional $\mathcal{J}_\tau(v)$ can be established in many ways. Depending on how the optimization problem is reformulated, different methods are obtained, e.g. deep Ritz methods (E and Yu, 2018) using energy minimization and physics-informed neural networks (PINNs) in Lagaris et al. (1998) and Raissi et al. (2019), which is based on residual minimization while approximate solutions being neural networks.

[2] The L^2 norm on the left can be replaced by $\|u\|_{H^{1/2}(\Omega)}$.

3.2 Strong formulation

The most straightforward method is based on the strong form of the governing equation.

$$\mathcal{J}(v) = \|f - \mathcal{D}v\|_Y^2 + \|g - \mathcal{B}v\|_Z^2 + \epsilon^2 \|Lv\|_H^2 . \tag{3.6}$$

Here ϵ is usually a small parameter. This is one of the most popular approach used in practice. To implement this formulation, we enforce the approximation to satisfy the governing equation over a finite set of points. More precisely, let $\Omega_M \subset \Omega$ and $\Gamma_{M'} \subset \partial\Omega$ be such sets with $|\Omega_M| = M$ and $|\partial\Omega_{M'}| = M'$. Let $\omega_M(\cdot) = \frac{1}{M}\sum_{x\in\Omega_M}\delta_x(\cdot)$ and $\mu_{M'}(\cdot) = \frac{1}{M'}\sum_{x\in\Gamma_{M'}}\delta_x(\cdot)$ be empirical measures for Ω_M and $\Gamma_{M'}$, respectively, where δ_x is the Dirac measure. The naive PINN method (Lagaris et al., 1998; Raissi et al., 2019) defines an objective (also known as a physics-informed loss) functional as $\mathcal{L}(v) = \mathcal{L}_{\text{re}}(v) + \mathcal{L}_{\text{bc}}(v)$ where

$$\mathcal{L}_{\text{re}}(v) = \int_\Omega \mathrm{L}_{\text{re}}[v](x)d\omega_M(x), \quad \mathcal{L}_{\text{bc}}(v) = \int_\Gamma \mathrm{L}_{\text{bc}}[v](x)d\mu_{M'}(x), \tag{3.7}$$
$$\mathrm{L}_{\text{re}}[v](x) := [\mathcal{D}[v](x) - f(x)]^2 , \mathrm{L}_{\text{bc}}[v](x) := [\mathcal{B}[v](x) - g(x)]^2 .$$

and seeks an approximation in V_n that minimizes the loss functional $\mathcal{L}$, i.e., $u_n^* = \operatorname{argmin}_{v\in V_n}\mathcal{L}(v)$. Ideally, one wants to minimize the continuous loss functional defined by

$$\mathcal{L}_\infty(v) = \int_\Omega \mathrm{L}_{\text{re}}[v](x)d\omega(x) + \int_\Gamma \mathrm{L}_{\text{bc}}[v](x)d\mu(x), \tag{3.8}$$

where ω and μ are probability measures typically determined by the underlying PDEs and the corresponding solution space. However, since the exact calculations of the integrals are not available, one has to rely on reasonable discretization or approximation. The work of Sirignano and Spiliopoulos (2018) proposed an NN approach, namely, the deep Galerkin method, which aims at minimizing the ideal loss (3.8) by using the stochastic mini-batch gradient descent (see also Section 5.2).

The loss functional can be further generalized by introducing nonhomogeneous weights for the summands:

$$\mathcal{L}(v;\boldsymbol{\lambda}) = \sum_{x\in\Omega_M}\lambda_x\Big(\mathcal{D}[v](x) - f(x)\Big)^2 + \sum_{x\in\Gamma_{M'}}\lambda_x\Big(\mathcal{B}[v](x) - g(x)\Big)^2, \tag{3.9}$$

where $\boldsymbol{\lambda}$ is the collection of all the λ_x, which includes the application of Monte Carlo integration when $\lambda_x = \frac{1}{M}$ if $x \in \Omega_m$ and $\lambda_x = \frac{1}{M'}$ if $x \in \partial\Omega_{M'}$.

It is worth mentioning that a minimizer u_n^* depends on Ω_M, $\partial\Omega_{M'}$, $\boldsymbol{\lambda}$ and V_n.

3.3 Weak/variational formulations

A weak formulation of the general equation may be written as follows. Let V be a Hilbert space satisfying $\mathcal{B}[u] = g$ for all u. For general discussion, let us define

$$a(u, v) := \langle \mathcal{D}[u], v\rangle, \quad L(v) := \langle f, v\rangle,$$
$$\mathrm{L}_{\mathrm{re}}[v](x) := [\mathcal{D}[v](x) - f(x)]^2, \quad \mathrm{L}_{\mathrm{bc}}[v](x) := [\mathcal{B}[v](x) - g(x)]^2,$$

where $\langle\cdot,\cdot\rangle$ is an appropriate inner product. The variational formulation then seeks to find $u \in V$ such that

$$a(u, v) = L(v) \quad \forall v \in V.$$

The NN approaches seek to solve the variational problem with a subset of V, denoted by V_n of NNs, which is the set of approximate solutions.

Let us consider the Poisson equation to illustrate the variational approach. Let $\mathcal{D} = -\Delta$, $g = 0$ and $\Omega = (0, 1)$. Let $V = H_0^1(\Omega) = \{v \in H^1(\Omega) : v(0) = v(1) = 0\}$, $\langle u, v\rangle = \int_0^1 u(x)v(x)dx$ for all $u, v \in V$ and $f \in L^2(\Omega)$. It then follows from the integration by parts that

$$a(u, v) = \langle u', v'\rangle, \quad L(v) = \langle f, v\rangle.$$

If V_n is a linear space spanned by $\{\phi_j \in V : j = 1, .., n\}$, the above is equivalent to solving a linear system of equations of the form

$$Ac = b, \quad \text{where} \quad A_{ij} = a(\phi_i, \phi_j), \quad b_i = L(\phi_i). \tag{3.10}$$

The solution is then given by $u_n^* = \sum_{j=1}^n \hat{c}_j \phi_j$ where $\hat{c}$ is the solution to $Ac = b$, which basically describes the finite element method (FEM). If V_n is a class of neural networks, since V_n is not a linear space, the FEM-like approach cannot apply. Therefore, one needs new approaches to solve the variational problem using NNs.

Ritz-Galerkin approach. The work of Ainsworth and Dong (2021) proposed an NN approach based on a standard Galerkin approximation of the variation equation, namely Galerkin Neural Networks (GNNs). Let $a : V \times V \mapsto \mathbb{R}$ be a bounded symmetric, bilinear form satisfying $a(v, v) \geq 0$ and $a(v, v) = 0$ if and only if $v = 0$, which defines an associated norm given by $|||v||| := \sqrt{a(v, v)}$.

For a given basis $\{\phi_i\}_{i=1}^n$, let us define $u_n^* = \mathrm{GALERKIN}(a, L, \{\phi_i\}_{i=1}^n)$ where u_n^* is the solution obtained from (3.10). The GNN method seeks basis functions from a class of NNs and provides the corresponding Galerkin NN approximation. The GNN constructs one basis at a time and gradually augments the basis as the algorithm proceeds. The GNN mechanism is built based on the following result.

Proposition 3.8 (Proposition 2.3 of Ainsworth and Dong, 2021). *Let u be the solution to the variational formulation and $u_0 \in V$ be a given approximation. Let $\varphi_1 = \frac{u-u_0}{|||u-u_0|||}$ and let $r(u_0) : V \to \mathbb{R}$ be a functional given by the rule $r(u_0)[v] := L(v) - a(u_0, v)$. Then, $r(u_0)[\varphi_1] = \max_{v \in B} r(u_0)[v]$ where B is the closed unit ball in V.*

The proposition established the relationship between the residual $r(u_0)$ and the approximation error $u - u_0$. Based on this error indicator, the GNN commences with an initial guess u_0 and recursively augments the basis according to the following. For $k = 1, \dots, n$,

$$v_k := \underset{v \in V_n, |||v||| \neq 0}{\operatorname{argmax}} \; r(u_{k-1})[\bar{v}] = \underset{v \in V_n, |||v||| \neq 0}{\operatorname{argmax}} \; \frac{L(v) - a(u_{k-1}, v)}{|||v|||},$$

and $u_k = \text{GALERKIN}(a, L, \{\varphi_i^{\text{NN}}\}_{i=0}^{k})$ where $\varphi_i^{\text{NN}} = \frac{v_i}{|||v_i|||}$ with $v_0 = u_0$. The GNN approximation to the variational formulation is then given by u_n where n is a user-defined input representing the number of NN basis functions. The original work allows one to use different V_n for each v_k to further facilitate the training and improve the performance. The convergence of the GNN with respect to the number of basis functions is proved in Ainsworth and Dong (2021).

Minmax formulation. The work of Zang et al. (2020) followed the min-max formulation

$$\min_{u \in V_n} \left(\max_{v \in V_n} \frac{|a(u, v) - L(v)|^2}{\|v\|^2} \right),$$

and the approach was termed the weak adversarial network (WAN). In practice, the integration is approximated by a quadrature rule. Also, similar to the deep Ritz method, the boundary loss term is added. Let $\{x_k, w_k\}_{k=1}^{M}$ be a quadrature rule over $[0, 1]$. Then the loss functional is given by

$$\mathcal{L}(u, v) = \frac{\left| \sum_{k=1}^{M} w_k \cdot \left(u'(x_k) v'(x_k) - f(x_k) v(x_k) \right) \right|^2}{\left| \sum_{k=1}^{M} w_k \cdot v^2(x_k) \right|^2}.$$

To further improve the training, the WAN approach takes the log of the loss and aims to solve the minmax problem

$$\min_{u \in V_n} \left(\max_{v \in V_n} \log \mathcal{L}(u, v) + \mathcal{L}_{\text{bc}}(u) \right).$$

Petrov-Galerkin approach. Several works investigated NN approaches that use the Petrov-Galerkin (PG) formulation. Let W be a Hilbert space that differs from V. The PG seeks to find $u \in V$ such that $a(u, v) = L(v)$ for all $v \in W$. The corresponding NN approach uses a fixed finite-dimensional linear space of

W_n and seeks a solution in a class V_n of NNs. The works of Khodayi-Mehr and Zavlanos (2020); Shang et al. (2022) used linear finite element spaces and the work of Kharazmi et al. (2019) used sine functions, Legendre polynomials as test spaces. In Kharazmi et al. (2021), the *hp*-variational physics-informed neural networks were proposed as a general framework that combines the PG approach and *hp*-refinement via domain decomposition, where piecewise polynomials were used for test functions.

3.4 Extended PINN: domain decomposition

Extended PINNs (XPINNs) (Jagtap and Karniadakis, 2020) combine a domain decomposition method with PINNs. Let $\{\Omega_i\}_{i=1}^N$ be a partition of the domain Ω and consider a set of subproblems:

$$\begin{aligned} \mathcal{D}[u_i](x) &= f(x) \quad && x \in \Omega_i \\ \mathcal{B}[u_i](x) &= g(x) \quad && x \in \partial\Omega_i \cap \Gamma \\ u_i(x) &= u_j(x) \quad && x \in \partial\Omega_i \cap \partial\Omega_j. \end{aligned} \tag{3.11}$$

Then, the solution to the original problem (3.1) is written in terms of u_j's as $u(x) = \sum_{j=1}^N u_j|_{\Omega_j}(x)$, where $u|_{\Omega_j}(x) = u(x)$ if $x \in \Omega_j$ and 0 if $x \notin \Omega_j$. The XPINN aims at approximating each u_j using a NN, say $v_j \in V_n$. While XPINNs can be formulated in many different ways (e.g. variational formulation), for ease of discussion, we focus on the original formulation based on the strong form. Let us define the interface term by

$$\begin{aligned} \mathrm{L}_{\mathrm{if}}[v,u](x) = \beta_0(v(x)-u(x))^2 + \beta_1\|\nabla v(x) - \nabla u(x)\|_2^2 \\ + (\mathcal{D}[v](x) - \mathcal{D}[u](x))^2, \end{aligned}$$

for some $\beta_0, \beta_1 \geq 0$, where "if" stands for interface.

Let $\Omega_{M_j} \subset \Omega_j$, $\partial\Omega_{M_j'} \subset \partial\Omega_j$, $\Gamma_j = \Gamma \cap \partial\Omega_{M_j'}$, and $\partial\Omega_{M_{ij}'} \subset \partial\Omega_{M_i'} \cap \partial\Omega_{M_j'}$ be finite sets of interest and consider the corresponding empirical probability distributions: $\omega_j(\cdot) = \frac{1}{M_j}\sum_{x\in\Omega_{M_j}} \delta_x(\cdot)$, $\mu_j(\cdot) = \frac{1}{|\Gamma_j|}\sum_{x\in\Gamma_j}\delta_x(\cdot)$, and $\bar{\mu}_{ij} = \frac{1}{M_{ij}'}\sum_{x\in\partial\Omega_{M_{ij}'}}\delta_x(\cdot)$. Then, the loss functional for XPINNs is given by

$$\begin{aligned} \mathcal{L}[v_1,\ldots,v_N] = \sum_{j=1}^N \Bigg[\int_{\Omega_j} \mathrm{L}_{\mathrm{re}}[v_j](x)d\omega_j(x) + \int_{\Gamma_j} \mathrm{L}_{\mathrm{bc}}[v_j](x)d\mu_j(x) \\ + \sum_{i\neq j}\int_{\partial\Omega_i\cap\partial\Omega_j} \mathrm{L}_{\mathrm{if}}[v_i,v_j](x)d\bar{\mu}_{ij}(x)\Bigg], \end{aligned}$$

where v_j's are all in V_n (e.g. a class of neural networks). Let $\{u_j^*\}_{j=1}^N$ be a minimizer of the loss functional. Then, the NN approximation by the XPINN formulation is given by

$$u_{\mathrm{XPINN}}(x) = \sum_{j=1}^{N} u_j^*(x) \cdot \mathbb{I}_j(x), \qquad \mathbb{I}_j(x) = \begin{cases} 0 & \text{if } x \notin \Omega_j \\ \frac{1}{S_j(x)} & \text{if } x \in \partial\Omega_j \\ 1 & \text{otherwise,} \end{cases}$$

where $S_j(x)$ represents the number of subdomains whose boundary contains x. Note that the XPINN does not necessarily preserve the continuity at interfaces.

The performance of XPINN can be enhanced by adding the jump condition at the interface, e.g., in Jagtap et al. (2020) and De Ryck et al. (2023) in the loss function. In Hu et al. (2022), it is shown that XPINN is not necessarily performing better than PINN. The XPINN can have less complex parts and simple locally but less data in each part can lead to overfitting and thus affect the accuracy.

More aspects of domain decompositions techniques have been explored, such as in Sheng and Yang (2022) (variational formulation) and Shukla et al. (2021) addressing parallel computation, Meng et al. (2020) and Penwarden et al. (2023) for time domains, Kim and Yang (2023) using overlapping subdomains, and Kopaničáková et al. (2023) using preconditioning, and Sun et al. (2023) assigning different boundary conditions on different subdomains.

3.5 Useful techniques

We discuss some techniques and methods that may improve the performance of the aforementioned formulations.

Enforcement of boundary conditions. The aforementioned NN approaches do not satisfy the boundary conditions from the beginning and require an additional loss term which penalizes the discrepancy between the NN prediction on boundary and the given boundary conditions. This could be cumbersome when it comes to the PINN method as well as many variational approaches. In general, the loss functionals comprise multiple terms and each term may have significantly different scales. If this is the case, training algorithms would focus on minimizing the largest term while the other terms marginally change.

In principle, one can use the distance function and modify the NN structure. The distance function d to the boundary $\partial\Omega$ is defined to be the function that gives the shortest distance between any point x to $\partial\Omega$. Thus, for example,

$$\overline{u}_{\mathrm{NN}}(x;\theta) = \overline{g}(x) + \mathrm{d}(x) u_{\mathrm{NN}}(x;\theta),$$

where $\overline{g}$ is a smooth extension of g, will exactly satisfy the Dirichlet boundary condition as $\mathrm{d}(x) = 0$ on $\partial\Omega$. However, the numerical solution here is only Lipschitz continuous as $\mathrm{d}(x)$ is. Several works followed this principle with some variations (see Lagaris et al., 1998; Berg and Nyström, 2018). This idea can be further generalized. In particular, the work of Sukumar and Srivastava (2022) used the theory of R-functions to construct an approximate distance function ξ that allows one to satisfy exactly inhomogeneous Dirichlet, Neumann, and Robin boundary conditions on complex geometries.

Constraints. Instead of modifying NN structures, one can formulate the problem as a constrained optimization problem via the augmented Lagrangian method. For example, the PINN loss may be written as

$$\min_{\theta} \mathcal{L}_{\mathrm{re}}(\theta) \quad \text{subject to} \quad \mathcal{B}[u_{\mathrm{NN}}(\cdot;\theta)](x) = g(x) \quad \forall x \in \Gamma_{M'},$$

which takes the boundary conditions as constraints. The method of Lagrange multipliers will then construct the Lagrangian defined by $\mathcal{L}(\theta,\lambda) = \mathcal{L}_{\mathrm{re}}(\theta) + \langle \lambda, h(\theta)\rangle$ where $h(\theta) := (\mathcal{B}[u_{\mathrm{NN}}(\cdot;\theta)](x) - g(x))_{x\in\Gamma_{M'}}$ and seek to solve the minmax problem of $\max_\lambda \min_\theta \mathcal{L}(\theta,\lambda)$. The augmented Lagrangian method can also apply, which seeks to solve the minmax problem of

$$\max_{\lambda} \min_{\theta} \mathcal{L}(\theta,\lambda) + \beta \|h(\theta)\|^2,$$

for some appropriately chosen $\beta > 0$. Since $\|h(\theta)\|^2 = M'\mathcal{L}_{\mathrm{bc}}(\theta)$, this may be viewed as a combination of the original PINN loss and the Lagrangian. The work of Son et al. (2023) proposed this formulation, namely, the augmented Lagrangian relaxation method for PINNs. See also Lu et al. (2021b) where the augmented Lagrangian is used to enforce constraints.

Continuity equations. One can impose a certain structure into the NN. For example, we can define a divergence-free NN by using a curl field, e.g., in Richter-Powell et al. (2022).

Sobolev formulation. In the design of loss functions, we can take Sobolev spaces for Y and Z in (3.6) instead of L^2 or L^p spaces. For example, we can take $Y = H^1(\Omega)$, where the norm of $v \in Y$ is defined by $\sqrt{\|v\|_{L^2}^2 + \sum_{i=1}^d \|\partial_{x_i} v\|_{L^2}^2}$. Similarly, we can consider $Z = H^1(\Gamma)$ instead of $L^2(\Gamma)$. Then we can apply proper discretizations on the integrals and obtain a loss functional for training, see e.g., in Yu et al. (2022) and Son et al. (2021). Even higher derivatives of the residuals are considered in Yu et al. (2022) and Son et al. (2021).

Adaptive activation function. The motivation of adaptive activation functions (Jagtap et al., 2022b) is that different activation functions can significantly affect the spectral gaps of neural networks and thus the training results. Two extreme cases are sine or hyperbolic tangent (smooth) and the ReLu (nonsmooth). To appreciate different activation functions, one can set the form of the activation functions as $\sigma(\cdot) = \sum_{i=1}^r a_i \sigma_i(\cdot)$, where σ_i's are the commonly used activation functions which users prefer and a_i's are trainable logical values.

Separable PINNs. In separable PINNs (Cho et al., 2022), the following network architecture $\sum_{j=1}^r \prod_{i=1}^d N_{i,j}(x_i)$ is used, where $N_{i,j}(x_i)$ is a feedforward neural network in x_i only. Compared to the feedforward neural networks as a function of $(x_1, \ldots, x_d)$ in (2.1), PINNs can be trained faster by 10-100 times. The acceleration is due to the tensor-product nature of the network architecture, which facilitates auto-differentiation. Similar network architectures of low rank are considered in Wang et al. (2022a,b); Zhang et al. (2023a).

PINNs for high dimensions. A random batch method in dimensionality has been developed in Hu et al. (2023), where the computational cost of derivatives of feed-forward neural networks consists of computing the derivative in a few randomly picked dimensions in batches and thus the computational time becomes feasible even when the dimension is very high.

4 Approximation error for PINN in strong formulations

Motivated by the stability, we can directly derive an (posterior) error estimate. Let $u* \in X$ be a solution to (3.1).

Assume that f and g are continuous (otherwise we use their approximations which can be evaluated at the sampling points). Assume the neural network u_{NN} is smooth enough such that $\mathcal{D}u_{NN}$ and $\mathcal{B}u_{NN}$ is well-defined at sampling points. Suppose that $\epsilon = 0$, i.e., we have no regularization.

4.1 A posteriori estimate

Let $X = L^2(\Omega)$ and $Y = L^2(\Gamma)$. By the conditional stability (3.3), we obtain

$$\begin{aligned}\mathcal{N}(u_{NN} - u^*) &\le \rho_C(\|\mathcal{D}u_{NN} - f\|_Y + \|\mathcal{B}u_{NN} - g\|_Z) \\ &\le \rho_C\sqrt{2\,\|\mathcal{D}u_{NN} - f\|_Y^2 + 2\,\|\mathcal{B}u_{NN} - g\|_Z^2}) \\ &= \rho_C(\sqrt{2\mathcal{L}(v,\lambda)} + \sqrt{2\epsilon_{\text{appr, f}} + 2\epsilon_{\text{appr, g}}})\end{aligned}$$

where we have applied the inequality $\sqrt{a+b} \le \sqrt{a} + \sqrt{b}$ for $a, b \ge 0$ and denote

$$\epsilon_{\text{appr,f}} = \|\mathcal{D}u_{NN} - f\|_Y^2 - \sum_{x\in\Omega_M} \frac{1}{\#\Omega_M}(\mathcal{D}u_{NN} - f)^2(x),$$

$$\epsilon_{\text{appr,g}} = \|\mathcal{B}u_{NN} - g\|_Y^2 - \sum_{x\in\Gamma_M} \frac{1}{\#\Gamma_M}(\mathcal{B}u_{NN} - g)^2(x)$$

Both quantities can be estimated by the upper bounds of their corresponding Radmacher complexity. If the sampling points are quadrature points and weights are points-dependent, we may improve the bounds of these quantities by errors of quadrature points, if all functions are smooth. We refer to Shin et al. (2023) (applying Radmacher complexity) and Mishra and Molinaro (2023, 2022) (using quadrature bounds) for more detailed analysis.

4.2 A priori estimate

Let $\mathcal{S}u^*$ be a good mollifier of u^* so that one can make of the derivatives of u^* at the sampling points. By the conditional stability (3.3), we obtain

$$\begin{aligned}\mathcal{N}(u_{NN}-u^*) &\le \rho_C(\|\mathcal{D}u_{NN}-f\|_Y+\|\mathcal{B}u_{NN}-g\|_Z)\\ &\le \rho_C(\underbrace{\left\|\mathcal{D}(u_{NN}-\mathcal{S}u^*)\right\|_Y+\left\|\mathcal{B}(u_{NN}-\mathcal{S}u^*)\right\|_Z}_{\text{approximation}}\\ &\quad+\underbrace{\left\|f-\mathcal{D}\mathcal{S}u^*\right\|_Y+\left\|(g-\mathcal{B}\mathcal{S}u^*)\right\|_Z)}_{\text{mollifying}}.\end{aligned}$$

Observe that $\|\mathcal{D}(u_{NN}-\mathcal{S}u^*)\|_Y+\|\mathcal{B}(u_{NN}-\mathcal{S}u^*)\|_Z$ can be bounded by certain norms of $u_{NN}-\mathcal{S}u^*$ and their derivatives. Thus this error can be small by the universal approximation theorem of the underlying neural networks and convergence rate can be quantified. The mollifying error can be estimated by the classical mollifier $\int_{\mathbb{R}^d}\epsilon^{-d}\varphi(\frac{x-y}{\epsilon})u(y)\,dy$, where $\epsilon^{-d}\varphi(\frac{x-y}{\epsilon})$ is a compactly supported function approximating the Dirac delta function in the distribution sense. With some mild moment conditions on φ, the convergence order of mollifier depends on the smoothness of the solution u^*. Also, $\mathcal{S}u^*$ can be some approximate solution by certain numerical methods. A similar idea is presented in Shin et al. (2023) in more details.

The strong formulation may require high smoothness which the solution may not admit. While the estimate here is insightful, it may not be useful in practice, especially when solutions are not smooth. This is a strong motivation of using weak formulations.

5 Training/optimization methods

The PINN methodology requires to solve an optimization problem. In practice, one needs to properly choose a numerical optimization algorithm to solve it. First-order gradient-based optimization algorithms have been popularly employed in this regard mainly due to their simple implementations and reasonably good empirical performance for diverse problems. We briefly discus several numerical optimization algorithms for the PINN methodology.

Let V_n be a class of neural networks, e.g., V_L defined in (2.2). Optimization problems defined on V_n are equivalently written as optimization problems on a finite-dimensional space, e.g., $\mathbb{R}^{|\vec{n}_L|}$. That is, by letting $\mathcal{L}(\theta):=\mathcal{L}(u_{\mathrm{NN}}(\cdot;\theta))$ for any $u_{\mathrm{NN}}\in V_L$, we are concerned with solving the following minimization problem

$$\min_{\theta\in\mathbb{R}^{|\vec{n}_L|}}\mathcal{L}(\theta). \tag{5.1}$$

In what follows, we briefly discuss popular optimization methods and strategies for the PINN methodology.

5.1 Initialization schemes

How to initialize hyperparameters of neural networks plays a pivotal role when it comes to iterative algorithms. As illustrated by the convergence analysis of Newton's method, a well-chosen initialization can dramatically improve the performance in practice. Several works have studied random initialization schemes for deep NNs. Popular initialization schemes use either Gaussian or uniform distributions for weight matrices and set bias vectors being zeros. See the table below. The underlying principle of many initialization schemes is to make the unit variance to avoid the so-called exploding/vanishing gradient, which refers to the phenomenon in which deep NNs yield either too big or too small gradients. Very large gradients will make training algorithms diverge and very small gradients will take forever for algorithms to converge. This requires one to compute the variance with respect to all the weights.

The Glorot initialization (Glorot and Bengio, 2010) was proposed based on the linearity assumption. Suppose $\phi(x) = x$, also known as the linear activation function). Since all the bias vectors are initialized to zeros, assuming W^ℓ's are independently initialized whose element has mean zero and variance σ_ℓ^2, it can be checked that the variance of the i-th element of u^ℓ is

$$\mathrm{Var}[u_i^\ell(x)] = \prod_{j=1}^{\ell} (n_{j-1}\sigma_j^2) \cdot \|x\|_2^2 \quad \forall 1 \le i \le n_\ell.$$

Let $s^\ell(x) = W^\ell u^{\ell-1}(x) + b^\ell$. It follows from the chain rule that

$$\frac{\partial u_{\mathrm{NN}}}{\partial s_k^\ell} = \left(\frac{\partial s^{\ell+1}}{\partial s_k^\ell}\right)^\top \frac{\partial u_{\mathrm{NN}}}{\partial s^{\ell+1}} = \phi'(s^\ell)(W_{:,k}^{\ell+1})^\top \frac{\partial u_{\mathrm{NN}}}{\partial s^{\ell+1}},$$

where $W_{:,k}^{\ell+1}$ is the k-th column of $W^{\ell+1}$. Since $\phi'(x) = 1$ and $\frac{\partial u_{\mathrm{NN}}}{\partial s^{\ell+1}} = (W^L \cdots W^{\ell+2})^\top$, we have

$$\mathrm{Var}\left[\frac{\partial \mathcal{L}}{\partial s_k^\ell}\right] = \prod_{j=\ell+1}^{L} (n_j\sigma_j^2) \cdot \|x\|_2^2 \quad \forall 1 \le k \le n_\ell.$$

The Glorot initialization aims at satisfying

$$\mathrm{Var}[u_k^\ell(x)] = \mathrm{Var}[u_1^1(x)], \qquad \mathrm{Var}\left[\frac{\partial \mathcal{L}}{\partial s_k^\ell}\right] = \mathrm{Var}\left[\frac{\partial \mathcal{L}}{\partial s_1^1}\right], \qquad \forall \ell, k,$$

which are equivalent to $n_\ell\sigma_\ell^2 = 1$ and $n_{\ell-1}\sigma_\ell^2 = 1$ for all ℓ. Since the two are met only if $n_\ell = n_{\ell-1}$, as a compromise, the Glorot initialization (Glorot and Bengio, 2010) sets

$$\text{(Glorot initialization)}: \quad \sigma_\ell^2 = \frac{2}{n_\ell + n_{\ell-1}}.$$

The Kaiming initialization scheme (He et al., 2015) was developed based on the rectified linear unit (ReLU) activation function. Let $\phi(x) = \max\{x, 0\}$ and suppose that the bias vectors are initialized to zeros and the weight matrices are independently initialized from symmetric distributions around zero. Let σ_ℓ^2 be the variance of each component of W^ℓ. It follows from the symmetric distribution assumption that $\mathbb{E}[(u_k^{\ell-1}(x))^2] = \frac{1}{2}\mathrm{Var}[s_k^{\ell-1}(x)]$ and thus

$$\begin{aligned}\mathrm{Var}[s_i^\ell(x)] = \sigma_\ell^2 \mathbb{E}[\|u^{\ell-1}(x)\|_2^2] &= \frac{1}{2}\sigma_\ell^2 \sum_{k=1}^{n_{\ell-1}} \mathrm{Var}[s_k^{\ell-1}(x)] \\ &= \frac{1}{2}\sigma_\ell^2 n_{\ell-1} \mathrm{Var}[s_i^{\ell-1}(x)].\end{aligned}$$

The Kaiming initialization enforces $\mathrm{Var}[s_i^\ell(x)] = \mathrm{Var}[s_1^1(x)]$ for all ℓ and i, which results in the condition of

$$(\text{Kaiming initialization}): \quad \sigma_\ell^2 = \frac{2}{n_{\ell-1}}.$$

	weight matrix	bias vector	hyperparameters	note
Glorot Normal (Glorot and Bengio, 2010)	$\mathcal{N}(0, \sigma^2)$	0	$\sigma^2 = \frac{2}{n_\ell + n_{\ell-1}}$	Linear, Tanh
Glorot Uniform (Glorot and Bengio, 2010)	$\mathcal{U}(-a, a)$	0	$a^2 = \frac{6}{n_\ell + n_{\ell-1}}$	
Kaiming Normal (He et al., 2015)	$\mathcal{N}(0, \sigma^2)$	0	$\sigma^2 = \frac{2}{n_\ell}$	ReLU and variants
Kaiming Uniform (He et al., 2015)	$\mathcal{U}(-a, a)$	0	$a^2 = \frac{6}{n_\ell}$	
RAI (Lu et al., 2020)	$\mathcal{N}(0, \sigma^2) + \mathrm{Beta}(2, 1)$		$\sigma^2 = \frac{0.6007^2}{n_{\ell-1}}$	Preventing Dying ReLU

Dying ReLU. Dying ReLU refers to the phenomenon in which ReLU neurons are inactivated and return zeros for all inputs. It has been empirically observed that this could happen not only at the initialization but also during training. If a deep NN was initialized as a constant function, i.e., all the neurons were dead, the network cannot be trained by gradient-based algorithms as the gradient is zero. The work of Lu et al. (2020) estimated the probability of such an event (namely "born dead probability" in Lu et al. (2020)) with standard initialization schemes including Glorot (Glorot and Bengio, 2010) and He (He et al., 2015).

Theorem 5.1 (Theorem 3.2 of Lu et al., 2020). *Let $\vec{\boldsymbol{n}} = (n_1, \ldots, n_{L-1})$ be an architecture. Suppose that all weights are independently initialized from symmetric continuous probability distributions around zero, and all biases are either drawn from a symmetric distribution or set to zero. Then, the probability that the initialized deep ReLU NN is born dead is upper bounded by $1 - \prod_{\ell=1}^{L-1}(1 - 2^{-n_\ell})$.*

The randomized asymmetric initialization (RAI) was proposed in Lu et al. (2020) to alleviate the dying ReLU by breaking the symmetry in distributions. Let $V^\ell = [W^\ell, b^\ell] \in \mathbb{R}^{n_\ell \times (n_{\ell-1}+1)}$. The RAI scheme works as follows: $W^1_{ij} \sim N(0, \frac{2}{d_{\text{in}}}$ and $b^1 = 0$ as is in the Kaiming normal initialization. For every $\ell \geq 2$ and $j \in \{1, \ldots, n_\ell\}$, (a) randomly select an index k^ℓ_j from $\{1, \ldots, n_{\ell-1}, n_{\ell-1}+1\}$, and (b) initialize $(V^\ell_j)_{-k^\ell_j} \sim \mathcal{N}(0, \sigma^2_\ell I)$ and $(v^\ell_j)_{k^\ell_j} \sim \text{Beta}(2, 1)$ where $\sigma^2_\ell = \frac{0.6007^2}{n_{\ell-1}}$, V^ℓ_j is the j-th row of V^ℓ, $(V^\ell_j)_k$ is the k-th component of V^ℓ_j, $(V^\ell_j)_{-k}$ is a vector constructed from V^ℓ_j by omitting the k-th component. Here 0.6007 is an approximation of $-\frac{2\sqrt{2}}{3\sqrt{\pi}} + \sqrt{1 + \frac{8}{9\pi}}$, which is derived from the second moment analysis (Lu et al., 2020).

Data-dependent bias initialization. For two-layer NNs, the aforementioned schemes often do not perform well as these schemes were meant for deep NNs. In Shin and Karniadakis (2020), the data-dependent bias initialization scheme for two-layer ReLU NNs was proposed which located neurons at data points. Given W^1, the scheme initializes the bias vector b^1 according to $(b^1)_k = -w_k^\top x_{i_k}$ where x_{i_k} is a randomly selected input data point. By construction, the two-layer ReLU NN at initialization is written as

$$\text{(Data-dependent Bias Initialization)}: \quad u_{\text{NN}}(x) = \sum_{k=1}^{n_1} W^2_{:,k} \phi(W^1_k (x - x_{i_k})),$$

$$\text{(Zero Bias Initialization)}: \quad u_{\text{NN}}(x) = \sum_{k=1}^{n_1} W^2_{:,k} \phi(W^1_k x),$$

where $W^2_{:,k}$ is the k-th column of W^2. When the biases are initialized as zeros, all initialized neurons are clustered at the origin. Consequently, it would take a long time for training algorithms to distribute neurons over the training domain to achieve a small training loss. In the worst case, along the way of distributing neurons, it will get stuck at the local minimum. The data-dependent bias initialization ensures that all neurons are randomly distributed over the training data points.

5.2 Generic methods: stochastic gradient descent

The first-order gradient-based optimization algorithms have been popularly employed when it comes to (5.1) due to their simple implementation and reasonably good empirical performance.

The vanilla gradient descent (GD) algorithm commences with a random guess $\theta^{(0)}$ (following one of the initialization schemes) and updates the parameter according to the rule: for $k = 1, 2, \ldots,$

$$\text{(Full-batch GD)}: \quad \theta^{(k)} = \theta^{(k-1)} - \eta_k \cdot \nabla \mathcal{L}(\theta^{(k-1)}), \tag{5.2}$$

where η_k is the learning rate or the stepsize for the k-th iteration. This is also known as the full-batch training.

If the underlying computational domain is in either 3D or 4D (including time), the discrete loss evaluation may require a large number of points in computational domains. Consequently, the computation of the gradient becomes very expensive. To alleviate such difficulty, (stochastic) mini-batch GD approaches are popularly employed. There are two main approaches. The first one is the subsampling from a fixed set containing a large number of points. Let $\Omega_M \subset \Omega$ be a fixed set with $M \gg 1$. Let m be a reasonably small number satisfying $1 \le m \ll M$, which is referred to as the batch size. In the PINN formulation, the full gradient of $\mathcal{L}$ is written as

$$\nabla\mathcal{L}(\theta) = \frac{1}{M}\sum_{x\in\Omega_M} \nabla \mathrm{L}_{\mathrm{re}}[\theta](x) + \lambda\nabla\mathcal{L}_{\mathrm{bc}}(\theta),$$

where $\mathrm{L}_{\mathrm{re}}[\theta](x) := \mathrm{L}_{\mathrm{re}}[u_{\mathrm{NN}}(\cdot;\theta)](x)$. It can be seen that the full gradient requires M evaluations of $\nabla\mathrm{L}_{\mathrm{re}}[\theta](\cdot)$ which can be computationally demanding as $M \gg 1$. The computation cost grows linearly with respect to the number M of sample points. Thus, the mini-batch training is popularly used in practice. For the k-th iteration, let $\Omega_m^{(k)} \subset \Omega_M$ and define

$$\nabla\mathcal{L}(\theta^{(k-1)};\Omega_m^{(k)}) := \frac{1}{|\Omega_m^{(k)}|}\sum_{x\in\Omega_m^{(k)}} \nabla\mathrm{L}_{\mathrm{re}}[\theta^{(k-1)}](x) + \lambda\nabla\mathcal{L}_{\mathrm{bc}}(\theta), \tag{5.3}$$

whose computation requires only m evaluations of $\nabla\mathrm{L}_{\mathrm{re}}[\theta](\cdot)$. The mini-batch gradient descent algorithm then updates the parameter according to the rule:

$$\text{(Mini-batch GD)}: \quad \theta^{(k)} = \theta^{(k-1)} - \eta_k \cdot \nabla\mathcal{L}(\theta^{(k-1)};\Omega_m^{(k)}), \tag{5.4}$$

where the sampling set $\Omega_m^{(k)}$ varies over iterations. Depending on how these sampling sets are chosen, different mini-batch schemes are obtained. Let $\Omega_M = \{x_j\}_{j=1}^M \subset \Omega$ and let $M = qm + r$ where q is the quotient and $r \in \{1,\ldots,m\}$ is the remainder. Define

$$I_k = \begin{cases} \{1+(k-1)m,\ldots,m+(k-1)m\} & \text{if } k \not\equiv 0 \pmod{q+1}, \\ \{1+qm,\ldots,M\} & \text{if } k \equiv 0 \pmod{q+1}. \end{cases}$$

Let S_M be the permutation group of the set $\{1,\ldots,M\}$. Whenever $(k-1) = s(q+1)$ for some nonnegative integer s, let $\boldsymbol{\sigma}_s$ be a randomly uniformly chosen element from S_M and define

$$\sigma^{(k)} := \begin{cases} \boldsymbol{\sigma}_s & \text{if } k \equiv 1 \pmod{q+1}, \\ \sigma^{(k-1)} & \text{if } k \not\equiv 1 \pmod{q+1}. \end{cases}$$

We are now in a position to describe some popular mini-batch schemes:

$$\begin{aligned}
\text{(Mini-batch without shuffling)}: &\quad \Omega_m^{(k)} = \{x_j\}_{j \in I_k}, \\
\text{(Mini-batch with shuffling)}: &\quad \Omega_m^{(k)} = \{x_{\sigma^{(k)}(j)}\}_{j \in I_k}, \\
\text{(Mini-batch with replacement)}: &\quad \Omega_m^{(k)} \sim \mathrm{P},
\end{aligned}$$

where P is the uniform probability distribution over the set of all subsets containing m points from Ω_M. Among the above three schemes, the mini-batch without shuffling is considered as deterministic while the other two are stochastic. An epoch is a unit that refers to the number of iterations taken to sweep the entire M points ($q+1$ iterations). In the full batch training, an epoch is equivalent to an iteration.

The other is based on random sampling from the underlying probability measure ω defined on Ω. The distinct feature of this approach is that the reference set Ω_M needs not to be formed. Rather, every iteration requires newly sampled fresh m points from the underlying probability distribution ω. Specifically,

$$\begin{aligned}
\text{(Stochastic Mini-batch)}: \quad \Omega_m^{(k)} &= \{x_j^{(k)}\}_{j=1}^m \\
&\text{where} \quad \{x_j^{(k)}\} \subset \Omega \text{ are i.i.d. samples from } \omega.
\end{aligned}$$

At every iteration, one has the option to design an appropriate learning rate or stepsize η_k. It has been widely known that the learning rates should be chosen sufficiently small for convergence and, at the same time, sufficiently large for making meaningful progress within practically reasonable maximum epochs/iterations (e.g. 100K or 1M). From the optimization perspective, the optimal learning rate is possibly found by following the minimization principle. Given the current update direction $d^{(k-1)}$ (e.g. $d^{(k-1)} = -\nabla\mathcal{L}(\theta^{(k-1)}; \Omega_m^{(k)})$), the principle seeks the best learning rate that solves

$$\eta_k^* = \operatorname*{argmin}_{\eta \in \mathbb{R}} \mathcal{L}\left(\theta^{(k-1)} + \eta d^{(k-1)}\right).$$

However, the optimization problem is difficult to solve in general and the optimum solution is only available for a small class of problems (e.g. convex, linear). In addition, the mini-batch version of the minimization rule remains elusive which further prevents its use in practice. Hence, practically popular designs heavily rely on users' trial-and-errors, which require problem-dependent hyperparameters to be tuned appropriately.

Let T be a positive integer representing the number of iterations on which the learning rate remains unchanged. While the initial learning rate η_0 is chosen in a problem-dependent manner, some popularly used values lie in the range of $[10^{-5}, 10^{-3}]$. The decay factor is denoted by γ and the maximum and minimum learning rates are denoted by $\eta_{\max}$ and $\eta_{\min}$, respectively. Some popular schedulers include

$$\begin{aligned}
\text{[Constant learning rate]} \quad & \eta_k = \eta_0, \\
\text{[Step Scheduler]} \quad & \eta_k = \eta_0 \gamma^{\lfloor \frac{k}{T} \rfloor}, \\
\text{[Exponential Scheduler]} \quad & \eta_k = \eta_0 \gamma^k, \\
\text{[Cosine Scheduler]} \quad & \eta_k = \eta_{\min} + \frac{\eta_{\max} - \eta_{\min}}{2}(1 + \cos(\pi k / T)).
\end{aligned}$$

In practice, especially when it comes to the PINN methodology, the vanilla (stochastic) GD is rarely used, rather, their variants are typically employed. One of the most popular variants is the adaptive moment estimation algorithm, widely known as `Adam`. The update rule of `Adam` is given as follows. For $\beta_1, \beta_2 \in [0, 1)$ whose default values are 0.9 and 0.999, respectively, let $m^{(0)} = 0$, $v^{(0)} = 0$ and $\epsilon = 10^{-8}$. Given an initialization $\theta^{(0)}$, the k-th iterated parameter is given by

$$(\texttt{Adam}): \quad \theta^{(k)} = \theta^{(k-1)} - \eta_k \cdot \hat{m}^{(k)} / (\sqrt{\hat{v}^{(k)}} + \epsilon),$$

where $\hat{m}^{(k)} = \frac{1}{1-\beta_1^k} m^{(k)}$, $\hat{v}^{(k)} = \frac{1}{1-\beta_2^k} v^{(k)}$, $m^{(k)} = \beta_1 m^{(k-1)} + (1 - \beta_1) g^{(k-1)}$, $v^{(k)} = \beta_2 v^{(k-1)} + (1 - \beta_2) g^{(k-1)} \otimes g^{(k-1)}$, and $g^{(k-1)} = \nabla \mathcal{L}(\theta^{(k-1)}; \Omega_m^{(k)})$. All the operations are understood element-wise.

5.3 Quasi-Newton methods of 1.5-order

While first-order gradient-based optimization algorithms are popularly used, they often suffer from either being stuck at local minima or slow convergence. Perhaps, a natural extension of the first-order methods is the optimization methods that (approximately) use the second-order information, i.e., Hessian. Newton's method is the representative one. While it could achieve the quadratic rate of convergence, the computation of Hessian is computationally very expensive if optimization problems are defined in a high dimension, which is the case for NNs as the number of NN parameters is large. Therefore, the Gauss-Newton type algorithms are popularly used which aim at approximating Hessian using only gradient information. These algorithms are called 1.5-order methods.

The underlying idea of higher-order algorithms is to appropriately design a local proxy function on which the minimum can be found easily. More precisely, let U be a neighborhood of the current parameter $\theta^{(k-1)}$ and one seeks to design an approximation $\tilde{\mathcal{L}}$ on U. The minimizer of the proxy function is then set to the update rule. A natural proxy function comes from the Taylor expansion:

$$\begin{aligned}
\mathcal{L}(\theta) \approx \mathcal{L}(\theta^{(k-1)}) &+ \langle \nabla \mathcal{L}(\theta^{(k-1)}), \theta - \theta^{(k-1)} \rangle \\
&+ \frac{1}{2}(\theta - \theta^{(k-1)})^\top H_{k-1} (\theta - \theta^{(k-1)}) + \dots,
\end{aligned}$$

where H_{k-1} is the Hessian of $\mathcal{L}$ at $\theta^{(k-1)}$. The Taylor approximation degree up to one yields

$$\tilde{\mathcal{L}}_1(\theta) = \mathcal{L}(\theta^{(k-1)}) + \langle \nabla \mathcal{L}(\theta^{(k-1)}), \theta - \theta^{(k-1)} \rangle.$$

Seeking the minimizer of $\tilde{\mathcal{L}}_1$ then gives the gradient descent (also known as the steepest decent) method, i.e., $\theta^* = \theta^{(k-1)} - \eta_k \cdot \nabla \mathcal{L}(\theta^{(k-1)})$. The Taylor approximation degree up to two provides the multivariate polynomial of degree up to two involving the Hessian

$$\begin{aligned}\tilde{\mathcal{L}}_2(\theta) = \mathcal{L}(\theta^{(k-1)}) &+ \langle \nabla \mathcal{L}(\theta^{(k-1)}), \theta - \theta^{(k-1)} \rangle \\ &+ \frac{1}{2\eta}(\theta - \theta^{(k-1)})^\top H_{k-1}(\theta - \theta^{(k-1)}).\end{aligned}$$

Assuming the Hessian is invertible, the minimizer of $\tilde{\mathcal{L}}_2$ is explicitly written as

$$\text{(Newton's method)} \quad \theta^{(k)} := \theta^* = \theta^{(k-1)} - \eta_k \cdot H_{k-1}^{-1} \nabla \mathcal{L}(\theta^{(k-1)}),$$

which defines Newton's update rule. Since the Hessian is expensive to compute, the 1.5-order optimization algorithms seek to find alternative proxy functions that do not require the Hessian. A generic proxy function may be written as

$$\begin{aligned}\tilde{\mathcal{L}}_{1.5}(\theta; B) = \mathcal{L}(\theta^{(k-1)}) &+ \langle \nabla \mathcal{L}(\theta^{(k-1)}), \theta - \theta^{(k-1)} \rangle \\ &+ \frac{1}{2\eta_k}(\theta - \theta^{(k-1)})^\top B(\theta - \theta^{(k-1)}),\end{aligned}$$

where B is an appropriately chosen matrix that does not require the second order partial derivatives. The 1.5-order method is then determined by the minimizer of $\tilde{\mathcal{L}}_{1.5}(\theta; B)$, i.e.,

$$\text{(1.5-order method)} \quad \theta^{(k)} := \theta^* = \theta^{(k-1)} - \eta_k \cdot B^{-1} \nabla \mathcal{L}(\theta^{(k-1)}).$$

The quasi-Newton type algorithms are based on the principle of seeking the Hessian approximation with the exact gradient. That is, it aims at finding B such that $\nabla \mathcal{L}(\theta^{(k-1)} + \mu) = \nabla \tilde{\mathcal{L}}_{1.5}(\theta^{(k-1)} + \mu; B)$ where the gradient is taken with respect to μ. It then can be checked that μ and B should satisfy the so-called secant equation

$$\text{(quasi-Newton)} \quad \nabla \mathcal{L}(\theta^{(k-1)} + \mu) = \nabla \mathcal{L}(\theta^{(k-1)}) + \eta_k^{-1} B \mu.$$

The quasi-Newton type algorithms iteratively find the solution μ_k, B_k to the secant equation. The update rule for μ_k is determined by the first-order optimality condition, i.e., $\nabla \mathcal{L}(\theta^{(k-1)}) + \eta_k^{-1} B_k \mu_k = 0 \iff \mu_k = -\eta_k \cdot B_k^{-1} \nabla \mathcal{L}(\theta^{(k-1)})$ where the learning rate η_k is typically chosen to satisfy the Wolfe conditions. Hence, the quasi-Newton algorithm reads $\theta^{(k)} = \theta^{(k-1)} + \mu_k$. The core part of the quasi-Newton algorithms is on the update formula for B_k. While several formulas exist, in the context of the PINN methodology, the BFGS (Broyden–Fletcher–Goldfarb–Shanno) algorithm is one of the most popular ones. The BFGS updates B_k according to $B_{k+1} = B_k + \frac{y_k y_k^\top}{y_k^\top \mu_k} - \frac{B_k \mu_k (B_k \mu_k)^\top}{\mu_k^\top B_k \mu_k}$, where $y_k = \nabla \mathcal{L}(\theta^{(k-1)} + \mu_k) - \nabla \mathcal{L}(\theta^{(k-1)})$.

For nonlinear least squares problems, which are equivalent to minimizing a sum of squared function values, the Gauss-Newton type algorithms are popularly used. These algorithms also follow the above principle and are obtained by appropriate designs of B. The Levenberg-Marquart algorithm is one of the most well-known algorithms of this type, whose design is given by

$$\text{(Levenberg-Marquart)} \quad B = \lambda I + \mathrm{J}_r^\top \mathrm{J}_r \quad \text{or} \quad \lambda \mathrm{diag}(\mathrm{J}_r^\top \mathrm{J}_r) + \mathrm{J}_r^\top \mathrm{J}_r.$$

6 Approximation theory with small weights

Both shallow and deep neural networks are universal approximators, however, their training behaviors may vary significantly. Since the NN parameters are initialized as small numbers, deep NNs may be preferred if they remain universal under the constraints on the magnitude of weights and biases. Fig. 1 shows the histogram of the magnitude of weights and biases of a 5-layer NN of width 100 at the initialization. We employ the Kaiming initialization (He et al., 2015), one of the most popular initialization schemes for ReLU NNs. This demonstrates that weights and biases are initialized to small values.

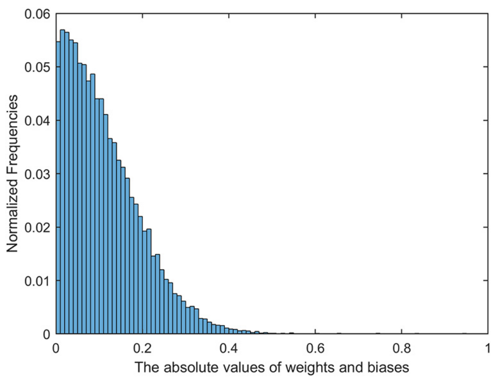

FIGURE 1 The histogram of the magnitude of randomly generated weights and biases for a 5-layer ReLU network of width 100. The Kaiming initialization (He et al., 2015) is employed.

This subsection establishes a universal approximation theorem of ReLU NNs with constraints on the magnitude of weights and biases. The primary focus will lie on the ReLU approximation to functions with (countably many) discontinuities. As a pedagogical example, let us consider a simple learning task of approximating a step function $f^*(x) = \mathbb{I}_{[-1,1]}(x)$ where $\mathbb{I}_A(x)$ is the indicator function where $\mathbb{I}_A(x) = 1$ if $x \in A$ and $\mathbb{I}_A(x) = 0$ if $x \notin A$. It is readily checked that for any $\epsilon > 0$, there exists a two-layer ReLU NN $u_{\text{NN}}(x)$ of width 4 that approximates f^* within the ℓ_1-error of ϵ, i.e., $\int_{\mathbb{R}} |f^*(x) - u_{\text{NN}}(x)| dx = \epsilon$. One such NN can explicitly be written as

$$u_{\text{NN}}(x) = \frac{1}{2\epsilon} \left[\phi(x + 1 + \epsilon) - \phi(x + 1 - \epsilon) - \phi(x - 1 + \epsilon) + \phi(x - 1 - \epsilon)\right],$$

where $\phi(x) = \max\{x, 0\}$. This indicates that if the desired accuracy is $\epsilon = 10^{-3}$ and a two-layer ReLU network, $\sum_{i=1}^{4} c_i \phi(w_i x + b_i)$, is used, the magnitudes of $c_i w_i$ and $c_i b_i$ should be at around 500. It may suggest that many gradient descent iterations are required to make all the $c_i w_i$, $b_i w_i$'s have the magnitude of 500. In Fig. 2, the histograms of NN parameters are plotted that illustrate the changes in magnitudes before and after the training. The top row shows the results with the two-layer (shallow) NNs at varying widths – 4, 16, 64. As expected, the magnitudes of parameters remain small as the width increases. Similar results are observed with the three-layer NNs whose results are shown in the bottom row.

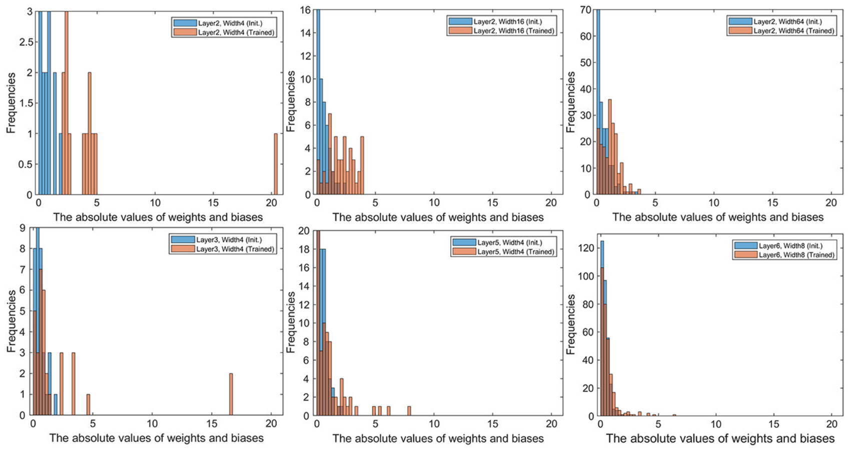

FIGURE 2 The histograms of weights and biases of networks before and after training for the task of approximating $f^*(x) = \mathbb{I}_{[-1,1]}(x)$. (Top row) Two-layer ReLU networks of width 4, 16, 64 (left to right) are employed. (Bottom row) L-layer ReLU networks of width W are employed where $(L, W) = (3, 4), (5, 4), (6, 8)$. Training is done by Adam over 300 equidistant data points from $[-3, 3]$. The maximum number of epochs is either 50,000 or 100,000.

Fig. 3 reports the training loss versus the number of iterations at varying NN architectures. It is seen that there is a tendency that the loss decays faster as the width gets larger and the depth gets larger.

In what follows, we present a universal approximation theorem of deep ReLU NNs with constraints on the magnitudes of NN parameters. We followed the proof structure of Petersen and Voigtlaender (2018), while we made modifications in each step to ensure that the magnitudes of the weights and biases

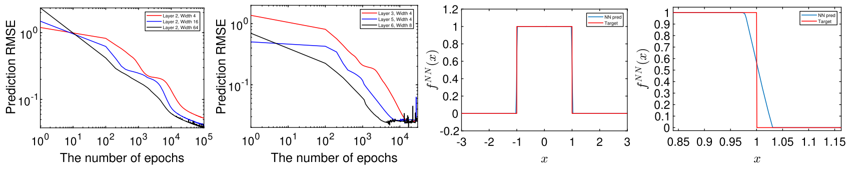

FIGURE 3 The prediction errors with respect to the number of epochs for the task of approximating $f^*(x) = \mathbb{I}_{[-1,1]}(x)$. (a) Two-layer ReLU networks of width 4, 16, 64 (left to right) are employed. (b) L-layer ReLU networks of width W are employed where $(L, W) = (3, 4), (5, 4), (6, 8)$. Training is done by Adam over 300 equidistant data points from $[-3, 3]$. The maximum number of epochs is 100,000.

were well controlled. Consequently, our new results will show the dependency of the magnitudes of NN parameters in the NN architecture.

The class of functions to be approximated contains functions defined on $\Omega = [-1/2, 1/2]^d$ that have the form of $\mathbb{I}_K(x)g(x)$ where g is a smooth function and K is an appropriate compact set in Ω on which $\mathbb{I}_K$ is locally a horizon function. To be more precise, let $r \in \mathbb{N}$, $\beta \in (0, \infty)$ such that $\beta = n + \sigma$ where $n \in \mathbb{N}_0$ and $\sigma \in (0, 1]$, $d \in \mathbb{N}_{\geq 2}$, and $B, p > 0$. Then, $\mathcal{E}^p_{r,\beta,d,B}$ is the target function class that is defined by

$$\mathcal{E}^p_{r,\beta,d,B} := \left\{\mathbb{I}_K(x)g(x) : K \in \mathcal{K}_{r,\beta,d,B},\ g \in \mathcal{F}_{\beta',d,B}\right\}, \tag{6.1}$$

where $\beta' = \frac{d\beta}{p(d-1+\beta)}$, $\mathcal{F}_{\beta',d,B}$ is a class of smooth functions and $\mathcal{K}_{r,\beta,d,B}$ is a class of tailored domains with smooth boundaries. Both classes will be defined shortly.

The smooth function class is defined as follows.

$$\mathcal{F}_{\beta,d,B} = \left\{f \in C^n([-1/2, 1/2]^d) : \|f\|_{C^{0,\beta}} \leq B\right\},$$

where $\|f\|_{C^{0,\beta}} = \max\left\{\max_{|\mathbf{k}|\leq n} \|\partial^{\mathbf{k}} f\|_{C^0}, \max_{|\mathbf{k}|=n} \mathrm{Lip}_\sigma(\partial f)\right\}$ and $\mathrm{Lip}_\sigma(\partial f) := \sup_{x,y\in\Omega, x\neq y} \frac{|f(x)-f(y)|}{\|x-y\|^\sigma}$.

$$\|f\|_{C^{0,\beta}} = \max\left\{\max_{|\mathbf{k}|\leq n} \|\partial^{\mathbf{k}} f\|_{C^0}, \max_{|\mathbf{k}|=n} \mathrm{Lip}_\sigma(\partial f)\right\},$$

$$\mathrm{Lip}_\sigma(\partial f) := \sup_{x,y\in\Omega, x\neq y} \frac{|f(x) - f(y)|}{\|x - y\|^\sigma}.$$

Next, let us introduce the class of horizon functions. Let $d \in \mathbb{N}_{\geq 2}$, $\mathbf{x}_{-1} = (x_2, \cdots, x_d) \in \mathbb{R}^{d-1}$ and $H := \mathbb{I}_{[0,\infty)\times\mathbb{R}^{d-1}}$ be the Heaviside function. We define

$$\begin{aligned}\mathcal{HF}_{\beta,d,B} = \{f \circ T \in L^\infty(\Omega) : f(x) = H(x_1 + \gamma(\mathbf{x}_{-1}), \mathbf{x}_{-1}),\\ \gamma \in \mathcal{F}_{\beta,d-1,B}, T \in \Pi(d, \mathbb{R})\},\end{aligned}$$

where $\Pi(d,\mathbb{R})$ is the group of permutation matrices. We are now in a position to define a class of piecewise smooth functions of interest, which was originally introduced in Petersen and Voigtlaender (2018). Let us consider a set of domains with smooth boundaries. For $r \in \mathbb{N}$, let

$$\mathcal{K}_{r,\beta,d,B} := \big\{K \subset \Omega : \forall y \in \Omega, \exists f_y \in \mathcal{HF}_{\beta,d,B} \\ \text{such that } \mathbb{I}_K(x) = f_y(x) \text{ on } x \in \Omega \cap \overline{B}_{2^{-r}}(y)\big\},$$

where $\overline{B}_{2^{-r}}(y)$ is the closed ball centered at y with the radius of 2^{-r} in $\|\cdot\|_\infty$-norm. Note that every closed set K' in Ω whose boundary is locally the graph of a C^β function of all but one coordinate belongs to $\mathcal{K}_{r,\beta,d,B}$ for sufficiently large r and B.

To give a concrete idea of what functions belong to the class $\mathcal{E}^p_{r,\beta,d,B}$, here we present an example. Let $K = \{(x_1,x_2) \in \Omega : x_1 + 0.5\sin(2\pi x_2) \geq 0\}$ which corresponds to the yellow region shown on the left of Fig. 4. Since $\gamma = 0.5\sin(2\pi x) \in \mathcal{F}_{2,1,2\pi}$ as $\|\gamma\|_{C^{0,2}} = \pi$, K belongs to $\mathcal{K}_{r,\beta,d,B}$ with $r = 1$, $\beta = 2$, $d = 2$, $B = 100$. In particular, for all $y \in \Omega$ and for all $r \in \mathbb{N}$, we have $f_y(x_1,x_2) = H(x_1 + 0.5\sin(2\pi x_2), x_2) \in \mathcal{HF}_{\beta,d,B}$ which is equal to $\mathbb{I}_K$ and shows, in fact, r can be any integer. Lastly, let g be the Rosenbrock function defined by $g(x_1,x_2) = (1-0.5X_1)^2 + 100 * (0.5x_2 - (0.5x_1).^2)^2$, which belongs to $\mathcal{E}^p_{r,\beta,d,B}$ where $\beta' = 2$. Then, the function $\mathbb{I}_K g$ belongs to the function class $\mathcal{E}^2_{1,2,2,100}$ whose graph is shown on the right of Fig. 4.

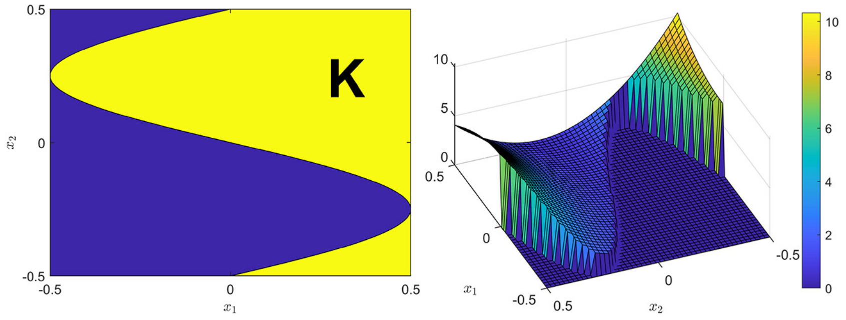

FIGURE 4 Left: The yellow region corresponds to $K = \{(x_1,x_2) \in \Omega : x_1 + 0.5\sin(2\pi x_2) \geq 0\}$ that belongs to the domain class $\mathcal{K}_{r,\beta,d,B}$ with $r = 1$, $\beta = 2$, $d = 2$, $B = 100$. Right: A piecewise smooth function of the form $\mathbb{I}_K(x)g(x) \in \mathcal{E}^p_{r,\beta,d,B}$ where $g(x)$ is the Rosenbrock function.

We now state the main theorem. Let r, n, σ, d, B, p be given with $\beta = n + \sigma$ where $r \in \mathbb{N}$, $n \in \mathbb{N}_0$, $\sigma \in (0,1]$, $d \in \mathbb{N}_{\geq 2}$, and $B, p > 0$, which define the function class $\mathcal{E}^p_{r,\beta,d,B}$ of interest.

Let $\mathrm{M} \geq 1$ be given and let $\mathfrak{n} = \lceil \log_2 n \rceil$. Let $m \in \mathbb{N}$ be sufficiently large to satisfy $(2m+1) \geq (r+1)\frac{p(d-1+\beta)}{\beta}$.

- Let $(k,L) \in \mathbb{N}^2$ such that $m = k(L-1) \geq \frac{1}{2}\log_2\left(\frac{\log_2 n}{2}\right)$.

- Let $(s', r') \in \mathbb{N}^2$ such that $\tilde{\gamma}(d-1, n, B) \le (s'\mathrm{M})^{r'-1}$ where $\tilde{\gamma}$ is defined in (6.B.1).
- Let $(s'', r'', r''') \in \mathbb{N}^3$ such that $2^{2(m+1)}\mathrm{M}^{-1} \le 0.5(s''\mathrm{M})^{r''-1}$ and $(1 + \frac{(d-1)^{n+\beta}}{n!})B \le (((d-1)s'+1)\mathrm{M})^{r'''-1}$.
- Let $(s_h, r_h) \in \mathbb{N}^2$ such that $2^{\frac{\beta}{d-1+\beta}(2m+1)}\mathrm{M}^{-1} \le 0.5(s_h\mathrm{M})^{r_h}$.
- Let $(s_I, r_I) \in \mathbb{N}^2$ such that $2^{\frac{\beta}{p(d-1+\beta)}(2m+1)}\mathrm{M}^{-1} \le 0.5(s_I\mathrm{M})^{r_I-1}$.
- Let $(s'_g, r'_g) \in \mathbb{N}^2$ such that $\tilde{\gamma}(d, n, B) \le (s'_g\mathrm{M})^{r'_g-1}$ where $\tilde{\gamma}$ is defined in (6.B.1).
- Let $(s''_g, r''_g, r'''_g) \in \mathbb{N}^3$ such that $2^{2(m+1)}\mathrm{M}^{-1} \le 0.5(s''\mathrm{M})^{r''-1}$ and $(1 + \frac{d^{n+\beta}}{n!})B \le ((ds'+1)\mathrm{M})^{r'''_g-1}$.

Let us define an architecture of θ_K by

$$\vec{n}_{\theta_K} = (2^{rd} \cdot (\vec{n}_{\theta_0} + 2^{\oplus L_{\theta_0}}, (2s_h)^{\oplus r_h}), (2^{rd+1}(ds_I+1))^{\oplus r_I}), \tag{6.2}$$

where L_{θ_0} is the length of $\vec{n}_{\theta_0}$, and

$$\vec{n}_{\theta_0} = (\vec{n}_{\theta_1}, 2(N^{d-1}+d-1), (2N^{d-1}((d-1)s''+1))^{\oplus(r''+r''')}),$$
$$N^{d-1} = 2^{\frac{d-1}{d-1+\beta}(2m+1)},$$

where $P_{d,n} = \binom{d+n}{n}$ and

$$\vec{n}_{\theta_1} = \left(P_{d-1,n} \cdot [2^{\mathfrak{n}-1}, \dots, 2^0] \otimes \vec{n}_{\tilde{\times}}, (2s')^{\oplus(r'-1)}\right) + (2d)^{\oplus(\mathfrak{n}L+r'-1)},$$
$$\text{where} \quad \vec{n}_{\tilde{\times}} = (3(2^k+2))^{\oplus L}.$$

Also, define an architecture of θ_g by

$$\vec{n}_{\theta_g} = (\vec{n}_{\theta_2}, 2(N_g^d+d), (2N_g^d(ds''_g+1))^{\oplus(r''_g+r'''_g)}), \qquad N_g^d = 2^{\frac{d}{d+\beta}(2m+1)}, \tag{6.3}$$

where $\vec{n}_{\theta_2} = \left(P_{d,n} \cdot [2^{\mathfrak{n}-1}, \dots, 2^0] \otimes \vec{n}_{\tilde{\times}}, (2s'_g)^{\oplus(r'_g-1)}\right) + (2d)^{\oplus(\mathfrak{n}L+r'_g-1)}$.

Theorem 6.1. *For any $\mathbb{I}_K(\cdot)g(\cdot) \in \mathcal{E}^p_{r,\beta,d,B}$, there exists a ReLU NN, Φ such that*

$$\|\mathcal{R}[\Phi](x) - \mathbb{I}_K(x)g(x)\|_{L_p} \le \tilde{C} \cdot 2^{-\frac{\beta}{p(d-1+\beta)+\beta}(2m+1)},$$

with $\vec{n}_\Phi = (\vec{n}_{\theta_K} + \vec{n}_{\theta_g}, \vec{n}_{\tilde{\times}})$ and $|\Phi|_\infty \le \mathrm{M}$. Here $\tilde{C}$ is a constant that depends only on r, β, d, B, p. Also, θ_K and θ_g are ReLU NNs whose architectures are given by (6.2) and (6.3), respectively.

Proof. The proof can be found in Appendix 6.D. □

Remark 6.2. In the Appendix, we provided relevant approximation theorems with the controlled magnitude. The results indicate that one can approximate smooth functions with small weights, while discontinuous functions need delicate approaches to control the magnitude.

7 PINN with observational data

Assume we have a forward problem to solve, and have some observational data of the solution u, we can use the loss function by adding the data loss in the PINN loss (3.9).

$$\begin{aligned}\mathcal{L}(v;\boldsymbol{\lambda}) = \sum_{x\in\Omega_M} \lambda_x \Big(\mathcal{D}[v](x) - f(x)\Big)^2 + \sum_{x'\in\Gamma_{M'}} \lambda_{x'} \Big(\mathcal{B}[v](x) - g(x)\Big)^2 \\ + \sum_{x_{\text{obs}}} \lambda_{x_{\text{obs}}} |v - u(x_{\text{obs}})|^2 \end{aligned} \tag{7.1}$$

Then we solve the minimization problem $\min_\theta \mathcal{L}(u_{NN};\lambda)$.

If we have an inverse problem to solve, we may use a similar loss functional. For example, when we solve the data assimilation Problem 3.2 or the Cauchy problem of Poisson's equation 3.3, we can still use the above formulation. For the coefficient inverse Problem 3.4, we need to have two neural networks, one for the solution u, and the other for p. In this case, we denote the loss function at the continuous level by

$$\begin{aligned}\mathcal{L}(u_N, p_N;\lambda) = \left\|\partial_t^2 u_N - \nabla\cdot(p_N \nabla u_N)\right\|^2_{L^2(\Omega)} + \|u(x,0) - g(x)\|^2_{L^2} \\ + \|\partial_t u(x,0)\|^2_{L^2} + \|u - g\|^2_{L^2(S_T)} + \underbrace{\|\nabla u\cdot\mathbf{n} - h(x,t)\|^2_W}_{\text{data}}.\end{aligned}$$

Here the space W and the norm on it can be selected according to the Lipschitz stability in Klibanov and Yamamoto (2006). Suppose that Θ are the parameters for the network p_N, we then solve the following problem $\inf_{\theta,\Theta} \mathcal{L}(u_N, p_N;\lambda)$. While the observation data $u(x_{\text{obs}})$ are available, the choices of Ω_M, $\Gamma_{M'}$ may be adjusted from the formulation for forward problems.

In the PINNs framework, it is essentially the same to solve both forward and inverse problems. In both cases, we need stability of the problem to facilitate the setup of loss functions. The difference may lie in the complexity– we need two networks to solve inverse problems. Also, we may need some regularization to stabilize the formulation, as the stability may not be available theoretically.

PINNs have been applied to solve various inverse problems, such as elliptic and wave equations in Mishra and Molinaro (2022), supersonic problems in Jagtap et al. (2022a) and many others.

8 Deep operator networks

The PINN for PDEs usually solve one single problem at a time and can be time consuming. In particular training neural networks are performed for differential equations with fixed inputs, such as initial conditions, boundary conditions, forcing, and coefficients. If one input is changed, the training process has to be repeated. It is difficult to obtain outputs in real-time for systems that require various sets such in multiphysics.

Operator learning via neural networks has been developed to accommodate varying inputs and predict in real-time. In operator learning, a fixed-weighted/pretrained network approximates a continuous operator from the input(s) to the output(s), e.g. in the DeepONets (8.4). Once such networks are trained, we can predict/ evaluate at the desired input(s) in real-time.

To learn such networks, it often requires a huge amount of data to perform a least-squares regression for accurate approximations. For the learning of solution operators from PDEs, the data are usually generated by accurately solving the underlying PDEs with various inputs, which are usually modelled by stochastic processes with mild smoothness, such as in truncated Karhunen-Loeve expansions or a truncated Fourier expansion of stochastic processes.

8.1 Introduction

Let X and Y be Banach spaces. Consider a continuous operator $\mathcal{G} : X \to Y$ as follows. Let $\mathcal{G}(\cdot)$ be represented using a Schauder basis $\{e_k\}$ in the Banach space Y by $\mathcal{G}(u)(y) = \sum_{k=1}^{\infty} c_k(\mathcal{G}(u))e_k(y)$, where $c_k(\cdot)$ is a linear functional. We denote by $T_m u$ an approximation (encoder) or a parameterization of u in X and a truncation of the series

$$\mathcal{G}_{m,p}(T_m u)(y) = \sum_{k=1}^{p} c_k(\mathcal{G}(T_m u))e_k(y). \tag{8.1}$$

Assumption 8.1 (Modulus of continuity). Let $S \subset X$ and $w : [0, \infty) \to [0, \infty)$ be continuous at 0. There exists a constant $C > 0$ independent of $u, v \in S$.

$$\|\mathcal{G}(u) - \mathcal{G}(v)\|_Y \le C w(\|u - v\|_X), \quad \forall u, v \in S, \quad 0 < \alpha \le 1. \tag{8.2}$$

A particular useful choice of the function is $w(\cdot) = |\cdot|^\alpha$ for some $0 < \alpha \le 1$.

The error estimate of a network approximation can be derived as follows. We first split into three parts: approximation error from input, approximation error from output and approximation error from neural networks and then we can apply the modulus of continuity, the Schauder expansion and

$$\begin{aligned}\|\mathcal{G}(u) - \mathcal{G}_{\mathbb{N}}(T_m u)\|_Y \le \|\mathcal{G}(u) - \mathcal{G}(T_m u)\|_Y + \left\|\mathcal{G}(T_m u) - \mathcal{G}_{m,p}(T_m u)\right\| \\ + \left\|\mathcal{G}_{mp}(T_m u) - \mathcal{G}_{\mathbb{N}}(T_m u)\right\|_Y\end{aligned}$$

$$\begin{aligned}
&\le Cw(\|u - T_m u\|_X) + \sum_{k=p+1}^{\infty} |c_k(\mathcal{G}(T_m u))| \, \|e_k\|_Y \\
&\quad + \left\|\mathcal{G}_{m,p}(T_m u) - \mathcal{G}_{\mathbb{N}}(T_m u)\right\|_Y \qquad (8.3)\\
&\le Cw(\|u - T_m u\|_X) \\
&\quad + \sum_{k=p+1}^{\infty} (|c_k(\mathcal{G}(T_m u) - \mathcal{G}(u))| + |c_k(\mathcal{G}(u))|) \, \|e_k\|_Y \\
&\quad + \left\|\mathcal{G}_{m,p}(T_m u) - \mathcal{G}_{\mathbb{N}}(T_m u)\right\|_Y .
\end{aligned}$$

From this decomposition, we observe the factors of affecting the approximation error of an neural operator:

- How the input u is encoded/approximated. At least two typical approximations are used: interpolation, Deng et al. (2022) and Li et al. (2020); and truncated spectral expansion, Lanthaler et al. (2021) (from known basis) and Zhang et al. (2023b) (from Mercer's theorem).
- The choices of basis, which may significantly affect the efficiency of the approximation as in the spectral methods, empirical spectral methods (e.g., proper orthogonal decomposition (Lu et al., 2022)).
- The architecture of neural networks, e.g. DeepONets (Lu et al., 2019, 2021a) and extensions in Lu et al. (2022) and Zhang et al. (2023b), Fourier Neural operators (Li et al., 2020).

In practice, we also have errors from training/optimization and from *the mismatch of the inputs for training and evaluating*. Here we focus on the error from the mismatch. DeepONet takes function values at a set of fixed m points for the branch while the new input to evaluate at has function values at a different set of m' points. One way to accommodate this situation is to use a proper interpolation (or approximation) to derive a function (say $S_{m'}u$) so that one can evaluate the function at the set of m points that are required in the trained DeepONets. Note that this reformulation of the inputs does not require retraining DeepONets. The error for this accommodation can be bounded as follows.

$$\begin{aligned}
\|\mathcal{G}_{\mathbb{N}}(T_m S_{m'} u) - \mathcal{G}(u)\|_Y &\le \|\mathcal{G}_{\mathbb{N}}(T_m S_{m'} u) - \mathcal{G}(S_{m'} u)\|_Y + \|\mathcal{G}(S_{m'} u) - \mathcal{G}(u)\|_Y \\
&\le \|\mathcal{G}_{\mathbb{N}}(T_m S_{m'} u) - \mathcal{G}(S_{m'} u)\|_Y + Cw(\|S_{m'} u - u\|_X),
\end{aligned}$$

where we have applied the triangle inequality and the modulus continuity (8.2). The estimate of $w(\|S_{m'}u - u\|_X)$ is well studied in approximation theory and thus it is always assumed that the same set of m points is used for training and evaluating.

8.2 Vanilla DeepONets

In Chen and Chen (1995b); Lu et al. (2021a), the architecture of neural networks for operator learning is called Deep Operator Networks (DeepONets) in the following form

$$\mathcal{G}_{\mathbb{N}}(u) = \sum_{k=1}^{p} \underbrace{b^{\mathcal{N}}(T_m u; \Theta^{(k)})}_{\text{branch}} \underbrace{t^{\mathcal{N}}(y; \theta^{(k)})}_{\text{trunk}}, \tag{8.4}$$

where $T_m u = (u(x_1), u(x_2), \cdots, u(x_m))^\top$. Here the neural networks $f^{\mathcal{N}}$ and $g^{\mathcal{N}}$ can be any class of functions that satisfy the classical universal approximation theorem of continuous functions on compact sets.

Theorem 8.2 (Universal approximation theorem, Chen and Chen, 1995b; Lu et al., 2021a). *Let $K_1 \subset \mathbb{R}^{d_1}$ be a compact set. Let V be a compact set in $C(K_1)$, $K_2 \subset \mathbb{R}^d$ be a compact set and $Y = C(K_2)$. Assume that $\mathcal{G}: V \to Y$ is continuous. Then for any $\epsilon > 0$, there are positive integers p and m, neural networks $t^{\mathcal{N}}(\cdot; \theta^{(k)})$ and $b^{\mathcal{N}}(\cdot; \Theta^{(k)})$, $k = 1, \ldots, p$, $x_j \in K_1$, $j = 1, \ldots, m$, such that*

$$\sup_{u \in V} \sup_{y \in K_2} |\mathcal{G}(u)(y) - \mathcal{G}_{\mathbb{N}}(T_m u)(y)| < \epsilon, \tag{8.5}$$

where $T_m u$ is an interpolation of u at $x_1, \ldots, x_m$ using $(u(x_1), u(x_2), \cdots, u(x_m))^\top$.

The universal approximation theorem can be proved by emulating a truncated Fourier expansion as in Chen and Chen (1995b). However, the convergence order cannot be established therein as the branch networks is high-dimensional as a function of $(u(x_1), u(x_2), \cdots, u(x_m))$. Also, according to the proofs in Chen and Chen (1993, 1995a,b); Chen (1998), $C(K_1)$ can be replaced by $L^p(K_1)$, $p \geq 1$ and $C(K_2)$ can be replaced by $L^q(K_2)$, $q \geq 1$. But $T_m u$ should be replaced by averaged values $T_m u = (T_m u_h(x_1), \ldots, T_m u_h(x_m))^\top$, where $T_m u_h(x_i) = \int_{B(x_i,h)\cap K} u(t)\, dt / \mu(B(x_i, h) \cap K)$ while μ is the Lebesgue measure and $B(x_i, h)$ is the ball centered at x_i with radius h. More generally, these spaces may be replaced by Banach spaces with a Schauder basis.

8.3 Approximation rates for general Hölder operators

As indicated in (8.3), for a given operator, the approximation error of DeepONets depends on encoder/approximation of the input $T_m u$ and the choices of basis and architecture of branch and trunk networks. To establish approximation rates, we may use the current state-of-the-art where we emulate DeepONets by certain numerical methods.[3] Thus it is important to identify numerical methods which the architecture is analogy to and how the numerical methods behave.

Here we use some conclusions from Deng et al. (2022) on the approximation error of DeepONets using feedforward neural networks for Hölder continuous operators. The error is obtained by emulating truncated Fourier expansions in

[3] The emulation provides error estimates and thus insights on how one can obtain and improve architectures of networks for operator learning.

the L^q space. Let $\mathcal{N}(\cdot;\theta)$ be an feedforward neural network with parameters θ. We use $|\theta|$ to denote the size of is the total number of nonzero parameters; $N_{\mathcal{N}}$ for the width of $\mathcal{N}$ is the number of neurons in each layer; $L_{\mathcal{N}}$ for the depth or the number of layers.

Assume that

$$\|T_m u - u\|_X \le C m^{-r} \|u\|_V, \forall u \in V. \quad r > 0. \tag{8.6}$$

Theorem 8.3 (Error estimates of DeepONets for Hölder continuous operators, Deng et al., 2022). *Assume that the conditions in Theorem 8.2 and (8.6) hold. Assume that T_m is Lipschitz continuous and $\mathcal{G}$ is Hölder continuous with exponent α. Let $Y = L^q(K_2)$, where $1 \le q \le \infty$, $K_2 \subset \mathbb{R}^d$ is compact. Then we have*

$$\|\mathcal{G}(u) - \mathcal{G}_{\mathbb{N}}(T_m u)\|_Y \le \left\|\mathcal{G}(u) - \mathcal{G}_{m,p}(T_m u)\right\|_Y + \left\|\mathcal{G}_{m,p}(T_m u) - \mathcal{G}_{\mathbb{N}}(T_m u)\right\|_Y, \tag{8.7}$$

where $\mathcal{G}_{m,p}(T_m u)$ is defined in (8.4) and

$$\left\|\mathcal{G}(u) - \mathcal{G}_{m,p}(T_m u)\right\|_Y \le C m^{-r\alpha} + C\omega_2(\mathcal{G}(T_m u), p^{-1/d})_q,$$
$$\left\|\mathcal{G}_{m,p}(T_m u) - \mathcal{G}_{\mathbb{N}}(T_m u)\right\|_Y \le C\left(p\sqrt{m} N_{b\mathcal{N}}^{-2\alpha/m} L_{b\mathcal{N}}^{-2\alpha/m} + p\exp(-|\theta|^{\frac{1}{1+d}})\right),$$

where r is from (8.6). The positive constant C is independent of m, p and $N_{b\mathcal{N}}$ is the number of neurons in each layer of the branch network, $|\theta|$ is the size (number of nonzero parameters) of each trunk network $t^{\mathcal{N}}$, and $L_{b\mathcal{N}}$ is the numbers of layers of the branch network.

Let us denote the order/magnitude of '$\cdot$' by '$\sim \cdot$'. According to Theorem 8.3, in order to make the total error $\|\mathcal{G}(u) - \mathcal{G}_{\mathbb{N}}(T_m u)\|_Y < \varepsilon$, we need to set $m \sim \varepsilon^{-\frac{d}{\alpha}}$, $p \sim \varepsilon^{-\frac{d}{2}}$, $N_{g\mathcal{N}} L_{g\mathcal{N}} \sim \varepsilon^{-\frac{d}{\varepsilon}}$ and $|\theta| \sim \left(\frac{d+2}{2}\ln(\frac{1}{\varepsilon})\right)^{d+1}$. The low approximation rate and the high complexity is caused by the high dimensionality of the function (of $T_m u$) and Hölder continuous operators. If an operator is smoother, faster convergence can be obtained, e.g. exponential convergence in Lanthaler et al. (2021) and Marcati and Schwab (2023). In addition to the smoothness of the operator $\mathcal{G}$, the structure of the operators or approximate operator $\mathcal{G}_{m,p}$ is also important for the convergence rates of DeepONet. For example, for solution operators from partial differential equations we may have better convergence as we can obtain $\mathcal{G}_{m.p}$ using well-studied and efficient numerical methods and then neural networks can emulate these methods. In Deng et al. (2022), it is shown that there is no curse of dimensionality (exponential complexity in m) for several solution operators from PDEs. We will present such examples from Deng et al. (2022) in Section 8.4.

Below, we discuss some theoretical considerations when obtaining error estimates. *I. Physical domains of inputs and outputs K_1 and K_2.* We usually assume that the domains of K_1 and K_2 are cubes or balls, where approximations with convergence rates are well studied. If they are not cubes or balls, we use the Tietze-Urysohn-Brouwer extension theorem to extend the operator such that domains of input and output are cubes or balls (denoted by D). We denote the resulting operator (with also extension of V and image) by $\mathcal{G}(u)$ if no confusion arises. *II. Choices of function spaces for* $\mathcal{G}_{\mathbb{N}}$. Let's suppose that $\mathcal{G}_{\mathbb{N}}$ maps functions in V_m to functions in Y_p. Once we identify choices of the spaces V, V_m, Y, and Y_p, we are ready to calculate the approximation rate. In addition to the truncated Fourier expansion for both input and output, we list in Table 1 some typical choices of these spaces. Other choices are possible, e.g., V_m can be chosen as the set of networks with m parameters, which can well approximate functions in V while V_m is not a linear subspace of V. It is important to choose the best possible choices to parameterize the input and output as they determine the dimensionality of the input and output of the operator $\mathcal{G}_{m,p}(T_m u)$. The choices are highly problem dependent. In this work, we only consider two choices of piecewise polynomials and truncated Fourier space. We refer the interested readers for more discussion in Kovachki et al. (2021b), Deng et al. (2022), Lanthaler et al. (2021), and Marcati and Schwab (2023).

TABLE 1 List of some parameterizations of the input and output. Here RKHS refers to reproducing kernel Hilbert space.

$X(V)$ or Y	V_m or Y_p	$T_m u$	Ref.
L^p	piecewise polynomials	function values	Deng et al. (2022)
L^q	truncated Fourier space	Fourier coefficients	Deng et al. (2022) Lanthaler et al. (2021)
RKHS	truncated RKHS (first N term of the basis)	expansion coefficients	Lanthaler et al. (2021) Zhang et al. (2023b)
analytic functions	truncated orthogonal polynomials	expansion coefficients	Marcati and Schwab (2023)

8.4 Error estimates for solution operators from PDEs

We now consider error estimates of DeepONets for solution operators from linear and nonlinear advection-diffusion-reaction equations.

Example 8.4. Consider the following 1D advection-diffusion equation with Dirichlet boundary condition:

$$-u_{xx} + a(x)u_x = f(x), \quad x \in (0, L), \, 0 < L < \infty, \qquad u(0) = u(L) = 0, \quad (8.8)$$

where $a(x), f(x) \in L^\infty(0, L)$. The solution operator is defined from the coefficient a to the solution u by

$$u = \mathcal{G}^f(a). \quad (8.9)$$

Let $\{x_i\}_{i=1}^m$ be a partition of $(0, L)$. Define $\mathbf{a}_m = (a_1, \cdots, a_m)^\top$, where $a_i = a(x_i)$, $x_i \in (0, L)$, $i = 1, 2, \ldots, m$. From the solution (8.10), we can define a solution operator by

Theorem 8.5 (Error estimates of DeepONet for the solution operator (8.9), Deng et al., 2022). *For any given $f \in L^\infty$, let $\mathcal{G}^f(a)$ be the solution operator (8.9). Let $S = \{a(x) : a \in L^\infty(0, L),\ \partial_x a \in L^\infty(0, L)\}$. Then, there exist ReLU branch networks $b^N(\mathbf{a}_m; \Theta^{(i)})$ of size $|\Theta^{(i)}| = m^4 \ln(m)$ for $i = 1, \cdots, p$, and ReLU trunk networks $t^N(x; \theta^{(k)})$ of width $N_{t_N} = 3$ and depth $L_{t_N} = 1$, $k = 1, \cdots, p$, such that*

$$\sup_{a \in S} \left\| \mathcal{G}^f(a) - \mathcal{G}^f_{\mathbb{N}}(\mathbf{a}_m) \right\|_{L^\infty} \le C\left(p^{-1} + m^{-1} + \left|\Theta^{(i)}\right|^{-\frac{1}{4}+\epsilon}\right),$$

where $\mathcal{G}^f_{\mathbb{N}}(\mathbf{a}_m)$ is of the form in (8.4), $\epsilon > 0$ is arbitrarily small and $C > 0$ is independent of m, p, $|\Theta^{(i)}|$ and $a(x)$.

The proof can be done by emulating the analytic solution as follows. The solution can be written as

$$u(x) = -\left(\mathcal{A}_- \circ \mathcal{A}_+\right)(f)(x) + \frac{\mathcal{A}_-(\mathbb{1})(x)}{\mathcal{A}_-(\mathbb{1})(L)}\left(\mathcal{A}_- \circ \mathcal{A}_+\right)(f)(L), \qquad (8.10)$$

where $\mathbb{1}(x) = 1$ for $x \in [0, L]$ and $\mathcal{A}_+(g)(x) := \int_0^x A(y)g(y)dy$, $\mathcal{A}_-(g)(x) := \int_0^x A^{-1}(y)g(y)dy$ and $A(x) = \exp(-\int_0^x a(y)\,dy)$. We can utilize the analytical formulation (8.10) to show that $\left\|\mathcal{G}^f(a) - \mathcal{G}^f(I_m^0 a)\right\|_{L^\infty} \le C \left\|a - I_m^0 a\right\|_{L^\infty}$, where $\mathcal{I}_m^0 v(x) = v(x_{i-1})$ (piecewise constant interpolation) on $[x_{i-1}, x_i)$. Define an approximation of $\mathcal{G}^f(a)$ as

$$\mathcal{G}^f_m(\mathbf{a}_m)(x) := -\left(\mathcal{A}_-^N \circ \mathcal{A}_+^N\right)(f)(x) + \frac{\mathcal{A}_-^N(\mathbb{1})(x)}{\mathcal{A}_-^N(\mathbb{1})(L)}\left(\mathcal{A}_-^N \circ \mathcal{A}_+^N\right)(f)(L), \quad (8.11)$$

where $\mathcal{A}_+^N(g)(x) := \int_0^x \mathcal{I}_m^0(Ag)(y)dy$, $\mathcal{A}_-^N(g)(x) := \int_0^x \mathcal{I}_m^0(A^{-1}g)(y)dy$. The approximation error introduced in this step can be calculated and the order is m^{-1}, which is the convergence order of piecewise linear interpolation. Then $\mathcal{G}(a)$ can be approximated by $\sum_{i=1}^p \mathcal{G}^f_m(\mathbf{a_m})(y_i)L_j^p(y)$ where $L_j^p(y)$ is the piecewise interpolation polynomial on a partition of $(0, L)$ and the resulting error is of order $p^{-1} + m^{-1}$. Approximating $\mathcal{G}^f_m(\mathbf{a_m})(y_i)$ by ReLU feedforward neural network $b^N(\mathbf{a_m}; \Theta^{(i)})$ and $L_j^p(y)$ by $t^N(y; \theta^{(k)})$ we then obtain a DeepONet in the form (8.4). The approximation error by neural networks can be bounded by observing that $\mathcal{G}^f_m(\mathbf{a_m})(y_i)$ is a rational polynomial in $\mathbf{V}_m = (v_1, v_2, \ldots, v_m)$ where $v_i = \exp(a_i(x_i - x_{i-1})$ and $L_j^p(y)$ can be rewritten by a linear combination of two-layer ReLU neural networks and approximation error estimates in Telgarsky (2017).

Example 8.6. Consider the Burgers' equation (8.18). Define the solution operator by $u = \mathcal{G}(u_0)$ from the initial condition to the solution.

For $l \in \mathbb{Z}$, $x_j^l = x_j + 2\pi l$, $j = 0, 1, \cdots, m$, where $\Pi_m = \{-\pi = x_0 < x_1 < \cdots < x_m = \pi\}$ is a partition of $[-\pi, \pi]$. Let $M_0, M_1 > 0$. Define

$$\mathcal{S} = \mathcal{S}(M_0, M_1) := \{v|_{[-\pi,\pi)} : \left\|v|_{[-\pi,\pi)}\right\|_{L^\infty} \le M_0, \|\partial_x v|_{[-\pi,\pi)}\|_{L^\infty} \le M_1,$$
$$\bar{v} := \int_{-\pi}^{\pi} v(s)\, ds = 0\}. \tag{8.12}$$

Let $L_j^m(x)$ be the piecewise linear nodal basis over the partition Π_m, $h_j = x_j - x_{j-1}$, and $h = \max_{1 \le j \le m} h_j$.

Theorem 8.7 (Error estimate of DeepONet for 1D Burgers equation with periodic boundary). *Let $\mathcal{G}(u_0)$ be the solution operator of the Burgers equation* (8.18). *Then, there exist ReLU branch networks $g^{\mathcal{N}}(T_m u_{0,m}; \Theta^{(i)})$ of size $|\Theta^{(i)}| = \mathcal{O}(m^2 \ln(m))$ for $i = 1, \cdots, p$, and ReLU trunk networks $f^{\mathcal{N}}(x; \theta^{(k)})$ of size $\mathcal{O}(1)$, $k = 1, \cdots, p$, such that for any $t > 0$,*

$$\sup_{u_0 \in \mathcal{S}} \|\mathcal{G}(u_0)(\cdot, t) - \mathcal{G}_{\mathbb{N}}(T_m u_0)(\cdot, t)\|_{L^\infty([-\pi,\pi))}$$
$$\le C\left(p^{-1} + m^{-1} + \left|\Theta^{(i)}\right|^{-\frac{1}{2}+\epsilon}\right),$$

where $\mathcal{G}_{\mathbb{N}}(T_m u_0)$ is of the form (8.4), *$\epsilon > 0$ is arbitrarily small and $C > 0$ is independent of m, p, $|\Theta^{(i)}|$ and the initial condition u_0.*

The proof of the theorem can be established by three steps. *First*, it can be shown that by (8.19), the solution operator $\mathcal{G} : X \to Y$ is Lipschitz continuous, where $X = Y = L^\infty([-\pi, \pi))$. *Second*, let

$$\mathcal{G}_{m,p}(T_m u_0)(x, t) = \sum_{k=1}^{p} \mathcal{G}_m(T_m u_0)(x_k, t) L_k^p(x), \quad x_k \in \Pi_p,$$

where $\mathcal{G}_m(T_m u_0)$ is defined by applying the piecewise constant and linear interpolation in (8.19)

$$\mathcal{G}_m(T_m u_0)(\mathbf{x}) = \frac{-2\kappa \int_{\mathbb{R}} \partial_x \mathcal{K}(x, y, t)(\mathcal{I}_m^1 v_0)(y) dy}{\int_{\mathbb{R}} \mathcal{K}(x, y, t)(\mathcal{I}_m^0 v_0)(y) dy}$$
$$= \frac{v_0^0 c_0^1(\mathbf{x}) + v_1^0 c_1^1(\mathbf{x}) + \cdots + v_{m-1}^0 c_{m-1}^1(\mathbf{x})}{v_0^0 c_0^2(\mathbf{x}) + v_1^0 c_1^2(\mathbf{x}) + \cdots + v_{m-1}^0 c_{m-1}^2(\mathbf{x})}, \ \mathbf{x} = (x, t), \tag{8.13}$$

where $\mathcal{I}_m^0 f$ and $\mathcal{I}_m^1 f$ be the piecewise constant interpolation and piecewise linear interpolation of f over Π_m, respectively, and $v_j^0 = v_0(x_j) = \prod_{i=0}^{j} V_i$, ($V_i = \exp(-\frac{u_{0,j}+u_{0,j-1}}{4\kappa} h_i)$) and $c_j^i(\mathbf{x})$, $i = 1, 2$, $j = 0, \cdots, m-1$ are functions in $\mathbf{x}$. In this step, the approximation error can be bounded by (Theorem 3.3, Deng et al., 2022), for any $\mathbf{x} = (x, t) \in (-\pi, \pi) \times (0, +\infty)$,

$$\begin{aligned}\sup_{u_0 \in \mathcal{S}} \left| \mathcal{G}(u_0)(\mathbf{x}) - \mathcal{G}_{m,p}(T_m u_0; \mathbf{x}) \right| &\leq C p^{-1} + \sup_{u_0 \in \mathcal{S}} \left| \mathcal{G}(u_0)(\mathbf{x}) - \mathcal{G}_m(T_m u_0; \mathbf{x}) \right| \\ &\leq C p^{-1} + 2\left(\frac{M_0^2}{\kappa} + M_1\right) h. \end{aligned} \tag{8.14}$$

Third, we can view $\mathcal{G}_m(T_m u_0)(x)$ as a rational function in $(V_0, V_1, \cdots, V_{m-1})^\top$ and thus can be well approximated by a ReLu neural network as in Telgarsky (2017). Specifically, there exists a ReLU network $g^{\mathcal{N}}(T_m u_{0,m}; \Theta)$ of size $|\Theta| = O(m^2 \ln(m))$ and a constant $C = C(\kappa, M_0, M_1) > 0$, such that for any $\mathbf{x} \in [-\pi, \pi) \times (0, \infty)$, there exists a set of parameters $\Theta_{\mathbf{x}}$ such that

$$\sup_{u_0 \in \mathcal{S}} \left| \mathcal{G}_m(T_m u_0; \mathbf{x}) - g^{\mathcal{N}}(T_m u_0; \Theta_{\mathbf{x}}) \right| \leq C h.$$

Rewriting the basis L_k^p by a ReLu neural network, we then obtain the desired conclusion.

Extensions to 1D Burgers equation with Dirichlet boundary condition and/or forcing terms and 2D Burgers equation have been discussed in Deng et al. (2022).

Example 8.8 (2D diffusion-reaction equation). Consider the following 2D diffusion-reaction equation:

$$-\Delta u + a(x, y) u = f, \quad \text{in } \Omega \subset \mathbb{R}^2, \quad \mathcal{B} u = 0, \quad \text{on } \partial\Omega, \tag{8.15}$$

where Ω is a rectangular domain and $\mathcal{B}$ can be the Dirichlet, Neumann or Robin boundary operator.

Let $a(x, y) \in \Sigma$, where

$$\Sigma = \{a(x, y) \in W^{r-2,2}(\Omega) \cap L^\infty(\Omega) : 0 \leq a(x, y) \leq C_0\}, \quad C_0 > 0.$$

Here $1 < r \leq 3$ and $W^{s,2}$ is the Sobolev space with square-integrable s-th order weak derivatives. The theorem below states the rate of DeepONets approximating the solution operator of 2D advection-reaction-diffusion equation (8.15).

Theorem 8.9 (Approximation rate of DeepONet for 2D diffusion-reaction equations). *Assume $f \in W^{r-2,2}(\Omega)$, $1 < r \leq 3$. Let $\mathcal{G}^f(a)$ be the solution operator. There exist a branch network (blessed representation) $b^{\mathcal{N}}(\mathbf{a}_m; \Theta) \in \mathbb{R}^{p \times 1}$ of size $O(m^3 \ln(\varepsilon^{-1}))$ and ReLU trunk networks $t^{\mathcal{N}}(x; \theta^{(k)}) \in \mathbb{R}^{1 \times 1}$ of size $|\theta^{(k)}| = O(1)$, $k = 1, \cdots, p$ ($p = m$), such that*

$$\sup_{a \in \Sigma} \left\| \mathcal{G}^f(a) - \mathcal{G}^f_{\mathbb{N}}(\mathbf{a}_m) \right\|_{L^\infty} \leq C\left((\log m)^{\frac{1}{2}} m^{-\frac{r-1}{2}} + \left| N_{b^{\mathcal{N}}} L_{b^{\mathcal{N}}} \right|^{-\frac{1}{3}+\epsilon} \right),$$

where $\epsilon > 0$ is arbitrarily small and $C > 0$ is independent of m, $N_{b^{\mathcal{N}}}$, $L_{b^{\mathcal{N}}}$ and $a(x, y)$. Here $\mathcal{G}^f_{\mathbb{N}}(\mathbf{a}_m) = (b^{\mathcal{N}}(\mathbf{a}_m; \Theta))^\top \vec{t}^{\mathcal{N}}$ and $\vec{t}^{\mathcal{N}} = (t^{\mathcal{N}}(x; \theta^{(1)}), \ldots, t^{\mathcal{N}}(x; \theta^{(p)}))$.

As we don't have analytical solutions, we emulate the finite difference method to establish the approximation rate of a DeepONet. The proof is established in three steps. First, we approximate the solution by the central finite difference scheme and obtain the approximation error $(\log(m))^{\frac{1}{2}} m^{-\frac{r-1}{2}}$ when m grid points are used. Second, we rewrite the solver for the linear system resulting from the central finite difference scheme by applying the Sherman-Morrison's formula. Third, we emulate the Sherman-Morrison's formula by a blessed representation (a tree structure) (Mhaskar and Poggio, 2016) for branch networks $b^{\mathcal{N}}$. Then we can obtain the approximation error estimates by the estimates of the blessed representation in Mhaskar and Poggio (2016). Details of the proof can be bounded in Section 4 of Deng et al. (2022).

8.5 Training DeepONets

To train DeepONets, we need the following:

- Sufficient amount of data, which represents diverse inputs and output;
- A suitable architecture for the underlying operator;
- A proper training method such as stochastic gradient descent methods.

High quality data generation is perhaps the first essential step for the success of DeepONets. In real applications, data may be captured by placing affordable and feasible number of sensors. When working with solution operators, we need sufficient amount of data from efficient solvers of the underlying problems with various representative inputs. The inputs are usually modelled with Gaussian random fields with covariance functions such as Gaussian $\exp(-|x - y|^2 / l^2)$ in Lu et al. (2021a, 2022), Matern kernel (Li et al., 2020). It is crucial to tune the parameters in these covariance functions to have representative inputs. Over a bounded domain D, we can obtain via Mercer's theorem that the Gaussian process can be written as (known as Karhunen-Loeve expansion) $\sum_{k=1}^{\infty} \sqrt{\lambda_k} e_k(x) \xi_k$, where (λ_k, e_k) is an eigenpair of the covariance function, ξ_k's are independent and identically distributed standard Gaussian random variables. Also, $\{e_k\}_{k=1}^{\infty}$ forms an orthonormal basis in $L^2(D)$ over a bounded domain D and thus has the capacity to represent a large class of functions in a Hilbert space, which is called the reproduced kernel Hilbert space.

The architectures for DeepONets may be adjusted for different operators or even for the same operator for a different level of accuracy. We will not discuss this aspect further.

In the DeepONets, we often use the l_2 regression. Suppose that we are given the data pairs $(u^{(j)}, v^{(j)})_{j=1}^{J}$ where $v^{(j)} \approx \mathcal{G}(u^{(j)})$ (such as those obtained from certain numerical methods or from noisy observations). Here the functions $v^{(j)}$

are given at certain points y_i's.

$$\sum_{j=1}^{J}\sum_{i=1}^{I}\left\|\mathcal{G}_{\mathbb{N}}(T_m u^{(j)})(y_i) - v^{(j)}(y_i))\right\|^2. \tag{8.16}$$

Here the numbers I and J and the size of neural networks should be chosen carefully to have an efficient calculation. These numbers can be estimated according to error estimates, e.g. those in last subsections.

If we know the physics laws that the data obey, e.g. in the case of solution operators from $\mathcal{D}v = f$ with certain boundary or initial conditions, we may use the physics-informed DeepONets, e.g., in Wang et al. (2021) and Goswami et al. (2023) to improve the efficiency. The formulation may read as follows.

$$\sum_{j=1}^{J}\Big(\sum_{i=1}^{I}\left\|\mathcal{G}_{\mathbb{N}}(T_m u^{(j)})(y_i) - v^{j}(y_i))\right\|^2 + \sum_{l=1}^{L}\left|\mathcal{L}\mathcal{G}_{\mathbb{N}}(T_m u^{(j)}) - f\right|^2(z_l)\Big) \tag{8.17}$$

with possibly extra terms on boundary and initial conditions. Here the points $\{z_l\}$ can different from the points $\{y_i\}$. The accuracy of DeepONets may be enhanced by adding a norm of the gradients, e.g. Luo et al. (2023a) and in Section 8.7.

The training of the DeepONets can be performed with stochastic gradient descent methods. However, due to the structure of DeepONets, the training of the DeepONets can be split into two steps as in DeepONets with POD (Lu et al., 2022) or SVD (Venturi and Casey, 2023): one can first perform singular value decomposition and use the eigenvectors scaled by the positive eigenvalues as the basis and emulate this basis to obtain trunk networks and then one trains branch networks. In Lee and Shin (2023), a two-step training method is proposed based on the Gram-Schmidt orthonormalization. The two-step method of Lee and Shin (2023) is based on the matrix expression of the loss. Since the DeepONet can be written as

$$\mathcal{G}_{\mathbb{N}}(u)(y) = T(y)B(u) \quad \text{where} \quad T(y;\theta) = [t^{N}(y;\theta^{(1)}), \ldots, t^{N}(y;\theta^{(p)})],$$

$$B(u;\Theta) = \begin{bmatrix} b^{N}(T_m u;\Theta^{(1)}) \\ \vdots \\ b^{N}(T_m u;\Theta^{(p)}) \end{bmatrix},$$

where θ and Θ are the collections of all the parameters of the trunk and branch networks, respectively, it can be checked that the loss of (8.16) can be expressed as

$$\mathcal{L}(\theta,\Theta) = \|T(\theta)B(\Theta) - V\|_F^2,$$

where T is the matrix of size $I \times p$ whose i-th row is $T(y_i;\theta)$, B is the matrix of size $p \times J$ whose j-th column is $B(u^{(j)};\Theta)$, and V is the matrix of size $I \times J$

whose j-th column is $[v^{(j)}(y_1), \ldots, v^{(j)}(y_I)]^\top$. In the first step, the method solves

$$\min_{\theta, A} \|T(\theta)A - V\|_F^2,$$

where A is a trainable matrix of size $p \times J$ and let (θ^*, A^*) be an optimal solution. Then, the Gram-Schmidt orthonormalization is applied to the trunk network, which can be done by the QR-factorization (or equivalently SVD), which gives $T(\theta^*) = Q^* R^*$. The second step then trains the branch network to fit $R^* A^*$, i.e.,

$$\Theta^* = \operatorname*{argmin}_{\Theta} \|B(\Theta) - R^* A^*\|.$$

Accordingly, the trunk network is given by $T(y; \theta^*)(R^*)^{-1}$ and the branch network is $B(u; \Theta^*)$, which gives the DeepONet trained by the two-step of Lee and Shin (2023) is $\mathcal{G}_{\mathbb{N}}(u)(y) = T(y; \theta^*)(R^*)^{-1} B(u; \Theta^*)$.

Remark 8.10. It is worth mentioning some differences between the above two-step training methods and the direct use of either SVD (Lu et al., 2022) or POD (Venturi and Casey, 2023) for the trunk networks. A major difference lies in the linear learning versus the nonlinear learning. The direct use of SVD or POD for the trunk network means that one should expect only a linear relation between data and the basis functions on a fixed grid. Yet, the use of the trunk network in the learning process creates a nontrivial nonlinear relation between data and the resulting basis functions. Furthermore, one can evaluate the basis (the trunk network) on any points aside from the given grid. Admittedly, if one is interested in problems where the standard SVD/POD approaches perform well, one may skip the first step of the training and use the SVD or POD basis instead of training the trunk network. However, if one is concerned with problems where the standard SVD/POD approaches suffer, e.g. advection dominated problems, the nonlinear learning approach provides a viable option. In particular, Theorem 3.5 of Lee and Shin (2023) shows that there exists a trunk network architecture having the smallest width of 4 that approximates any SVD basis. This indicates that one advantage expected from nonlinear learning may be an efficient dimension reduction capability. A similar discussion can be found in the context of the reduced order modeling (ROM) – linear manifold ROM versus nonlinear manifold ROM (Kim et al., 2022).

8.6 Extending DeepONets

In Chen and Chen (1995b), $t^{\mathcal{N}}$ and $b^{\mathcal{N}}$ are two-layer (shallow neural networks) while multilayer feedforward neural networks are used in Lu et al. (2021a). Various extensions in architectures are made such as using convolutions neural networks for branch neural networks in order to accelerate the computation as the input of the branch networks is very high-dimensional, adding Fourier features trunk networks or using cosine and sine functions as the inner layer of the trunk network, e.g. in Lu et al. (2022).

DeepONets with proper orthogonal decomposition (POD) has been developed in Lu et al. (2022), where the trunk networks are replaced by the empirical orthogonal basis from performing POD on the data. A similar idea is implemented in Venturi and Casey (2023), where singular value decomposition is applied.

Convolutional neural networks can be employed in branch networks to reduce the computational cost in Lu et al. (2022). Similarly, Fourier neural operators (Li et al., 2020; Kovachki et al., 2021a) can be applied as branch networks.

As mentioned in Section 8.1, the vanilla DeepONets depends on a fixed discretization and thus can be extended to take data from discretization of different mesh size in Zhang et al. (2023b) by adding one more layer as a function of y in branch networks. In Franco et al. (2023), the branch networks are designed to accommodate mesh-based data.

Architectures of DeepONets can also be established via emulating efficient numerical methods for the underlying operators. For example, the solution operators arising from partial differential equations. For example, in Deng et al. (2022), the above architecture of DeepONets for solution operators of 1D and 2D diffusion-reaction equations (on an interval and a square) can be established by emulating finite difference methods and a numerical solver for the resulting linear system. The approximation error estimates can be obtained accordingly.

8.7 Benchmark test: Burgers' equation

Consider the 1-D Burgers equation with periodic boundary condition

$$\begin{cases} u_t + uu_x = \kappa u_{xx}, \quad (x,t) \in \mathbb{R} \times (0,\infty), \quad \kappa > 0, \\ u(x-\pi,t) = u(x+\pi,t), \quad u(x,0) = u_0(x). \end{cases} \tag{8.18}$$

Then, by the Cole-Hopf transformation, the solution to (8.18) can be written as $u = \frac{-2\kappa v_x}{v}$, where $v_t = \kappa v_{xx}$, $v(x,0) = v_0(x) = \exp\left(-\frac{1}{2\kappa}\int_{-\pi}^{x} u_0(s)ds\right)$. With the heat kernel $\mathcal{K}(x,y,t) = \frac{1}{\sqrt{4\pi\kappa t}}\exp\left(-\frac{(x-y)^2}{4\kappa t}\right)$, we obtain

$$u(x,t) = -2\kappa \frac{\int_{\mathbb{R}} \partial_x \mathcal{K}(x,y,t) v_0(y)dy}{\int_{\mathbb{R}} \mathcal{K}(x,y,t) v_0(y)dy}. \tag{8.19}$$

Here we assume that the initial condition $u_0(x)$ has zero mean in a period $\bar{u}_0 := \int_{-\pi}^{\pi} u_0(s)ds = 0$ and thus $v_0(x)$ in (8.19) is 2π-periodic.

Remark 8.11. The condition we need in the Cole-Hopf transformation is $\int_{\mathbb{R}} \exp(-\epsilon x^2)\,|v_0(x)|\,dx < \infty$, which can be satisfied when $u_0(x)$ is piecewise linear as assumed. Here $\epsilon > 0$. If the initial condition $u_0(x)$ has the average

$\bar{u}_0 \neq 0$, we write the solution $u(x,t) := v(x - \bar{u}_0 t, t) + \bar{u}_0$, where v satisfies the Burgers equation (8.18) with the initial condition of zero average $u_0 - \bar{u}_0$.

In the following experiments, we test two different formulations for DeepONets for the solution operator $\mathcal{G}$ defined by $u = \mathcal{G}(u_0)$ over the interval $[0, 1]$ (by scaling the domain from $[0, 2\pi]$ to $[0, 1]$, e.g., by $x' = x/(2\pi)$, denoted also by x), when $\kappa = 0.001$.

Data generation. We use a Gaussian random field (GRF) to generate various initial conditions. Specifically, we let $u_0(x) = f(\sin^2(ax))$, where $f(x) \sim \mathcal{GP}(0, k_l(x_1, x_2))$, a is a constant s.t. u_0 satisfies the desired periodic condition. The covariance kernel $k_l(x_1, x_2)$ is taken to be the Gaussian kernel with a length scale parameter $l > 0$, i.e., $k_l(x_1, x_2) = \exp\left(-\|x_1 - x_2\|^2 / 2l^2\right)$. We use a high resolution Fourier spectral method to generate data u.

Architecture and Hyperparameters of DeepONets. We apply the plain DeepONet of the form (8.4) with the following setting:

- m (data length, #sensors) $= 641$; #u_{data} train $= 30{,}000$; #u_{data} test $= 3000$,
- (ADAM optimizer) learning rate $= 6 \times 10^{-4}$ with exponential decay; epochs $= 500{,}000$.

For the branch network, we use ReLU neural network with layers $[m] + [200] \times 10$; and for the trunk network, we use the swish neural network with layers $[1] + [200] \times 10$;

We use the following two Loss functions: the standard MSE:

$$\begin{aligned}\mathcal{L}_1 &= \frac{1}{N_u \times N_y} \|T_{\text{data}} - \mathcal{G}_T(u_{\text{data}})\|_{l^2}^2 \\ &= \frac{1}{N_u \times N_y} \sum_{i=1}^{N_u} \sum_{j=1}^{N_y} \left| T_j^{(i)} - \mathcal{G}_T\left(\mathbf{u}^{(i)}\right)(y_j) \right|^2 \end{aligned} \tag{8.20}$$

and MSE with regularization by first-order derivatives:

$$\mathcal{L}_2 = \frac{1}{N_u \times N_y} \|T_{\text{data}} - \mathcal{G}_T(u_{\text{data}})\|_{l^2}^2 + \frac{\lambda_g}{N_u \times N_y} |\partial_y T_{\text{data}} - \partial_y \mathcal{G}_T(u_{\text{data}})|_{l^2}^2 \tag{8.21}$$

Here we obtain $\partial_y T_{\text{data}}$ by finite difference or spectral differentiation. The DeepONet are implemented in Python3 and TensorFlow2.

Index	**1**	**2**
Loss function	$\mathcal{L}_1$	$\mathcal{L}_2$
test MSE	5.313×10^{-3}	5.963×10^{-4}
test relative l^2	8.146×10^{-2}	2.115×10^{-2}
test mean l^∞	3.065×10^{-1}	1.093×10^{-1}
test max l^∞	2.888×10^{0}	1.999×10^{0}

From the table, we observe that using H^1 regularization can enhance the accuracy.

More benchmark problems can be found in Lu et al. (2022) and Luo et al. (2023b) (fluid flows) and in many related works.

Acknowledgments

GEK acknowledges support by the DOE SEA-CROGS project (DE-SC0023191), the MURI-AFOSR FA9550-20-1-0358 project, and the ONR Vannevar Bush Faculty Fellowship (N00014-22-1-2795).

Appendix 6.A Approximation of elementary functions with ReLU NNs

Note that the ReLU activation function satisfies the positive-homogeneity of order 1, i.e., for any $a > 0$, we have

$$\phi(ax) = a\phi(x),$$

where $\phi(x) = \max\{x, 0\}$. Therefore, for any $\mathrm{C} > 0$ and any ReLU network emulated by $\{W^\ell, b^\ell\}_{\ell=1}^L$, we have

$$\mathcal{R}[\theta](x) = \mathrm{C}^L \mathcal{R}[\tilde{\theta}_\mathrm{C}](x) \quad \text{where} \quad \tilde{\theta}_\mathrm{C} = \{\mathrm{C}^{-1}W^\ell, \mathrm{C}^{-1}b^\ell\}_{\ell=1}^L.$$

Therefore, in principle, there is a simple way of controlling the magnitude of weights and biases by setting $\mathrm{C} \gg 1$ and by expressing a (possibly) large number C^L with additional layers of controlled weights. However, such a naive construction will not consider the potentially complex structure of $\mathcal{R}[\theta]$ and may lead to unnecessarily large NN architectures at the end.

Lemma 6.A.1 (Magnitude control by depth and width). *For any $C > 0$, suppose $s \in \mathbb{N}$ and $r \in \mathbb{N}_{\geq 2}$ satisfy $C \leq s^{r-1}\mathrm{M}^r$. Then, there exists a r-layer ReLU network $\mathcal{R}[\theta] : \mathbb{R} \to \mathbb{R}$ with the architecture of $\vec{\boldsymbol{n}}_\theta = (s^{\oplus(r-1)})$ and $\mathcal{M}[\theta] = 2s + (r-2)s^2$ such that*

$$\mathcal{R}[\theta](x) = C\phi(x), \qquad |\theta|_\infty \leq \mathrm{M}.$$

Furthermore, assuming $r \in \mathbb{N}_{\geq 2}$, there exists a r-layer ReLU network $\mathcal{R}[\theta'] : \mathbb{R} \to \mathbb{R}$ with the architecture of $\vec{\boldsymbol{n}}_{\theta'} = (2s^{\oplus(r-1)})$ and $\mathcal{M}[\theta'] = 4s + 4(r-2)s^2$ such that

$$\mathcal{R}[\theta'](x) = Cx, \qquad |\theta'|_\infty \leq \mathrm{M}.$$

Proof. Let $W^1 = [\mathrm{M}, \cdots, \mathrm{M}]^\top \in \mathbb{R}^{s\times 1}$ and $W^r = \frac{\mathrm{C}}{s^{r-1}\mathrm{M}^r}[\mathrm{M}, \cdots, \mathrm{M}] \in \mathbb{R}^{1\times s}$. For $2 \leq l < r$, let $[W^l]_{ij} = \mathrm{M} \in \mathbb{R}^{s\times s}$. By letting $b^l = 0$ for all $1 \leq l \leq r$, it then can be checked that $\mathcal{R}[\theta](x) = \mathrm{C}\phi(x)$ with $\theta = \{W^l, b^l\}_{l=1}^r$ and $|\theta|_\infty \leq \mathrm{M}$.

Similarly, let $W^1 = [\mathrm{M}, \cdots, \mathrm{M}, -\mathrm{M}, \cdots, -\mathrm{M}]^\top \in \mathbb{R}^{2s\times 1}$, and $W^r = \frac{\mathrm{C}}{s^{r-1}\mathrm{M}^r}(W^1)^\top \in \mathbb{R}^{1\times 2s}$. For $2 \le l < r$, let $W_i^l = (W^1)^\top \in \mathbb{R}^{1\times 2s}$ for all $1 \le i \le 2s$. By letting $b^l = 0$ for all $1 \le l \le r$, it then can be checked that $\mathcal{R}[\theta'](x) = \mathrm{C}x$ with $\theta' = \{W^l, b^l\}_{l=1}^r$ and $|\theta'|_\infty \le \mathrm{M}$. □

Lemma 6.A.2 (Duplication of Width). *Let $\theta = \{W^\ell, b^\ell\}_{\ell=1}^L$ be a ReLU NN whose architecture is $\vec{n}_\theta = (n_1, \ldots, n_{L-1})$, let $k \in \mathbb{N}_{\ge 1}$, and $j \in \{1, \ldots, L-1\}$. Let $\theta'_{[j,k]} := \{\boldsymbol{W}^\ell_{[j,k]}, \boldsymbol{b}^\ell_{[j,k]}\}_{\ell=1}^L$ where*

$$(\textit{if } j = 1):$$

$$\begin{cases} \boldsymbol{W}^1_{[j,k]} = 1_{k\times 1} \otimes W^1, \boldsymbol{b}^1_{[j,k]} = 1_{k\times 1} \otimes b^1 \\ \boldsymbol{W}^\ell_{[j,k]} = 1_{k\times k} \otimes W^\ell, \boldsymbol{b}^\ell_{[j,k]} = k^{\ell-1} 1_{k\times 1} \otimes b^\ell & \textit{if } 2 \le \ell \le L-1 \\ \boldsymbol{W}^L_{[j,k]} = 1_{1\times k} \otimes W^L, \boldsymbol{b}^L_{[j,k]} = k^{L-1} b^L, \end{cases}$$

$$(\textit{if } 2 \le j \le L-1):$$

$$\begin{cases} \boldsymbol{W}^\ell_{[j,k]} = W^\ell, & \boldsymbol{b}^\ell_{[j,k]} = b^\ell & \textit{if } 1 \le \ell < j \\ \boldsymbol{W}^\ell_{[j,k]} = 1_{k\times k} \otimes W^\ell, & \boldsymbol{b}^\ell_{[j,k]} = k^{\ell-j} 1_{k\times 1} \otimes b^\ell & \textit{if } j \le \ell \le L-1 \\ \boldsymbol{W}^L_{[j,k]} = 1_{1\times k} \otimes W^L, & \boldsymbol{b}^L_{[j,k]} = k^{L-j} b^L. \end{cases}$$

Then, the ReLU network $\theta'_{[j,k]}$ satisfies

$$\mathcal{R}[\theta'_{[j,k]}](\cdot) = k^{L-j} \cdot \mathcal{R}[\theta](\cdot),$$

with the architecture of $\vec{n}_{\theta'_{[j,k]}} = (n_1, \ldots, n_{j-1}, kn_j, \ldots, kn_{L-1})$. Furthermore, $|\boldsymbol{W}^\ell_{[k]}|_\infty = |W^\ell|_\infty$ and $|\boldsymbol{b}^\ell_{[k]}|_\infty = k^{\max\{0,\ell-j\}}|b^\ell|_\infty$ for all $1 \le \ell \le L$. In particular, if $b^\ell = 0$ for all $\ell \ge j$, we have $|\theta'|_\infty = |\theta|_\infty$.

Lemma 6.A.3 (Concatenation of two networks). *Let $\theta_1 = \{W^l, b^l\}_{l=1}^{L_1}$ and $\theta_2 = \{A^l, c^l\}_{l=1}^{L_2}$ be two networks whose architectures are $\vec{n}_i$ and the output dimension of $\mathcal{R}[\theta_1]$ is equal to the input dimension of $\mathcal{R}[\theta_2]$. Then, there is a ReLU network θ that emulates the composition of the two networks, i.e., $\mathcal{R}[\theta](x) = \mathcal{R}[\theta_2] \circ \mathcal{R}[\theta_1](x)$ for all x with the architecture of $\vec{n} = (\vec{n}_1, \vec{n}_2)$ where*

$$\theta := \{(W^1, b^1), \cdots, (A^1 W^{L_1}, A^1 b^{L_1} + c^1), (A^2, c^2), \cdots, (A^{L_2}, c^{L_2})\}. \tag{6.A.1}$$

Proposition 6.A.4. *Let $\mathcal{R}[\theta_0] : \mathbb{R}^d \to \mathbb{R}$ be a L-layer ReLU NN with the architecture of $\vec{n}_0 = (n_1, \ldots, n_{L-1})$ and $|\theta_0|_\infty = \mathrm{B} > 0$. For any M such that $\mathrm{M} \in (0, \mathrm{B})$, $k \in \mathbb{N}$ and $j \in \{1, \ldots, L-1\}$, suppose $(s, r) \in \mathbb{N} \times \mathbb{N}$ satisfying $(\frac{\mathrm{B}}{\mathrm{M}})^L \le k^{L-j}(s\mathrm{M})^{r-1}$. Then, there exists a ReLU NN $\mathcal{R}[\theta]$ such that*

$$\mathcal{R}[\theta](x) = \mathcal{R}[\theta_0](x) \quad \forall x \in \mathbb{R}^d, \quad |\theta|_\infty = \mathrm{M},$$

with the architecture of $\vec{n}_\theta = (n_1, \ldots, n_{j-1}, kn_j, \ldots, kn_{L-1}, 2s^{\oplus(r-1)})$.

Proof. Let $\theta_0 = \{W^\ell, b^\ell\}_{\ell=1}^L$ and observe that

$$\mathcal{R}[\theta_0](x) = (\frac{\mathrm{B}}{\mathrm{M}})^L \mathcal{R}[\theta_{0,\mathrm{M}}](x), \quad \theta_{0,\mathrm{M}} = \{\frac{\mathrm{M}}{\mathrm{B}} W^\ell, \frac{\mathrm{M}}{\mathrm{B}} b^\ell\}_{\ell=1}^L,$$

which gives $|\theta_{0,\mathrm{M}}|_\infty \le \mathrm{M}$.

Suppose $(s, r) \in \mathbb{N} \times \mathbb{N}_{\ge 2}$ satisfying $(\frac{\mathrm{B}}{\mathrm{M}})^L \le (s\mathrm{M})^{r-1}$. Following the construction given in Lemma 6.A.1, but by setting the weight matrix of the 1st hidden layer as $[1, \ldots, 1, -1, \ldots, 1]^\top \in \mathbb{R}^{2s\times 1}$, we obtain a r-layer ReLU NN $\mathcal{R}[\theta']$ such that $\mathcal{R}[\theta'](x) = (\frac{\mathrm{B}}{\mathrm{M}})^L x$ for all x with the architecture of $\vec{n}_{\theta'} = (2s^{\oplus(r-1)})$. Let $\mathcal{R}[\theta]$ be the composition of $\mathcal{R}[\theta']$ and $\mathcal{R}[\theta_{0,\mathrm{M}}]$, which gives $\mathcal{R}[\theta](x) = \mathcal{R}[\theta'] \circ \mathcal{R}[\theta_0](x) = (\frac{\mathrm{B}}{\mathrm{M}})^L \mathcal{R}[\theta_{0,\mathrm{M}}](x)$ for all x. It then can be checked that the architecture of $\mathcal{R}[\theta]$ is $\vec{n}_\theta = (\vec{n}_{\theta_0}, 2s^{\oplus(r-1)})$. The layer that connects the last layer of θ_0 and the first layer of θ' (see (6.A.1)) is given by

$$[1, \cdots, 1, -1, \cdots, -1]^\top (\frac{\mathrm{M}}{\mathrm{B}}) W^L, \qquad [1, \cdots, 1, -1, \cdots, -1]^\top (\frac{\mathrm{M}}{\mathrm{B}}) b^L,$$

whose maximum norm is bounded by M. Therefore, $|\theta|_\infty = \mathrm{M}$. □

Lemma 6.A.5 (Heaviside). *For* $x = (x_1, \cdots, x_d) \in \mathbb{R}^d$, *let*

$$H(x) = \begin{cases} \mathbb{I}_{[0,\infty)}(x) & \text{if } d = 1, \\ \mathbb{I}_{[0,\infty)\times\mathbb{R}^{d-1}}(x) & \text{if } d \ge 2. \end{cases}$$

Suppose $(s, r, k) \in \mathbb{N}^3$ *such that* $\epsilon^{-1}\mathrm{M}^{-2} \le k(s\mathrm{M})^{r-1}$. *Then, there exists a ReLU network* θ *such that* $\vec{n}_\theta = (2k, 2s^{\oplus(r-1)})$ *and* $|\theta|_\infty = \mathrm{M}$ *and*

$$\|\mathcal{R}[\theta](\cdot) - H(\cdot)\|_{L^p(\mathbb{R}^d)} \le \epsilon^{-\frac{1}{p}}.$$

In particular, by letting $s = k$, *we have* $\vec{n}_\theta = (2s^{\oplus r})$.

Proof. First, we observe that $\forall (x_2, \ldots, x_d) \in \mathbb{R}^{d-1}$,

$$\tilde{H}_\epsilon(x) = \tilde{H}_\epsilon(x_1, \ldots, x_d) = \epsilon^{-1}[\phi(x_1) - \phi(x_1 - \epsilon)] = \begin{cases} 1 & \text{if } \epsilon \le x_1 \\ 0 & \text{if } x_1 \le 0 \\ \epsilon^{-1} x_1 & \text{if } 0 \le x_1 \le \epsilon \end{cases}$$

and $\|\tilde{H}_\epsilon(\cdot) - H(\cdot)\|_{L^p(\mathbb{R}^d)}^p \le \epsilon$. Assuming $\epsilon \in (0, 1]$, observe that $\tilde{H}_\epsilon(x) = \epsilon^{-1}\mathrm{M}^{-2}\mathcal{R}[\theta_0](x)$ where $\mathcal{R}[\theta_0](x) = \mathrm{M}(\phi([\mathrm{M}, \mathbf{0}]x) - \phi([\mathrm{M}, \mathbf{0}]x - \epsilon\mathrm{M}))$ with $\vec{n}_{\theta_0} = (2)$ $|\theta_0| = \mathrm{M}$. Suppose $(s, r) \in \mathbb{N} \times \mathbb{N}_{\ge 3}$ such that $\epsilon^{-1}\mathrm{M}^{-2} \le (s\mathrm{M})^{r-1}$. It then follows from Proposition 6.A.4 that there exists θ such that $\mathcal{R}[\theta] = \tilde{H}_\epsilon$ with $\vec{n}_\theta = (2, 2s^{\oplus(r-1)})$ and $|\theta| = \mathrm{M}$. □

Lemma 6.A.6 (Indicator 1D). *Let $(s,r,k) \in \mathbb{N}^3$ satisfying $\epsilon^{-1}\mathrm{M}^{-2} \le k(s\mathrm{M})^{r-1}$. For any $a, b \in [-1/2, 1/2]$ and $\epsilon \in (0, \frac{1}{2}]$, there exists a ReLU NN, θ such that*

$$\|\mathcal{R}[\theta](\cdot) - \mathbb{I}_{[a,b]}(\cdot)\|_{L_p(\mathbb{R})} = (\frac{2}{p+1}\epsilon)^{\frac{1}{p}},$$

with $\vec{n}_\theta = (4k, 2s^{\oplus(r-1)})$ and $|\theta|_\infty = \mathrm{M}$. In particular, if $s = 2k$, we have $\vec{n}_\theta = (2s^{\oplus r})$.

Proof. For any $a, b \in [-1/2, 1/2]$ such that $a < b$, let us consider a two-layer NN, θ_ϵ such that

$$\mathcal{R}[\theta_\epsilon](x) = \frac{1}{\epsilon}\left[\phi(x-a) - \phi(x-a-\epsilon) - \phi(x-b+\epsilon) + \phi(x-b)\right],$$

which satisfies $\|\mathcal{R}[\theta_0](\cdot) - \mathbb{I}_{[a,b]}(\cdot)\|_{L_p(\mathbb{R})} = (2\int_0^\epsilon (\frac{1}{\epsilon}x)^p dx)^{1/p} = (\frac{2}{p+1}\epsilon)^{1/p}$.

Assuming $\epsilon \le \frac{1}{2}$, observe that $\mathcal{R}[\theta_\epsilon](x) = \epsilon^{-1}\mathrm{M}^{-2} \cdot \mathcal{R}[\theta_0](x)$ where

$$\begin{aligned}\mathcal{R}[\theta_0](x) = \mathrm{M}\big[&\phi(\mathrm{M}x - \mathrm{M}a) - \phi(\mathrm{M}x - \mathrm{M}(a+\epsilon)) - \phi(\mathrm{M}x - \mathrm{M}(b-\epsilon)) \\ &+ \phi(\mathrm{M}x - \mathrm{M}b)\big],\end{aligned}$$

with $\vec{n}_{\theta_0} = (4)$ and $|\theta_0| = \mathrm{M}$.

Let $(s, r) \in \mathbb{N} \times \mathbb{N}_{\ge 3}$ satisfying $\epsilon^{-1}\mathrm{M}^{-2} \le (s\mathrm{M})^{r-1}$. It then follows from Proposition 6.A.4 that there is a ReLU NN, θ, such that $\mathcal{R}[\theta] = \mathcal{R}[\theta_\epsilon]$ with $\vec{n}_\theta = (4, 2s^{\oplus(r-1)})$ and $|\theta|_\infty = \mathrm{M}$. □

Lemma 6.A.7 (Parallelization of multiple NNs, Petersen and Voigtlaender, 2018). *For $j = 1, \ldots, N$, let θ^j be a L-layer NN whose architecture is $\vec{\boldsymbol{n}}_{\theta^j}$. Let us define the separate $\mathrm{P}_{\mathrm{sp}}(\{\theta^j\}_{j=1}^N)$ and the joint $\mathrm{P}_{\mathrm{jt}}(\{\theta^j\}_{j=1}^N)$ parallelizations of $\{\theta^j\}$ by*

$$\mathcal{R}[\mathrm{P}_{\mathrm{sp}}(\{\theta^j\}_{j=1}^N)](z_1, \ldots, z_N) = \begin{bmatrix}\mathcal{R}[\theta^1](z_1) \\ \vdots \\ \mathcal{R}[\theta^N](z_N)\end{bmatrix},$$

$$\mathcal{R}[\mathrm{P}_{\mathrm{jt}}(\{\theta^j\}_{j=1}^N)](x) = \begin{bmatrix}\mathcal{R}[\theta^1](x) \\ \vdots \\ \mathcal{R}[\theta^N](x)\end{bmatrix},$$

whose architectures are

$$\vec{n}_{\mathrm{P}_{\mathrm{sp}}(\{\theta^j\}_{j=1}^N)} = \vec{n}_{\mathrm{P}_{\mathrm{jt}}(\{\theta^j\}_{j=1}^N)} = \sum_{j=1}^N \vec{n}_{\theta^j}$$

and

$$|\mathrm{P}_{\mathrm{sp}}(\{\theta^j\}_{j=1}^N)|_\infty = |\mathrm{P}_{\mathrm{jt}}(\{\theta^j\}_{j=1}^N)|_\infty = \max_j |\theta^j|_\infty.$$

Remark: If the output dimension of $\mathcal{R}[\theta^j]$ is 1, the weight matrix of the last layer of $\mathrm{P}_{\mathrm{sp}}(\{\theta^j\}_{j=1}^N)$ is the form of the block diagonal matrix. This allows one to express $\sum_{j=1}^N \mathcal{R}[\theta^j](z_j)$ without affecting the magnitude of NN. For the notational simplicity, if there is no ambiguity, we denote the NN for such summation by $\mathrm{P}_{\mathrm{sp}}(\{\theta^j\}_{j=1}^N)$. A similar statement works for $\mathrm{P}_{\mathrm{jt}}(\{\theta^j\}_{j=1}^N)$ as well.

Lemma 6.A.8 (Indicator in d-dimension). *For $d \in \mathbb{N}$, let $a_i, b_i \in [-1/2, 1/2]$ with $a_i < b_i$ and $\frac{b_i - a_i}{2} > \epsilon$ for all $i = 1, \cdots, d$. Let $\mathrm{M} \geq 1$ be a given. For any $B > 1$ and $\epsilon \in (0, \frac{1}{2}]$, let $(s, r, r') \in \mathbb{N} \times \mathbb{N}_{\geq 2}^2$ be integers satisfying*

$$\epsilon^{-1}\mathrm{M}^{-1} \leq 0.5(s\mathrm{M})^{r-1},$$
$$B \leq ((ds+1)\mathrm{M})^{r'-1}$$

Then, there exists a ReLU network Φ such that for any $|f| \leq B$,

$$\|\mathcal{R}[\Phi](\cdot, f(\cdot)) - \mathbb{I}_{\prod_{i=1}^d [a_i, b_i]}(\cdot) f(\cdot)\|_{L_p(\mathbb{R}^d)} \leq 4dB\epsilon,$$

whose architecture is $\vec{n}_\Phi = (2ds+2)^{\oplus(r+r')}$ and $|\Phi|_\infty = \mathrm{M}$. Also, for any $|f|, |g| \leq B$, we have

$$\|\mathcal{R}[\Phi](\cdot, f(\cdot)) - \mathcal{R}[\Phi](\cdot, g(\cdot))\|_{L_p(\mathbb{R}^d)} \leq 2\|f - g\|_{L_p(\prod_{i=1}^d [a_i, b_i])}.$$

Proof. Let $\mathbf{x} = (x_1, \cdots, x_d) \in \mathbb{R}^d$ and $y \in \mathbb{R}$, and define

$$n(\mathbf{x}, y) = dB\phi\left(\frac{1}{d}\sum_{i=1}^d \mathcal{R}[\mathcal{I}_i](x_i) + \frac{1}{d}\phi(y/B) - 1\right)$$
$$- dB\phi\left(\frac{1}{d}\sum_{i=1}^d \mathcal{R}[\mathcal{I}_i](x_i) + \frac{1}{d}\phi(-y/B) - 1\right),$$

where $B \geq 1$ and $\mathcal{R}[\mathcal{I}_i](\cdot)$ is a neural network from Lemma 6.A.6 satisfying $\|\mathbb{I}_{[a_i,b_i]}(x) - \mathcal{R}[\theta_{\mathcal{I}_i}](x)\|_{L_p(\mathbb{R})} \leq \epsilon^{-\frac{1}{p}}$. It then can be checked that

$$n(\mathbf{x}, y) = \begin{cases} y & \text{if } \mathbf{x} \in \prod_{i=1}^d [a_i + \epsilon, b_i - \epsilon], \\ 0 & \text{if } \mathbf{x} \notin \prod_{i=1}^d [a_i, b_i]. \end{cases}$$

Since ϕ is 1-Lipschitz, we have $|n(\mathbf{x}, y)| \leq 2B$ whenever $|y| \leq B$. Thus, for any $|f| \leq B$, since the Lebesgue measure of $\prod_{i=1}^d [a_i, b_i] \setminus \prod_{i=1}^d [a_i + \epsilon, b_i - \epsilon]$ is bounded by $(2d)\epsilon$, we have

$$\|n(\mathbf{x}, f(\cdot)) - \mathbb{I}_{\prod_{i=1}^{d}[a_i,b_i]}(\cdot)f(\cdot)\|_{L_p(\mathbb{R}^d)} \le (2B)(2d)\epsilon.$$

The rest of the proof constructs a ReLU network that exactly emulates $n(\mathbf{x}, y)$. Let the architecture of $\mathcal{I}_i$ be $\vec{n}_{\mathcal{I}} = (2s^{\oplus r})$ for all i.

Suppose $B > 1$ and $\mathrm{M} \ge 1$. Let $\theta^4 = \{W_4^\ell, b_4^\ell\}_{\ell=1}^{r+1}$ where $W_4^1 = [B^{-1}; -B^{-1}]$, $W_4^\ell = I_2$ for $2 \le \ell$ and $b_4^\ell = 0$. Then, $\mathcal{R}[\theta^4](y) = \begin{bmatrix} \phi(y/B) \\ \phi(-y/B) \end{bmatrix}$ with $\vec{n}_{\theta^4} = (2^{\oplus r})$ and $|\theta^4|_\infty = \mathrm{M}$.

By modifying the bias vector of the last layer in $\mathcal{I}_i$, one can find $\mathcal{I}_i'$ such that $\mathcal{R}[\mathcal{I}_i'](x) = \mathcal{R}[\mathcal{I}_i](x) - 1$ while having the same architectures and magnitude. Let $\theta^1 = \mathrm{P}_{\mathrm{sp}}(\{\mathcal{I}_i'\}_{i=1}^d \cup \theta^4)$ such that

$$\mathcal{R}\left[\mathrm{P}_{\mathrm{sp}}(\{\mathcal{I}_i\}_{i=1}^d \cup \theta^4)\right](\mathbf{x}, y) = \begin{bmatrix} \mathcal{R}[\mathcal{I}_1](x_1) - 1 \\ \vdots \\ \mathcal{R}[\mathcal{I}_d](x_d) - 1 \\ \mathcal{R}[\theta^4](y) \end{bmatrix}$$

with $|\theta^1|_\infty = \mathrm{M}$ and $\vec{n}_{\theta^1} = (d2s + 2)^{\oplus r}$.

Let $\theta^5 = \{W_5^l, b_5^l\}_{l=1}^2$ where $b_5^l = 0$, $W_5^2 = 1_{1\times k} \otimes [1, -1]$, and

$$W_5^1 = (ds+1)^{-1} 1_{k\times 1} \begin{bmatrix} 1 & 1 & \cdots & 1 & 1 & 0 \\ 1 & 1 & \cdots & 1 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{2k\times(d+2)}.$$

Then consider $\Theta := \theta^5 \bullet \theta^1$. Then, it can be checked that $\mathcal{R}[\Theta](\mathbf{x}, y) = \frac{k}{(ds+1)B} n(\mathbf{x}, y)$ with $|\Theta|_\infty = \mathrm{M}$. By letting $k = ds + 1$, we have $\vec{n}_\Theta = (2ds + 2)^{\oplus(r+1)}$.

Choose $r' \in \mathbb{N}$ such that $B \le ((ds+1)\mathrm{M})^{r'-1}$. It then follows from Proposition 6.A.4 that there exists a NN θ such that $\mathcal{R}[\theta] = B\mathcal{R}[\Theta] = n(\mathbf{x}, y)$ with $|\theta| = \mathrm{M}$ and $\vec{n}_\theta = (2ds+2)^{\oplus(r+r'+1)}$. □

Lemma 6.A.9 (Sum of Indicators). *For $d \in \mathbb{N}$, let $a_{i,l}, b_{i,l} \in [-1/2, 1/2]$ with $a_{i,l} < b_{i,l}$ and $\frac{b_{i,l}-a_{i,l}}{2} > \epsilon$ for all $i = 1, \cdots, d$ and $l = 1, \cdots, N$. Let $\mathrm{M} \ge 1$ be a given. For any $B > 1$ and $\epsilon \in (0, \frac{1}{2}]$, let $(s, r, r') \in \mathbb{N} \times \mathbb{N}_{\ge 2}^2$ be integers satisfying*

$$\epsilon^{-1}\mathrm{M}^{-1} \le 0.5(s\mathrm{M})^{r-1},$$
$$B \le ((ds+1)\mathrm{M})^{r'-1}$$

Let Ψ be a $\mathbb{R}^N$-valued ReLU NN whose architecture is $\vec{n}_\Psi$ with $|\Psi|_\infty \le \mathrm{M}$ and $|\mathcal{R}[\Psi]_l(x)| \le B$ for all l and for all $x \in \mathbb{R}^d$. Then, there exists a ReLU NN Φ

such that

$$\left\| \mathcal{R}[\Phi](\cdot) - \sum_{l=1}^{N} \mathbb{I}_{\prod_{i=1}^{d}[a_{i,l},b_{i,l}]} \mathcal{R}[\Psi]_l(\cdot) \right\|_{L_p([-1/2,1/2]^d)} \leq 4dBN\epsilon,$$

whose architecture is $\vec{n}_\Phi = (\vec{n}_\Psi, (2N(ds+1))^{\oplus(r+r')})$ *and* $|\Phi|_\infty \leq \mathrm{M}$.

Proof. Let Ψ be a given network having L_0 layers and Id_d be a L_0-layer identity network such that $\mathcal{R}[\mathrm{Id}_d](x) = x$ for all $x \in \mathbb{R}^d$ with $\vec{\boldsymbol{n}}_{\mathrm{Id}_d} = (2d, \cdots, 2d)$. Let $\theta_0 = \mathrm{P}_{\mathrm{jt}}(\{\mathrm{Id}_d, \Psi\})$. Then, $\vec{\boldsymbol{n}}_{\theta_0} = (n+2d)^{\oplus(L_0-1)}$ and $\mathcal{R}[\theta_0](x) = \begin{bmatrix} x \\ \mathcal{R}[\Psi](x) \end{bmatrix}$ with $|\theta_0| \leq \mathrm{M}$.

Let $\Theta_{0,k}$ be a $L = (r + r' + 1)$-layer network from Lemma 6.A.8 such that $\mathcal{R}[\Theta_{0,k}](x, y) \approx \mathbb{I}_{\prod_{i=1}^{d}[a_{i,k},b_{i,k}]}(x)y$ for all $|y| \leq B$ with $\vec{n}_{\Theta_{0,k}} = (2ds+2)^{\oplus(L-1)}$. Let $\tilde{\theta}_{1,k}$ be a NN such that $\mathcal{R}[\tilde{\theta}_{1,k}](x, v) = \mathcal{R}[\Theta_{0,k}](x, v_k)$ which can be obtained by modifying the first layer of $\Theta_{0,k}$. Then, $\vec{n}_{\tilde{\theta}_{1,k}} = \vec{n}_{\Theta_{0,k}}$ with $|\tilde{\theta}_{1,k}|_\infty = |\Theta_{0,k}|_\infty$. Let $\theta_1 = \mathrm{P}_{\mathrm{jt}}(\{\tilde{\theta}_{1,j}\}_{j=1}^{N})$ such that $\mathcal{R}[\theta_1](v, x) = \sum_{k=1}^{N} \mathcal{R}[\Theta_{0,k}](x, v_k)$ with $\vec{n}_{\theta_1} = (2N(ds+1))^{\oplus(r+r')}$ and $|\theta_1| \leq \mathrm{M}$.

Finally, let $\Phi := \theta_1 \bullet \theta_0$. It can be checked that $\mathcal{R}[\Phi](x) = \sum_{l=1}^{N} \mathcal{R}[\Theta_{0,k}](x, \mathcal{R}[\Psi]_l(x))$ with $\vec{n}_\Phi = (\vec{n}_\Psi, (2N(ds+1))^{\oplus(L-1)})$ and $|\Phi|_\infty \leq \mathrm{M}$.

Therefore,

$$\begin{aligned}
&\left\| \mathcal{R}[\Phi](x) - \sum_{l=1}^{N} \mathbb{I}_{\prod_{i=1}^{d}[a_{i,l},b_{i,l}]} \mathcal{R}[\Psi]_l(x) \right\|_{L_p([-1/2,1/2]^d)} \\
&= \left\| \sum_{l=1}^{N} n(x, \mathcal{R}[\Psi]_l(x)) - \sum_{l=1}^{N} \mathbb{I}_{\prod_{i=1}^{d}[a_{i,l},b_{i,l}]} \mathcal{R}[\Psi]_l(x) \right\|_{L_p([-1/2,1/2]^d)} \\
&\leq \sum_{l=1}^{N} \| n(x, \mathcal{R}[\Psi]_l(x)) - \mathbb{I}_{\prod_{i=1}^{d}[a_{i,l},b_{i,l}]} \mathcal{R}[\Psi]_l(x) \|_{L_p([-1/2,1/2]^d)} \\
&\leq 4dBN\epsilon,
\end{aligned}$$

which completes the proof. □

Lemma 6.A.10 (Square x^2). *For any* $m \in \mathbb{N}$, *let* $L \geq 2$ *and* $k \geq 1$ *such that* $m = k(L-1)$. *Then, there exists a ReLU network* θ *such that*

$$\|\mathcal{R}[\theta](x) - x^2\|_{L_\infty[0,1]} \leq 2^{-2(m+1)},$$

with the architecture of $\vec{n}_\theta = (2^k + 2)^{\oplus(L-1)}$ *and* $|\theta|_\infty \leq 1$.

Proof. We mainly follow the proof of Lemma A.3 of Petersen and Voigtlaender (2018). Let us first define as in Yarotsky (2022) the function

$$g(x) = \begin{cases} 2x & \text{if } 0 \le x < 0.5 \\ 2(1-x) & \text{if } 0.5 \le x \ge 1 \\ 0 & \text{elsewhere.} \end{cases}$$

For $t \in \mathbb{N}$, let $g_t(x) = \underbrace{g \circ g \circ \cdots g}_{t\text{-times}}$ be the t-times composition of g. It then can be checked that

$$g_t(x) = \begin{cases} 2^t(x - k/2^{t-1}) & \text{if } \frac{2k}{2^t} \le x \le \frac{2k+1}{2^t} \text{ for some } k \in \{0, 1, \cdots, 2^{t-1}-1\} \\ -2^t(x - k/2^{t-1}) & \text{if } \frac{2k-1}{2^t} \le x \le \frac{2k}{2^t} \text{ for some } k \in \{1, \cdots, 2^{t-1}\}, \\ 0 & \text{elsewhere.} \end{cases}$$

Let $f_m(x) = x - \sum_{t=1}^{m} 4^{-t} g_t(x)$. It then follows from Proposition 2 of Yarotsky (2022) that

$$\|f_m(x) - x^2\|_{L_\infty[0,1]} \le 2^{-2-2m}.$$

With this known result in mind, we now construct a ReLU network that exactly represents $f_m(x)$, which already has been done in Petersen and Voigtlaender (2018). However, we require all the network parameters to be bounded in absolute values by M.

Let us define $g_t(x;k)$ as follows:

$$g_t(x;k) = 2^t \left[\phi(x) + 2\sum_{s=1}^{2^t-1} (-1)^s \phi\left(x - 4^{-k}\frac{s}{2^t}\right) + \phi(x - 4^{-k}) \right].$$

We note that $g_t(x;0) = g_t(x)$ and $4^{-t} g_t(4^{-s} g_s(x); s) = 4^{-t-s} g_{t+s}(x)$.

Let $\boldsymbol{g}_{t,k}, \boldsymbol{c}_{t,k}, \boldsymbol{d}_{t,k} \in \mathbb{R}^{2^t+1}$ defined by

$$[\boldsymbol{g}_{t,k}]_i = \begin{cases} 4^{-k} & \text{if } i = 1 \text{ or } 2^t + 1 \\ 2 \cdot 4^{-k} & \text{if } 1 < i < 2^t + 1 \end{cases},$$

$$[\boldsymbol{d}_{t,k}]_i = \begin{cases} 0 & \text{if } i = 1 \\ -4^{-k} & \text{if } i = 2^t + 1 \\ -4^{-k}\frac{i}{2^{t-1}} & \text{if } 1 < i < 2^t + 1 \end{cases},$$

and $[\boldsymbol{c}_t]_i = 2^{-t}(-1)^{i+1}$, $[\boldsymbol{s}_t]_i = \sum_{(s,p)\in\Omega_t^i} 2^{-s}(-1)^{p+1}$ where

$$\Omega_t^i = \{(s,p) \in \{1, \cdots, t\} \times \{1, \cdots, 2^s\} : i = p2^{t-s}\}.$$

It then can be checked that

$$4^{-m}g_m(x) = \boldsymbol{c}_m^T\phi(\boldsymbol{g}_{m,0}x + \boldsymbol{d}_{m,0}), \qquad \sum_{s=1}^{m} 4^{-s}g_s(x) = \boldsymbol{s}_m^T\phi(\boldsymbol{g}_{m,0}x + \boldsymbol{d}_{m,0}).$$

Since $x = \phi(x)$ in $[0, 1]$ and $f_m(x)$ can be exactly represented by a two-layer network of width $2^m + 1$ whose weights and biases are all bounded by 1 (in absolute values) and the number of nonzero weights and biases is $3 \cdot 2^m + 2$.

For the deep ReLU construction, consider

$$C^1\phi(W^1x + b^1) = \begin{bmatrix} 4^{-t}g_t(x) \\ x - \sum_{s=1}^{t} 4^{-s}g_s(x) \end{bmatrix}$$

where

$$W^1 = \begin{bmatrix} \boldsymbol{g}_{t,0} \\ 1 \end{bmatrix} \in \mathbb{R}^{2^t+2}, \quad b^1 = \begin{bmatrix} \boldsymbol{d}_{t,0} \\ 0 \end{bmatrix} \in \mathbb{R}^{2^t+2},$$
$$C^1 = \begin{bmatrix} \boldsymbol{c}_t^T & 0 \\ -\boldsymbol{s}_t^T & 1 \end{bmatrix} \in \mathbb{R}^{2\times(2^t+2)}.$$

Next, we observe that

$$\begin{aligned}\boldsymbol{s}_t^T\phi(\boldsymbol{g}_{k,0}(4^{-t}g_t(x)) + \boldsymbol{d}_{k,t}) &= 4^{-t}\boldsymbol{s}_t^T\phi(\boldsymbol{g}_{k,0}(g_t(x)) + \boldsymbol{d}_{k,0}) \\ &= 4^{-t}\sum_{s=1}^{k} 4^{-s}g_s(g_t(x)) = \sum_{s=t+1}^{t+k} 4^{-s}g_s(x).\end{aligned}$$

Thus, we have

$$C\phi\left(V\begin{bmatrix} 4^{-t}g_t(x) \\ x - \sum_{s=1}^{t} 4^{-s}g_s(x) \end{bmatrix} + b\right) = \begin{bmatrix} 4^{-t-k}g_{t+k}(x) \\ x - \sum_{s=1}^{t+k} 4^{-s}g_s(x) \end{bmatrix},$$

where

$$V = \begin{bmatrix} \boldsymbol{g}_{k,0} & 0 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{(2^k+2)\times 2}, \quad b = \begin{bmatrix} \boldsymbol{d}_{k,t} \\ 0 \end{bmatrix} \in \mathbb{R}^{2^k+2},$$
$$C = \begin{bmatrix} \boldsymbol{c}_k^T & 0 \\ -\boldsymbol{s}_k^T & 1 \end{bmatrix} \in \mathbb{R}^{2\times(2^k+2)}.$$

For any integers $L \geq 3$ and $k_j \geq 1$ for $j = 2, \cdots, L$, let $m = \sum_{j=2}^{L} k_j$ and set

$$W^j = \begin{bmatrix} \boldsymbol{g}_{k_j,0}\boldsymbol{c}_{k_{j-1}}^T & 0 \\ -\boldsymbol{s}_{k_{j-1}}^T & 1 \end{bmatrix}, \quad b^j = \begin{bmatrix} \boldsymbol{d}_{k_j,\sum_{s=2}^{j} k_s} \\ 0 \end{bmatrix}, \qquad 1 < j < L,$$

and $W^L = [-\boldsymbol{s}_{k_L}^T, 1] \in \mathbb{R}^{1\times(2^{k_L}+2)}$ and $b^L = 0$. Then, $\theta = \{W^j, b^j\}_{j=1}^L$ is a L-layer ReLU network with the architecture of

$$\vec{\boldsymbol{n}} = (2^{k_2}+2, 2^{k_3}+2, \cdots, 2^{k_L}+2),$$

such that $\mathcal{R}[\theta](x) = f_m(x)$. Also, since every elements of $\boldsymbol{g}_{k,0}$ and $\boldsymbol{c}_k$ are less than one, all the parameters are bounded by 1 in absolute values. □

Lemma 6.A.11 (Multiplication). *For $B \ge \frac{1}{2}$ and $m \in \mathbb{N}$, let $(L, k, r) \in \mathbb{N}^3$ such that*

$$m = k(L-1),$$
$$B^2 \le (3(2^k+2))^{r-1}.$$

Then, there exists a ReLU network $\tilde{\times}$ satisfying

- *for all $x, y \in [-B, B]$, we have $|xy - \mathcal{R}[\tilde{\times}](x,y)| \le 2^{-2m}$;*
- *for all $x, y \in [-B, B]$ with $xy = 0$, we have $\mathcal{R}[\tilde{\times}](x,y) = 0$;*
- *$|\tilde{\times}|_\infty \le 1$ and $\vec{n}_{\tilde{\times}} = (3(2^k+2))^{\oplus(L+r-1)}$*

Proof. As in the proof of Proposition 3 in Yarotsky (2022), it follows from $xy = \frac{(x+y)^2-x^2-y^2}{2}$ that we define

$$h_m(x,y) = \frac{B^2}{2}\left[f_m\left(\frac{|x+y|}{B}\right) - f_m\left(\frac{|x|}{B}\right) - f_m\left(\frac{|y|}{B}\right)\right].$$

Then, $|h_m(x,y) - xy| \le \left(\frac{B}{2^m}\right)^2$.

Let $\theta^0 = \{W_0^l, b_0^l\}_{l=1}^2$ be a two-layer network of width 12 such that $\mathcal{R}[\theta^0](x,y) = (|x+y|/B, |x|/B, |y|/B)^\top$ and $|\theta^0|_\infty \le 1$.

By Lemma 6.A.10, there exists a network θ^1 such that $\mathcal{R}[\theta^1](x) = f_m(x)$. Then $\mathrm{P}_{\mathrm{sp}}(\theta^1)$ gives $\mathcal{R}[\mathrm{P}_{\mathrm{sp}}(\theta^1)](x,y,z) = \frac{f_m(x)-f_m(y)-f_m(z)}{2}$ with $|\theta^1|_\infty \le 1$ and $\vec{n}_{\theta^1} = (3(2^k+2))^{\oplus(L-1)}$.

Let $\theta = \mathrm{P}_{\mathrm{sp}}(\theta^1) \bullet \theta^0$. It then can be checked that $\mathcal{R}[\theta](x,y) = B^{-2}h_m(x,y)$ with $|\theta|_\infty \le 1$ and $\vec{n}_\theta = (12, (3(2^k+2))^{\oplus(L-1)})$. Lastly, let $(s,r) \in \mathbb{N}^2$ such that $B^2 \le s^{r-1}$. It then follows from Proposition 6.A.4 that there exists θ' such that $\mathcal{R}[\theta'](x,y) = h_m(x,y)$ with $|\theta'| \le 1$ and $\vec{n}_{\theta'} = (12, (3(2^k+2))^{\oplus(L-1)}, s^{\oplus(r-1)})$. □

Lemma 6.A.12 (Monomial). *Let $\boldsymbol{\alpha} = (\alpha_1, \cdots, \alpha_d)$ be a multiindex and let $\mathfrak{n} = \lceil \log_2 |\boldsymbol{\alpha}| \rceil$. For any integers $L \in \mathbb{N}$, let $k \ge 1$ such that $m = k(L-1) \ge \frac{1}{2}\log_2\left(\frac{\mathfrak{n}}{2}\right)$. Then, there exists a ReLU NN $\Theta_{\boldsymbol{\alpha}}$ satisfying*

- *$|x^{\boldsymbol{\alpha}} - \mathcal{R}[\Theta_{\boldsymbol{\alpha}}](x)| \le \mathfrak{n}2^{-2(m+1)}$ for all $x \in [-1/2, 1/2]^d$,*
- *$|\Theta_{\boldsymbol{\alpha}}|_\infty \le 1$, and $\vec{n}_{\Theta_{\boldsymbol{\alpha}}} = [2^{\mathfrak{n}-1}, \ldots, 2^0] \otimes \vec{n}_{\tilde{\times}}$ where $\vec{n}_{\tilde{\times}} = (3(2^k+2))^{\oplus L}$.*

Proof. Let $x = (x_1, \cdots, x_{n_0}) \in \mathbb{R}^{n_0}$. By letting $\theta^1 = \{W_1^1, b_1^1\}$ where $W_1^1 = e_j^T \in \mathbb{R}^{1\times n_0}$ and $b_1^1 = 0$, it can be checked that $\mathcal{R}[\theta^1](x) = x_j$. Also, by letting $\theta^0 = \{W_0^1 = 0_{1\times n_0}, b_0^1 = 1\}$, we have $\mathcal{R}[\theta^0](x) = 1$. Thus, any monomials of degree ≤ 1 can be represented by ReLU networks.

Let $\boldsymbol{\alpha} = (\alpha_1, \cdots, \alpha_d)$ be a multiindex such that

$$x^{\boldsymbol{\alpha}} = x_1^{\alpha_1} \cdots x_d^{\alpha_d}, \qquad |\boldsymbol{\alpha}| = \sum_{j=1}^{d} \alpha_j.$$

For any $\boldsymbol{\alpha}$, let us define $\pi_{\boldsymbol{\alpha}} : \{1, \cdots, |\boldsymbol{\alpha}|\} \to \{1, \cdots, d\}$ such that $\pi_{\boldsymbol{\alpha}}(i) = j$ if $\sum_{l=1}^{j-1} \alpha_l < i \leq \sum_{l=1}^{j} \alpha_l$. Then, there is a 1-layer ReLU network $\bar{\theta}$ such that $\mathcal{R}[\bar{\theta}](x) \in \mathbb{R}^{|\boldsymbol{\alpha}|}$ where $[\mathcal{R}[\bar{\theta}](x)]_i = x_{\pi_{\boldsymbol{\alpha}}(i)}$ for $1 \leq i \leq |\boldsymbol{\alpha}|$. It then can be checked that $\mathcal{M}(\bar{\theta}) = |\boldsymbol{\alpha}|$.

First, let us assume that $|\boldsymbol{\alpha}| = 2^{\mathfrak{n}}$ for any $\mathfrak{n} \in \mathbb{N}$. Let $\tilde{\times}$ be the multiplication network from Lemma 6.A.11 with $B = 1/2$ such that $|\mathcal{R}[\tilde{\times}](x, y) - xy| \leq 2^{-2(m+1)}$ for all $x, y \in [-0.5, 0.5]$. Then, let $\times_{\mathfrak{n}} := \mathrm{P}_{\mathrm{sp}}^{2^{\mathfrak{n}-1}}(\tilde{\times})$ if $\mathfrak{n} > 1$ and $\times_1 := \tilde{\times}$. By recursively applying similar procedures, we define

$$\times_{\mathrm{Full}}^{\boldsymbol{\alpha}} := \bar{\times}_1^{\mathfrak{n}}, \quad \text{where} \quad \bar{\times}_j^{\mathfrak{n}} := \times_j \bullet \cdots \bullet \times_{\mathfrak{n}-1} \bullet \times_{\mathfrak{n}} \bullet \bar{\theta}, \quad 1 \leq j \leq \mathfrak{n},$$

whose realization approximates $x^{\boldsymbol{\alpha}}$ within the error of $\mathfrak{n}2^{-2(m+1)}$, which can be verified as follows. First, note that the network architecture of $\times_{\mathrm{Full}}^{\boldsymbol{\alpha}}$ is given by

$$\vec{\boldsymbol{n}}_{\times_{\mathrm{Full}}^{\boldsymbol{\alpha}}} = [2^{\mathfrak{n}-1}, \ldots, 2^0] \otimes \vec{n}_{\tilde{\times}}.$$

We prove the error bound by induction on $\mathfrak{n}$. If $\mathfrak{n} = 1$, we have $|\boldsymbol{\alpha}| = 2$ and

$$\begin{aligned}|x_{\pi_{\boldsymbol{\alpha}}(1)} x_{\pi_{\boldsymbol{\alpha}}(2)} - \mathcal{R}[\times_{\mathrm{Full}}^{\boldsymbol{\alpha}}](x)| &= |x_{\pi_{\boldsymbol{\alpha}}(1)} x_{\pi_{\boldsymbol{\alpha}}(2)} - \mathcal{R}[\tilde{\times}](x_{\pi_{\boldsymbol{\alpha}}(1)}, x_{\pi_{\boldsymbol{\alpha}}(2)})| \\ &\leq 2^{-2(m+1)},\end{aligned}$$

which proves the claim for the case of $\mathfrak{n} = 1$.

Suppose the statement is true for $\tilde{\mathfrak{n}} < \mathfrak{n}$. For notational convenience, let us denote $\mathcal{R}[\theta]$ as θ. Observe that

$$\begin{aligned}&\left|\prod_{i=1}^{2^{\mathfrak{n}}} x_{\pi_{\boldsymbol{\alpha}}(i)} - \mathcal{R}[\times_{\mathrm{Full}}^{\boldsymbol{\alpha}}](x)\right| = \left|\prod_{i=1}^{2^{\mathfrak{n}}} x_{\pi_{\boldsymbol{\alpha}}(i)} - \mathcal{R}[\tilde{\times}]([\bar{\times}_2^{\mathfrak{n}}(x)]_1, [\bar{\times}_2^{\mathfrak{n}}(x)]_2)\right| \\ &\leq \left|\prod_{i=1}^{2^{\mathfrak{n}}} x_{\pi_{\boldsymbol{\alpha}}(i)} - [\bar{\times}_2^{\mathfrak{n}}(x)]_1 [\bar{\times}_2^{\mathfrak{n}}(x)]_2\right| \\ &\quad + \left|[\bar{\times}_2^{\mathfrak{n}}(x)]_1 [\bar{\times}_2^{\mathfrak{n}}(x)]_2 - \mathcal{R}[\tilde{\times}]([\bar{\times}_2^{\mathfrak{n}}(x)]_1, [\bar{\times}_2^{\mathfrak{n}}(x)]_2)\right| \\ &\leq 2^{-2(m+1)} + \left|\prod_{i=1}^{2^{\mathfrak{n}-1}} x_{\pi_{\boldsymbol{\alpha}}(i)} - [\bar{\times}_2^{\mathfrak{n}}(x)]_1\right| \cdot \left|[\bar{\times}_2^{\mathfrak{n}}(x)]_2\right|\end{aligned}$$

$$+\left|\prod_{i=2^{\mathfrak{n}-1}+1}^{2^{\mathfrak{n}}} x_{\pi_{\alpha}(i)}-[\bar{\times}_{2}^{\mathfrak{n}}(x)]_2\right|\left|\prod_{i=1}^{2^{\mathfrak{n}-1}} x_{\pi_{\alpha}(i)}\right|$$
$$\leq 2^{-2(m+1)}+(\mathfrak{n}-1)2^{-2(m+1)}=\mathfrak{n}2^{-2(m+1)},$$

where the last inequality uses the induction hypothesis and the facts that $\left|\prod_{i=1}^{2^{\mathfrak{n}-1}} x_{\pi_{\alpha}(i)}\right| \leq 2^{-2}$ and $\left|[\times_{2}^{\mathfrak{n}}(x)]_2\right| \leq (\mathfrak{n}-1)2^{-2(m+1)}+2^{-2} \leq 1-2^{-2}$. Note that m is chosen to satisfy

$$m \geq \frac{1}{2}\log_2\left(\frac{\mathfrak{n}-1}{2}\right).$$

It now suffices to prove the case where $2^{\mathfrak{n}-1}<|\boldsymbol{\alpha}|<2^{\mathfrak{n}}$. Let $\boldsymbol{\alpha}_1$, $\boldsymbol{\alpha}_2$ be multiindices such that $\boldsymbol{\alpha}_1+\boldsymbol{\alpha}_2=\boldsymbol{\alpha}$ where $|\boldsymbol{\alpha}_1|=2^{\mathfrak{n}-1}$ and $|\boldsymbol{\alpha}_2|=|\boldsymbol{\alpha}|-2^{\mathfrak{n}-1}$. Let us consider the following network. Given $\boldsymbol{\alpha}_2$, let $\hat{\theta}$ be a one-layer network with the architecture of $\vec{\boldsymbol{n}}_{\hat{\theta}}=(n_0,|\boldsymbol{\alpha}|)$ such that $[\mathcal{R}[\hat{\theta}](x)]_i=x_{\pi_{\alpha}(i)}$. Let $\mathrm{H}(x)=\lceil\frac{x}{2}\rceil$, $\mathrm{H}^{(j)}(x)=\overbrace{\mathrm{H}\circ\cdots\circ\mathrm{H}}^{j\text{ times}}(x)$ and $\mathrm{H}^{(0)}(x)=x$. Let $h_{|\boldsymbol{\alpha}|}^{j}:=\mathrm{H}^{(\mathfrak{n}-j)}(|\boldsymbol{\alpha}|)$. Since $2^{\mathfrak{n}-1}<|\boldsymbol{\alpha}|<2^{\mathfrak{n}}$, $h_{|\boldsymbol{\alpha}|}^{1}=2$.

Let us recursively define

$$\underline{\times}_{\mathrm{Full}}^{\boldsymbol{\alpha}}:=\underline{\times}_{1}^{\mathfrak{n}},\quad \text{where}\quad \underline{\times}_{j}^{\mathfrak{n}}:=\hat{\times}_j\bullet\cdots\bullet\hat{\times}_{\mathfrak{n}-1}\bullet\hat{\times}_{\mathfrak{n}}\bullet\hat{\theta},\quad 1\leq j\leq\mathfrak{n},$$

where $\hat{\times}_j$ is a network such that

$$\hat{\times}_j=\begin{cases}\mathrm{P}^{\lfloor\frac{h_{\alpha}^{j}}{2}\rfloor}(\tilde{\times}), & \text{if } h_{\alpha}^{j} \text{ is even,}\\ \mathrm{P}(\mathrm{P}^{\lfloor\frac{h_{\alpha}^{j}}{2}\rfloor}(\tilde{\times}),\mathrm{Id}_{L+1,1}) & \text{if } h_{\alpha}^{j} \text{ is odd,}\end{cases}$$

whose architecture is

$$\vec{\boldsymbol{n}}_{\hat{\times}_j}=\begin{cases}\lfloor\frac{h_{\alpha}^{j}}{2}\rfloor\tilde{\times} & \text{if } h_{\alpha}^{j} \text{ is even,}\\ \lfloor\frac{h_{\alpha}^{j}}{2}\rfloor\tilde{\times}+2^{\oplus L} & \text{if } h_{\alpha}^{j} \text{ is odd}\end{cases}$$

that satisfies

$$\left[\mathcal{R}[\underline{\times}_{j}^{\mathfrak{n}}](x)\right]_i$$
$$=\begin{cases}\mathcal{R}[\tilde{\times}]\left([\mathcal{R}[\underline{\times}_{j-1}^{\mathfrak{n}}](x)]_{2i-1},[\mathcal{R}[\underline{\times}_{j-1}^{\mathfrak{n}}](x)]_{2i}\right), & \text{if } 1\leq i\leq\lfloor\frac{h_{\alpha}^{j}}{2}\rfloor,\\ [\mathcal{R}[\underline{\times}_{j-1}^{\mathfrak{n}}](x)]_{h_{\alpha}^{j}}, & \text{if } h_{\alpha}^{j} \text{ is odd}\end{cases}$$

It then can be checked that $\left|x^{\boldsymbol{\alpha}} - \mathcal{R}[\underline{\times}^{\boldsymbol{\alpha}}_{\text{Full}}](x)\right| \le \mathfrak{n}2^{-2(m+1)}$, and the architecture of $\underline{\times}^{\boldsymbol{\alpha}}_{\text{Full}}$ is given by

$$\vec{\boldsymbol{n}}_{\underline{\times}^{\boldsymbol{\alpha}}_{\text{Full}}} = (d, \boldsymbol{v}^{\mathfrak{n}-1}_{\tilde{\times}}, \cdots, \boldsymbol{v}^{0}_{\tilde{\times}}, 1),$$
$$\text{where} \quad \boldsymbol{v}^{\mathfrak{n}-1-j}_{\tilde{\times}} = \lfloor \frac{h^j_{\boldsymbol{\alpha}}}{2} \rfloor \tilde{\times} + s_j 2^{\oplus L}, \quad s_j \equiv h^j_{\boldsymbol{\alpha}} \pmod 2),$$

for $0 \le j < \mathfrak{n}$. By observing $\lfloor \frac{h^j_{\boldsymbol{\alpha}}}{2} \rfloor + s_j \le 2^{\mathfrak{n}-1-j}$, the proof is completed. □

Lemma 6.A.13 (Polynomials). *Let $q \in \mathbb{N}$, $\{x_l\}_{l=1}^q \subset [-1/2, 1/2]^d$, $\mathfrak{n} = \lceil \log_2 n \rceil$. Let $\gamma_{j,\ell} = \sum_{j \le \boldsymbol{\alpha}, |\boldsymbol{\alpha}| \le n} c_{\boldsymbol{\alpha},\ell} \binom{\boldsymbol{\alpha}}{j} (-x_\ell)^{\boldsymbol{\alpha}-j}$ and let $\overline{\gamma} := \max_{|j| \le n, 1 \le \ell \le q} |\gamma_{j,\ell}|$. For any $m \in \mathbb{N}$, let $(k, L) \in \mathbb{N}^2$ such that $m = k(L-1) \ge \frac{1}{2}\log_2\left(\frac{\log_2 n}{2}\right)$. For a given $\mathrm{M} \ge 1$, let $(s, r) \in \mathbb{N}^2$ satisfying $\overline{\gamma} \le (s\mathrm{M})^{r-1}$. Then, there exists a $\mathbb{R}^q$-valued ReLU NN Φ_n such that $|\Phi_n|_\infty \le \mathrm{M}$ satisfying*

$$\left| \mathcal{R}[\Phi_n]_l(x) - \sum_{|\boldsymbol{\alpha}| \le n} c_{\boldsymbol{\alpha},l}(x - x_l)^{\boldsymbol{\alpha}} \right| \le \overline{\gamma} \cdot \mathfrak{n} \cdot \binom{d+n}{n} \cdot 2^{-2(m+1)},$$

for all $x \in [-1/2, 1/2]^d$ and for all $l \in \{1, \cdots, q\}$. The architecture of Φ_n is given by

$$\vec{\boldsymbol{n}}_{\Phi_n} = \left(P_{d,n} \cdot [2^{\mathfrak{n}-1}, \ldots, 2^0] \otimes \vec{n}_{\tilde{\times}}, (2s)^{\oplus(r-1)} \right)$$
$$\textit{where} \quad \vec{n}_{\tilde{\times}} = (3(2^k+2))^{\oplus L}.$$

Proof. Let $x, x_\ell \in \mathbb{R}^d$. By applying the binomial theorem, we have

$$(x - x_\ell)^{\boldsymbol{\alpha}} = \sum_{0 \le \mathbf{j} \le \boldsymbol{\alpha}} \binom{\boldsymbol{\alpha}}{\mathbf{j}} (-x_\ell)^{\boldsymbol{\alpha}-\mathbf{j}} (x)^{\mathbf{j}}, \qquad \binom{\boldsymbol{\alpha}}{\mathbf{j}} = \prod_{i=1}^d \binom{\alpha_i}{j_i}.$$

Thus, we have

$$\begin{aligned}\sum_{|\boldsymbol{\alpha}| \le n} c_{\boldsymbol{\alpha},\ell}(x - x_\ell)^{\boldsymbol{\alpha}} &= \sum_{|\boldsymbol{\alpha}| \le n} c_{\boldsymbol{\alpha},\ell} \sum_{0 \le \mathbf{j} \le \boldsymbol{\alpha}} \binom{\boldsymbol{\alpha}}{\mathbf{j}} (-x_\ell)^{\boldsymbol{\alpha}-\mathbf{j}} (x)^{\mathbf{j}} \\ &= \sum_{|\mathbf{j}| \le n} (x)^{\mathbf{j}} \left[\sum_{\mathbf{j} \le \boldsymbol{\alpha}, |\boldsymbol{\alpha}| \le n} c_{\boldsymbol{\alpha},\ell} \binom{\boldsymbol{\alpha}}{\mathbf{j}} (-x_\ell)^{\boldsymbol{\alpha}-\mathbf{j}} \right].\end{aligned}$$

Let $\gamma_{\mathbf{j},\ell} = \sum_{\mathbf{j} \le \boldsymbol{\alpha}, |\boldsymbol{\alpha}| \le n} c_{\boldsymbol{\alpha},\ell} \binom{\boldsymbol{\alpha}}{\mathbf{j}} (-x_\ell)^{\boldsymbol{\alpha}-\mathbf{j}}$ and let $\overline{\gamma} := \max_{\mathbf{j} \le n, 1 \le \ell \le q} |\gamma_{\mathbf{j},\ell}|$. Let $s, r \in \mathbb{N}$ satisfy $\overline{\gamma} \le (s\mathrm{M})^{r-1}$.

Let $\{\boldsymbol{\alpha} : 1 \le |\boldsymbol{\alpha}| \le n\} = \{\boldsymbol{\alpha}_1, \cdots, \boldsymbol{\alpha}_{P'_{d,n}}\}$ where $P'_{d,n} = P_{d,n} - 1$ and $P_{d,n} = \binom{d+n}{n}$, and let $\mathfrak{n} = \lceil \log_2 n \rceil$. Let $\Theta_{\boldsymbol{\alpha}}$ be the network from Lemma 6.A.12 such that $\mathcal{R}[\Theta_{\boldsymbol{\alpha}}](x) \approx x^{\boldsymbol{\alpha}}$. If $1 \le |\boldsymbol{\alpha}| < n$, let us consider $\Theta^{\text{ext}}_{\boldsymbol{\alpha}} = \Theta_{\boldsymbol{\alpha}} \bullet \text{Id}$ where Id is the identity network such that $\mathcal{R}[\text{Id}_{L,d}](x) = x$ for all $x \in \mathbb{R}^d$ and that makes $\Theta^{\text{ext}}_{\boldsymbol{\alpha}}$ a $(\mathfrak{n}L + 1)$-layer network. Specifically,

$$\vec{\boldsymbol{n}}_{\Theta^{\text{ext}}_{\boldsymbol{\alpha}}} = (d, \vec{\boldsymbol{v}}_{\Theta^{\text{ext}}_{\boldsymbol{\alpha}}}, 1), \qquad \vec{\boldsymbol{v}}_{\Theta^{\text{ext}}_{\boldsymbol{\alpha}}} = (\overbrace{2d, \cdots, 2d}^{(\mathfrak{n} - \lceil \log_2 |\boldsymbol{\alpha}| \rceil)L \text{ times}}, \boldsymbol{v}^{\lceil \log_2 |\boldsymbol{\alpha}| \rceil - 1}_{\tilde{\times}, \boldsymbol{\alpha}}, \cdots, \boldsymbol{v}^{0}_{\tilde{\times}, \boldsymbol{\alpha}}),$$

where $\boldsymbol{v}^{j}_{\tilde{\times}, \boldsymbol{\alpha}}$ is defined in Lemma 6.A.12.

Let $\Theta'_n := P_{jt}(\Theta^{\text{ext}}_{\boldsymbol{\alpha}_j} : 1 \le j \le P'_{d,n})$ be a $(\mathfrak{n}L + 1)$-layer network such that

$$\mathcal{R}[\Theta'_n]_j(x) = \mathcal{R}[\Theta^{\text{ext}}_{\boldsymbol{\alpha}_j}](x), \qquad 1 \le j \le P'_{d,n},$$

and whose architecture is $\vec{\boldsymbol{n}}_{\Theta'_n} = (d, \sum_{j=1}^{P'_{d,n}} \vec{\boldsymbol{v}}_{\Theta^{\text{ext}}_{\boldsymbol{\alpha}_j}}, P'_{d,n})$.

Let $\theta_1 = \{G_c, g_c\}$ be a layer such that

$$G_c \in \mathbb{R}^{q \times P'_{d,n}}, \quad g_c \in \mathbb{R}^q,$$
$$\text{where} \quad [G_c]_{\ell j} = \frac{1}{\overline{\gamma}} \gamma_{\boldsymbol{\alpha}_j, \ell}, [g_c]_\ell = \frac{1}{\overline{\gamma}} \gamma_{\boldsymbol{\alpha}_0, \ell}, \quad 1 \le \ell \le q, 1 \le j \le P'_{d,n}.$$

Let $\Theta_n = \theta_1 \bullet \Theta'_n$. Then, Θ_n is a network with the architecture of $\vec{\boldsymbol{n}}_{\Theta_n} = (d, \sum_{j=1}^{P'_{d,n}} \vec{\boldsymbol{v}}_{\Theta^{\text{ext}}_{\boldsymbol{\alpha}_j}}, q)$ that satisfies

$$\begin{aligned} |\mathcal{R}[\Theta_n]_l(x) - \sum_{|\boldsymbol{\alpha}| \le n} c_{\boldsymbol{\alpha}, l} (x - x_l)^{\boldsymbol{\alpha}}| &\le \overline{\gamma} \sum_{|\boldsymbol{\alpha}| \le n} |x^{\boldsymbol{\alpha}} - \mathcal{R}[\theta_{\boldsymbol{\alpha}}](x)| \\ &\le \overline{\gamma} \left(\sum_{k=1}^{\mathfrak{n}} k |\{\boldsymbol{\alpha} : |\boldsymbol{\alpha}| = k\}| \right) \cdot 2^{-2(m+1)} \le \overline{\gamma} \mathfrak{n} \binom{d+n}{n} 2^{-2(m+1)}. \end{aligned}$$ □

Lemma 6.A.14 (Lemma A.4 in Petersen and Voigtlaender, 2018). *Let $n \in \mathbb{N}_0$ and $\beta = n + \sigma$ where $\sigma \in (0, 1]$. For each $f \in \mathcal{F}_{\beta, d, B}$ and $x_0 \in (-1/2, 1/2)^d$, there exists a polynomial $p(x) = \sum_{|\boldsymbol{k}| \le n} c_{\boldsymbol{k}} (x - x_0)^{\boldsymbol{k}}$ with $|c_{\boldsymbol{k}}| \le \frac{B}{\boldsymbol{k}!}$ for all $|\boldsymbol{k}| \le n$, and such that*

$$|f(x) - p(x)| \le \frac{d^n}{n!} B \|x - x_0\|^\beta, \quad \forall x \in [-1/2, 1/2]^d.$$

Proof. See Petersen and Voigtlaender (2018). □

Lemma 6.A.15 (Truncation). *For $B > 0$, there exists a two-layer network $\theta = \{W^l, b^l\}_{l=1}^2$ with $|\theta|_\infty \le 1$ and $\vec{\boldsymbol{n}}_\theta = (d, 2d, d)$ that exactly represents the truncation function $\tau_B(x) = sign(x) \min\{|x|, B\}$, i.e., $\max\{1, B\} \mathcal{R}_k[\theta](x) = \tau_B(x_k)$ for all $1 \le k \le d$.*

Proof. Let $A = [1, -1]$, $1_{2\times 1} = [1, 1]^T$ and consider the following network $\theta = \{W^l, b^l\}_{l=1}^2$ such that

$$W^1 = \min\{1, B^{-1}\} I_d \otimes 1_{2\times 1} \in \mathbb{R}^{2d\times d}, \quad b^1 = \min\{1, B\} 1_{d\times 1} \otimes A^T \in \mathbb{R}^{2d\times 1},$$
$$W^2 = I_d \otimes A \in \mathbb{R}^{d\times 2d}, \quad b^2 = -\min\{1, B\} 1_{d\times 1} \mathbb{R}^{d\times 1}.$$

It then can be checked that $\max\{1, B\}\mathcal{R}[\theta]_l(x, B) = \tau_B(x_l)$ for all $1 \le l \le d$ and $|\theta|_\infty \le 1$. □

Appendix 6.B Approximation of piecewise polynomials

Theorem 6.B.1. *Let $d \in \mathbb{N}$, $\beta = n + \sigma$ where $\sigma \in (0, 1]$. Let $\mathfrak{n} = \lceil \log_2 n \rceil$. Let $\mathrm{M} \ge 1$ be given. The following seven integers $(k, L, s, s', r, r', r'')$ decide the specific architecture that depends on the approximation accuracy (controlled by m, the magnitude M of weights and biases.*

- *For any $m \in \mathbb{N}$, choose $(k, L) \in \mathbb{N}^2$ such that $m = k(L-1) \ge \frac{1}{2}\log_2\left(\frac{\log_2 n}{2}\right)$.*
- *For any $(s, r) \in \mathbb{N}^2$, let $\tilde{\gamma}(d, n, B) \le (s\mathrm{M})^{r-1}$ where $\tilde{\gamma}$ is defined in (6.B.1).*
- *For any $(s', r', r'') \in \mathbb{N}^3$, let $2^{2(m+1)}\mathrm{M}^{-1} \le 0.5(s'\mathrm{M})^{r'-1}$ and $(1 + \frac{d^{n+\beta}}{n!})B \le ((ds+1)\mathrm{M})^{r''-1}$.*

Then, there exists a ReLU NN Φ such that $|\Psi|_\infty \le \mathrm{M}$ and for any $f \in \mathcal{F}_{\beta,d,B}$, there are network parameters that depend on f satisfying

$$\|f(x) - \mathcal{R}[\Psi](x)\|_{L_p([-1/2,1/2]^d)} \le C'(d, n, \beta, B) 2^{-\frac{\beta}{d+\beta}(2m+1)},$$

where $C'(d, n, \beta, B) = 2\max\{\max\{d, \mathfrak{n}\}\max\{(1 + \frac{d^n}{n!}d^\beta)B, \tilde{\gamma}(d, n, B)P_{d,n}\}, \frac{d^{n+\beta}}{n!}B\}$. The architecture of Φ is given by

$$\vec{n}_\Phi = (\vec{n}_{\Theta_0}, 2(N^d + d), (2N^d(ds'+1))^{\oplus(r'+r'')}), \qquad N^d = 2^{\frac{d}{d+\beta}(2m+1)},$$

where $P_{d,n} = \binom{d+n}{n}$ and

$$\vec{\boldsymbol{n}}_{\Phi_n} = \left(P_{d,n} \cdot [2^{\mathfrak{n}-1}, \ldots, 2^0] \otimes \vec{n}_{\tilde{\times}}, (2s)^{\oplus(r-1)}\right) + (2d)^{\oplus(\mathfrak{n}L+r-1)},$$
$$\textit{where} \quad \vec{n}_{\tilde{\times}} = (3(2^k + 2))^{\oplus L}.$$

Proof. Let $N \in \mathbb{N}$ which will be chosen later. For $\lambda \in \{1, \cdots, N\}^d =: [N]^d$, let us consider a partition of $[-1/2, 1/2]^d$ (with disjointness up to null sets):

$$I_\lambda := \prod_{i=1}^d \left[\frac{\lambda_i - 1}{N} - \frac{1}{2}, \frac{\lambda_i}{N} - \frac{1}{2}\right], \qquad \bigcup_{\lambda \in [N]^d} I_\lambda = [-1/2, 1/2]^d,$$
$$x_\lambda = [x_\lambda]_i = (\frac{\lambda_i - \frac{1}{2}}{N} - \frac{1}{2}).$$

Note that $I_\lambda \subset \bar{B}^{|\cdot|}_{d/N}(x)$ for all $x \in I_\lambda$. It then follows from Lemma 6.A.14 that for each λ, there exists a polynomial $p_{\lambda,n}$ of degree up to n such that

$$\|f - p_{\lambda,n}\|_{C^0(I_\lambda)} \le \frac{d^n}{n!} B \left(\frac{d}{N}\right)^\beta \implies$$
$$\|f - p_{\lambda,n}\|_{L_p(I_\lambda)} \le N^{-d/p} \frac{d^n}{n!} B \left(\frac{d}{N}\right)^\beta .$$

Here we use $\|f\|_{L_p(\Omega)} \le \mu(\Omega)^{1/p}\|f\|_{C^0}$, where $\mu(\Omega)$ is the Lebesgue measure of Ω. Then, we have

$$\left\| f(x) - \sum_{\lambda\in[N]^d} \mathbb{I}_{I_\lambda}(x)\cdot p_{\lambda,n}(x)\right\|_{L_p([-1/2,1/2]^d)}$$
$$= \left(\sum_{\lambda\in[N]^d} \|f - p_{\lambda,n}\|^p_{L_p(I_\lambda)}\right)^{1/p} \le N^{-\beta}\frac{d^{n+\beta}}{n!}B.$$

Also, since $f \in \mathcal{F}_{\beta,d,B}$, $\|f\|_{C^0} \le B$ and thus, it follows from Lemma 6.A.14 that

$$\|p_{\lambda,n}\|_{C^0(I_\lambda)} \le (1 + \frac{d^n}{n!}d^\beta)B := \tilde{B}(d,\beta,B), \qquad \forall\lambda \in [N]^d,$$

where $\tilde{B}$ depends only on d, β and B. Let $x_\lambda = [x_\lambda]_i = (\frac{\lambda_i - \frac{1}{2}}{N} - \frac{1}{2})$ for $\lambda \in [N]^d$. Here $p_{\lambda,n}$ is the Taylor polynomial of degree n centered at x_λ, which can be written as

$$p_{\lambda,n} = \sum_{|\boldsymbol{\alpha}|\le n} c_{\boldsymbol{\alpha},\lambda}(x - x_\lambda)^{\boldsymbol{\alpha}} = \sum_{|\mathbf{k}|\le n} \gamma_{\mathbf{k}}(\mathbf{C}_\lambda, x_\lambda)x^{\mathbf{k}}, \qquad c_{\boldsymbol{\alpha},\lambda} = \frac{\partial^{\boldsymbol{\alpha}} f(x_\lambda)}{\boldsymbol{\alpha}!},$$

where $\gamma_{\mathbf{k}}(\mathbf{C}_\lambda, x_\lambda) = \sum_{\mathbf{k}\le\boldsymbol{\alpha},|\boldsymbol{\alpha}|\le n} \frac{c^f_{\boldsymbol{\alpha},\lambda}}{\boldsymbol{\alpha}!}\binom{\boldsymbol{\alpha}}{\mathbf{k}}(-x_\lambda)^{\boldsymbol{\alpha}-\mathbf{k}}$ and $\mathbf{C}_\lambda = \{c_{\boldsymbol{\alpha},\lambda}\}_{|\boldsymbol{\alpha}|\le n} \subset [-B,B]^{P_{d,n}}$. Note that for any $f \in \mathcal{F}_{\beta,d,B}$, $|c_{\boldsymbol{\alpha},\lambda}| \le B$ for any λ and $\boldsymbol{\alpha}$ such that $|\boldsymbol{\alpha}| \le n$. Let

$$\tilde{\gamma}(d,n,B) := \max_{\mathbf{C}\subset[-B,B]^{P_{d,n}}} \max_{|\mathbf{k}|\le n, x\in[-1/2,1/2]^d} |\gamma_{\mathbf{k}}(\mathbf{C},x)|. \tag{6.B.1}$$

For any $m \in \mathbb{N}$, let $(k, L) \in \mathbb{N}^2$ such that $m = k(L-1) \ge \frac{1}{2}\log_2\left(\frac{\log_2 n}{2}\right)$ where $\mathfrak{n} = \lceil \log_2 n \rceil$. Also, let $s, r \in \mathbb{N}^2$ satisfy $\tilde{\gamma}(d,n,B) \le (s\mathrm{M})^{r-1}$. By Lemma 6.A.13, we have a $\mathbb{R}^{N^d+d}$-valued ReLU NN Θ_n that approximates $p_{\lambda,n}$ for all $\lambda \in [N]^d$ such that

$$\|\mathcal{R}[\Theta_n]_\lambda(x) - p_{\lambda,n}(x)\|_{L^\infty(I_\lambda)} \le \tilde{\gamma}(d,n,B)\cdot P_{d,n}\cdot \mathfrak{n}\cdot 2^{-2(m+1)}, \quad \forall\lambda \in [N]^d,$$

and $\mathcal{R}[\Theta_n]_\lambda(x) = x_j$ for $\lambda = N^d + j$ and $1 \le j \le d$, whose architecture is

$$\vec{\boldsymbol{n}}_{\Phi_n} = \left(P_{d,n} \cdot [2^{\mathfrak{n}-1}, \ldots, 2^0] \otimes \vec{n}_{\tilde{\times}}, (2s)^{\oplus(r-1)}\right) + (2d)^{\oplus(\mathfrak{n}L+r-1)},$$
$$\text{where} \quad \vec{n}_{\tilde{\times}} = (3(2^k+2))^{\oplus L}.$$

Let τ be the separate concatenation of the truncation network from Lemma 6.A.15 and the identity network. That is, let $\Theta_n' := \tau_{\tilde{B}} \bullet \Theta_n$. Then, $\mathcal{R}[\Theta_n']_l(x) = \tau_{\tilde{B}}(\mathcal{R}[\Theta_n]_l(x))$ for $1 \le l \in N^d$, and $\mathcal{R}[\Theta_n']_l(x) = x_j$ for $l = N^d + j$ and $1 \le j \le d$. Its architecture is

$$\vec{\boldsymbol{n}}_{\Theta_n'} = (\vec{\boldsymbol{n}}_{\Phi_n}, 2(N^d + d)),$$

and $|\Theta_n'|_\infty \le \mathrm{M}$.

We now apply Lemma 6.A.9 with Θ_n'. Let $\epsilon < \frac{1}{2N}$. Let $(s', r', r'') \in \mathbb{N} \times \mathbb{N}_{\ge 2}^2$ such that

$$\epsilon^{-1}\mathrm{M}^{-1} \le 0.5(s'\mathrm{M})^{r'-1},$$
$$(1 + \frac{d^n}{n!} d^\beta) B = \tilde{B} \le ((ds+1)\mathrm{M})^{r''-1}.$$

Then, there exists a ReLU NN Φ such that $|\Phi|_\infty \le \mathrm{M}$, $\vec{n}_\Phi = (\vec{n}_{\Theta_0'}, (2N^d(ds'+1))^{\oplus(r'+r'')})$ and

$$\left\| \mathcal{R}[\Phi](x) - \sum_{\lambda \in [N]^d} \mathbb{I}_{I_\lambda}(x) \mathcal{R}[\Theta_n']_\lambda(x) \right\|_{L_p([-1/2,1/2]^d)} \le 4d\tilde{B}N^d\epsilon.$$

It then can be checked that

$$\begin{aligned}
&\left\| \mathcal{R}[\Phi](x) - \sum_{\lambda \in [N]^d} \mathbb{I}_{I_\lambda}(x) \cdot p_{\lambda,n}(x) \right\|_{L_p([-1/2,1/2]^d)} \\
&\le \sum_{\lambda \in [N]^d} \left\| n_\epsilon(x, \tilde{p}_{\lambda,n}(x)) - \mathbb{I}_{I_\lambda}(x) \cdot p_{\lambda,n}(x) \right\|_{L_p(I_\lambda)} \\
&\le \sum_{\lambda \in [N]^d} \Big\{ \left\| n_\epsilon(x, p_{\lambda,n}(x)) - \mathbb{I}_{I_\lambda}(x) \cdot p_{\lambda,n}(x) \right\|_{L_p(I_\lambda)} \\
&\quad + \left\| n_\epsilon(x, \tilde{p}_{\lambda,n}(x)) - n_\epsilon(x, p_{\lambda,n}(x)) \right\|_{L_p(I_\lambda)} \Big\} \\
&\le N^d d\tilde{B}\epsilon + \sum_{\lambda \in [N]^d} \left\| \tilde{p}_{\lambda,n}(x) - p_{\lambda,n}(x) \right\|_{L_p(I_\lambda)} \\
&\le N^d d\tilde{B}\epsilon + N^d \tilde{\gamma}(d,n,B) \cdot P_{d,n} \cdot \mathfrak{n} \cdot 2^{-2(m+1)} \\
&\le N^d C(d,n,B)\{d\epsilon + \mathfrak{n}2^{-2(m+1)}\},
\end{aligned}$$

where $C(d,n,B) = \max\{\tilde{B}(d,n,B), \tilde{\gamma}(d,n,B)P_{d,n}\}$. Therefore, we have

$$\begin{aligned}
&\|f(x) - \mathcal{R}[\Phi](x)\|_{L_p([-1/2,1/2]^d)} \\
&\leq \left\| \mathcal{R}[\Phi](x) - \sum_{\lambda \in [N]^d} \mathbb{I}_{I_\lambda}(x) \cdot p_{\lambda,n}(x) \right\|_{L_p([-1/2,1/2]^d)} \\
&\quad + \left\| f(x) - \sum_{\lambda \in [N]^d} \mathbb{I}_{I_\lambda}(x) \cdot p_{\lambda,n}(x) \right\|_{L_p([-1/2,1/2]^d)} \\
&\leq N^d C(d,n,B)\{d\epsilon + \mathfrak{n}2^{-2(m+1)}\} + N^{-\beta}\frac{d^{n+\beta}}{n!}B \\
&\leq C'(d,n,\beta,B)((\epsilon + 2^{-2(m+1)})N^d + N^{-\beta}),
\end{aligned}$$

where $C'(d,n,\beta,B) = \max\{\max\{d,\mathfrak{n}\}C(d,n,B), \frac{d^{n+\beta}}{n!}B\}$.

By letting $N = 2^{s''}$ where $s'' = \frac{2m+1}{d+\beta}$ and $\epsilon = 2^{-2(m+1)}$, we have

$$\begin{aligned}
&\|f(x) - \mathcal{R}[\Psi](x)\|_{L_p([-1/2,1/2]^d)} \\
&\leq C'(d,n,\beta,B)((\epsilon + 2^{-2(m+1)})2^{ds'} + 2^{-\beta s'}) \\
&= 2C'(d,n,\beta,B)2^{-\frac{\beta}{d+\beta}(2m+1)}.
\end{aligned}$$

□

Appendix 6.C Approximation of horizon functions

Lemma 6.C.1. *Let* $\mathrm{M} \geq 1$ *be given. Let* $\gamma \in \mathcal{F}_{\beta,d-1,B}$ *and* θ_γ *be a ReLU network that approximates* γ *from Theorem 6.B.1 with* $|\theta_\gamma|_\infty \leq \mathrm{M}$*. For any* $\epsilon \in (0,1]$*, let* $(s_h, r_h) \in \mathbb{N}^2$ *such that* $\epsilon^{-1}\mathrm{M}^{-1} \leq (s_h\mathrm{M})^{r_h}$*. Then, there exists a network* Φ *such that*

$$\begin{aligned}
&\|\mathcal{R}[\Phi](x) - \mathbb{I}_{[0,\infty)}(x_1 + \gamma(x_{-1}))\|_{L^p([-1/2,1/2]^d)} \\
&\leq 2^{\frac{1+p}{p}} \max\left\{ \|\gamma - \mathcal{R}[\theta_\gamma]\|_{L^1([-1/2,1/2]^{d-1})}^{\frac{1}{p}}, \epsilon^{\frac{1}{p}} \right\},
\end{aligned}$$

with $\vec{n}_\Phi = (\vec{n}_{\theta_\gamma} + 2^{\oplus L_{\theta_\gamma}}, (2s_h)^{\oplus r_h})$ *and* $|\Phi|_\infty \leq \mathrm{M}$.

Proof. For $\gamma \in \mathcal{F}_{\beta,d-1,B}$, let θ_γ be a ReLU NN from Theorem 6.B.1 that approximates γ. Since the permutation matrix does not change the accuracy of the resulting networks, without loss of generality (up to a constant), we assume $T = I_d$.

Let $x = (x_1, \cdots, x_d)$ and $x_{-1} = (x_2, \cdots, x_d)$. After modifying the first hidden layer of θ_γ, we obtain a NN, θ'_γ such that $\mathcal{R}[\theta'_\gamma](x) = x_1 + \mathcal{R}[\theta_\gamma](x_{-1})$ with $|\theta'_\gamma|_\infty = |\theta_\gamma|_\infty$ and $\vec{n}_{\theta'_\gamma} = \vec{n}_{\theta_\gamma} + 2^{\oplus L_{\theta_\gamma}}$ where L_{θ_γ} is the size of $\vec{n}_{\theta_\gamma}$.

For any $\epsilon \in (0, 1]$, let $(s_h, r_h) \in \mathbb{N}^2$ such that $\epsilon^{-1}\mathrm{M}^{-1} \leq (s\mathrm{M})^r$. Then, there is a NN θ_H such that $\|\mathcal{R}[\theta_H] - \mathbb{I}_{[0,\infty)}\|_{L_p} \leq \epsilon^{-1/p}$.

Let $\Phi = \theta_H \bullet \theta'_\gamma$ with $\vec{n}_\Phi = (\vec{n}_{\theta'_\gamma}, \vec{n}_{\theta_H})$ and $|\Phi|_\infty \leq \mathrm{M}$. It can be checked that

$$\|\mathcal{R}[\Phi](x) - \mathbb{I}_{[0,\infty)}(x_1 + \gamma(x_{-1}))\|_{L^p([-1/2,1/2]^d)} \leq 2^q \max\{E_I, E_{II}\},$$

where $q = 1 + p^{-1}$ and

$$\begin{aligned} E_I &= \|\mathbb{I}_{[0,\infty)}(\mathcal{R}[\theta'_\gamma](x)) - \mathbb{I}_{[0,\infty)}(x_1 + \gamma(x_{-1}))\|_{L^p([-1/2,1/2]^d)}, \\ E_{II} &= \|\mathcal{R}[\theta_H](\mathcal{R}[\theta'_\gamma](x)) - \mathbb{I}_{[0,\infty)}(\mathcal{R}[\theta'_\gamma](x))\|_{L^p([-1/2,1/2]^d)}. \end{aligned}$$

First, we note that for fixed $x_{-1} \in [-1/2, 1/2]^{d-1}$,

$$\begin{aligned} & x_1 + \gamma(x_{-1}) \geq 0 \text{ and } x_1 + \mathcal{R}[\theta'_\gamma](\tilde{x}) < 0 \\ & \iff x_1 \in [-\gamma(x_{-1}), -\mathcal{R}[\theta'_\gamma](x_{-1})), \\ & x_1 + \gamma(x_{-1}) < 0 \text{ and } x_1 + \mathcal{R}[\theta'_\gamma](x_{-1}) \geq 0 \\ & \iff x_1 \in [-\mathcal{R}[\theta'_\gamma](x_{-1}), -\gamma(x_{-1})). \end{aligned}$$

Thus, it then can be seen that

$$(2^q E_I)^p \leq 2^{1+p}\|\gamma - \mathcal{R}[\theta_\gamma]\|_{L^1([-1/2,1/2]^{d-1})}.$$

For E_{II}, since $|\mathbb{I}_{[0,\infty)}(x_1) - \mathcal{R}[\theta_H](x_1)| \leq \mathbb{I}_{[0,\epsilon]}(x_1)$ for all $x_1 \in \mathbb{R}$, we have

$$\begin{aligned} (2^q E_{II})^p &\leq 2^{1+p} \int_{[-1/2,1/2]^{d-1}} \int_{[-1/2,1/2]} \mathbb{I}_{0 \leq x_1 + \mathcal{R}[\theta_\gamma](x_{-1}) \leq \epsilon}(x) dx_1 dx_{-1} \\ &\leq 2^{1+p} \int_{[-1/2,1/2]^{d-1}} \epsilon dx_{-1} = 2^{1+p}\epsilon. \end{aligned}$$

Therefore,

$$\begin{aligned} & \|\mathcal{R}[\Phi](x) - \mathbb{I}_{[0,\infty)}(x_1 + \gamma(\tilde{x}))\|_{L^p([-1/2,1/2]^d)} \\ & \leq 2^{\frac{1+p}{p}} \max\left\{\|\gamma - \mathcal{R}[\theta_\gamma]\|_{L^1([-1/2,1/2]^{d-1})}^{\frac{1}{p}}, \epsilon^{\frac{1}{p}}\right\}. \end{aligned}$$ □

Theorem 6.C.2. *Let $d \in \mathbb{N}$, $\beta = n + \sigma$ where $\sigma \in (0, 1]$, $r \in \mathbb{N}$ and $\mathrm{M} \geq 1$ be given. Let $\mathfrak{n} = \lceil \log_2 n \rceil$. Let $m \in \mathbb{N}$ such that $(2m + 1) \geq (r + 1)^{\frac{p(d-1+\beta)}{\beta}}$.*

- *Let $(k, L) \in \mathbb{N}^2$ such that $m = k(L - 1) \geq \frac{1}{2}\log_2\left(\frac{\log_2 n}{2}\right)$.*
- *Let $(s', r') \in \mathbb{N}^2$ such that $\tilde{\gamma}(d - 1, n, B) \leq (s'\mathrm{M})^{r'-1}$ where $\tilde{\gamma}$ is defined in (6.B.1).*
- *Let $(s'', r'', r''') \in \mathbb{N}^3$ such that $2^{2(m+1)}\mathrm{M}^{-1} \leq 0.5(s''\mathrm{M})^{r''-1}$ and $(1 + \frac{(d-1)^{n+\beta}}{n!})B \leq (((d - 1)s' + 1)\mathrm{M})^{r'''-1}$.*

- *Let* $(s_h, r_h) \in \mathbb{N}^2$ *such that* $2^{\frac{\beta}{d-1+\beta}(2m+1)}\mathrm{M}^{-1} \le 0.5(s_h\mathrm{M})^{r_h}$.
- *Let* $(s_I, r_I) \in \mathbb{N}^2$ *such that* $2^{\frac{\beta}{p(d-1+\beta)}(2m+1)}\mathrm{M}^{-1} \le 0.5(s_I\mathrm{M})^{r_I - 1}$.

For any $K \in \mathcal{K}_{r,\beta,d,B}$*, there exists a ReLU NN,* Φ *such that* $|\Phi|_\infty \le \mathrm{M}$ *and*

$$\|\mathcal{R}[\Psi](x) - \mathbb{I}_K(x)\|_{L_p} \le C(r,\beta,d,B,p)2^{-\frac{\beta}{p(d-1+\beta)}(2m+1)},$$

where $C(r,\beta,d,B,p) = 2^{1+p^{-1}+rd}(4d + 2^{\frac{1+p}{p}}C'(d-1,n,\beta,B)^{1/p})$ *and* C' *is defined in Theorem 6.B.1. The architecture of* Φ *is*

$$\vec{n}_\Phi = (2^{rd}\cdot(\vec{n}_{\theta_\gamma} + 2^{\oplus L_{\theta_\gamma}}, (2s_h)^{\oplus r_h}), (2^{rd+1}(ds_I+1))^{\oplus r_I}),$$

where L_{θ_γ} *is the length of* $\vec{n}_{\theta_\gamma}$*, and*

$$\vec{n}_{\theta_\gamma} = (\vec{n}_{\Theta_0}, 2(N^{d-1}+d-1), (2N^{d-1}((d-1)s'+1))^{\oplus(r'+r'')}),$$
$$N^{d-1} = 2^{\frac{d-1}{d-1+\beta}(2m+1)},$$

where $P_{d,n} = \binom{d+n}{n}$ *and*

$$\vec{n}_{\Theta_0} = \left(P_{d,n}\cdot[2^{\mathfrak{n}-1},\dots,2^0]\otimes\vec{n}_{\tilde{\times}}, (2s)^{\oplus(r-1)}\right) + (2d)^{\oplus(\mathfrak{n}L+r-1)},$$
$$\text{where}\quad \vec{n}_{\tilde{\times}} = (3(2^k+2))^{\oplus L}.$$

Proof. For $\lambda = (\lambda_1,\cdots,\lambda_d) \in \{1,\cdots,2^{r_0}\}^d =: [2^{r_0}]^d$, let us consider a partition of $\Omega = [-1/2, 1/2]^d$ (disjointness upto null sets):

$$I_\lambda = \prod_{i=1}^{d}\left[(\lambda_i - 1)2^{-r_0} - \frac{1}{2}, \lambda_i 2^{-2} - \frac{1}{2}\right]. \tag{6.C.1}$$

Note that for $x \in I_\lambda$, $I_\lambda \subset \overline{B}_{2^{-r_0}}^{\|\cdot\|_{\ell^\infty}}(x)$.

From the definition of $\mathcal{K}_{r_0,\beta,d,B}$, for each $\lambda \in \{1,\cdots,2^{r_0}\}^d$, there is a horizon function $f_\lambda \in \mathcal{HF}_{\beta,d,B}$ such that $\mathbb{I}_{I_\lambda}(x)\mathbb{I}_K(x) = \mathbb{I}_{I_\lambda}(x)f_\lambda(x)$. Thus, for any $K \in \mathcal{K}_{r_0,\beta,d,B}$, we have

$$\mathbb{I}_K(x) = \sum_{\lambda\in[2^{r_0}]^d}\mathbb{I}_{I_\lambda}(x)f_\lambda(x).$$

The goal is to find a deep ReLU network that approximates $\sum_{\lambda\in[2^{r_0}]^d}\mathbb{I}_{I_\lambda}(x)f_\lambda(x)$.

Let $d-1 \in \mathbb{N}$, $\beta = n + \sigma$ where $\sigma \in (0,1]$. Let $\mathfrak{n} = \lceil\log_2 n\rceil$. Let $\mathrm{M} \ge 1$ be given

- For any $m \in \mathbb{N}$, choose $(k, L) \in \mathbb{N}^2$ such that $m = k(L-1) \ge \frac{1}{2}\log_2\left(\frac{\log_2 n}{2}\right)$.

- For any $(s,r) \in \mathbb{N}^2$, let $\tilde{\gamma}(d-1,n,B) \le (s\mathrm{M})^{r-1}$ where $\tilde{\gamma}$ is defined in (6.B.1).
- For any $(s', r', r'') \in \mathbb{N}^3$, let $2^{2(m+1)}\mathrm{M}^{-1} \le 0.5(s'\mathrm{M})^{r'-1}$ and $(1+\frac{(d-1)^{n+\beta}}{n!})B \le (((d-1)s+1)\mathrm{M})^{r''-1}$.

From Theorem 6.B.1, for any $\gamma_\lambda \in \mathcal{F}_{d-1,\beta,B}$, there exists a ReLU network θ_{γ_λ} such that

$$\|\gamma - \mathcal{R}[\theta_{\gamma_\lambda}]\|_{L_p} \le C'(d-1,n,\beta,B)2^{-\frac{\beta}{d-1+\beta}(2m+1)},$$

whose architecture is given by

$$\vec{n}_{\theta_{\gamma_\lambda}} = (\vec{n}_{\Theta_0}, 2(N^{d-1}+d-1), (2N^{d-1}((d-1)s'+1))^{\oplus(r'+r'')}),$$
$$N^{d-1} = 2^{\frac{d-1}{d-1+\beta}(2m+1)},$$

where $P_{d,n} = \binom{d+n}{n}$ and

$$\vec{n}_{\Theta_0} = \left(P_{d,n}\cdot[2^{\mathfrak{n}-1},\ldots,2^0]\otimes\vec{n}_{\tilde{\times}}, (2s)^{\oplus(r-1)}\right) + (2d)^{\oplus(\mathfrak{n}L+r-1)},$$
$$\text{where} \quad \vec{n}_{\tilde{\times}} = (3(2^k+2))^{\oplus L},$$

while $|\theta_{\gamma_\lambda}|_\infty \le \mathrm{M}$. By Lemma 6.C.1, for any $\epsilon \in (0,1]$, let $(s_h, r_h) \in \mathbb{N}^2$ such that $\epsilon^{-1}\mathrm{M}^{-1} \le (s_h\mathrm{M})^{r_h}$. Then, there exists a ReLU NN Θ_{γ_λ} such that $\mathcal{R}[\Theta_{\gamma_\lambda}](x) = \mathcal{R}[\theta_H](x_1 + \mathcal{R}[\theta_{\gamma_\lambda}](x_{-1}))$ whose architecture is given by $\vec{n}_{\Theta_\gamma} := \vec{n}_{\Theta_{\gamma_\lambda}} = (\vec{n}_{\theta_{\gamma_\lambda}} + 2^{\oplus L_{\theta_{\gamma_\lambda}}}, (2s_h)^{\oplus r_h})$ and $|\Phi|_\infty \le \mathrm{M}$.

By jointly concatenating $\{\Theta_{\gamma_\lambda}\}_{\lambda\in[2^r]^d}$, we have a ReLU NN θ_F whose architecture is $\vec{n}_{\theta_F} = 2^{r_0 d}\vec{n}_{\Theta_\gamma}$ satisfying $|\theta_F|_\infty \le \mathrm{M}$ such that $\mathcal{R}[\theta_F]_\lambda(x) = \mathcal{R}[\theta_H](x_1 + \mathcal{R}[\theta_{\gamma_\lambda}](\tilde{x}))$ for $\lambda \in [2^r]^d$.

For $\epsilon' \le \frac{1}{2^{r_0+1}}$, let $(s_I, r_I) \in \mathbb{N}^2$ such that $\epsilon'^{-1}\mathrm{M}^{-1} \le 0.5(s_I\mathrm{M})^{r_I-1}$. By applying Lemma 6.A.9, we have a ReLU network Φ such that

$$\left\|\mathcal{R}[\Phi](\cdot) - \sum_{\lambda\in[2^{r_0}]^d}\mathbb{I}_{I_\lambda}\mathcal{R}[\theta_F]_\lambda(\cdot)\right\|_{L_p([-1/2,1/2]^d)} \le 4\cdot d\cdot 2^{r_0 d}\cdot\epsilon',$$

whose architecture is $\vec{n}_\Phi = (\vec{n}_{\theta_F}, (2^{r_0 d+1}(ds_I+1))^{\oplus r_I}$ and $|\Phi|_\infty \le \mathrm{M}$.

Lastly, observe that

$$\|\mathcal{R}[\Phi](x) - \mathbb{I}_K(x)\|_{L_p}$$
$$\le 2^q\left\|\mathcal{R}[\Phi](x) - \sum_{\lambda\in[2^r]^d}\mathbb{I}_{I_\lambda}(x)\mathcal{R}[\theta_F]_\lambda(x)\right\|_{L_p}$$

$$
\begin{aligned}
&+ 2^q \left\| \sum_{\lambda \in [2^r]^d} \mathbb{I}_{I_\lambda}(x) \left(\mathcal{R}[\theta_F]_\lambda(x) - f_\lambda(x)\right) \right\|_{L_p} \\
&\le 2^{q+r_0 d} \left\{ 4d\epsilon' + \max_{\lambda \in [2^r]^d} \{ \|\mathcal{R}[\theta_F]_\lambda(x) - f_\lambda(x)\|_{L_p} \} \right\} \\
&\le 2^{q+r_0 d} \left\{ 4d\epsilon' + 2^{\frac{1+p}{p}} \max\{ C'(d-1, n, \beta, B)^{1/p} 2^{-\frac{\beta}{p(d-1+\beta)}(2m+1)}, \epsilon^{1/p} \} \right\}.
\end{aligned}
$$

By letting $\epsilon^{1/p} = \epsilon' = 2^{-\frac{\beta}{p(d-1+\beta)}(2m+1)}$ we have

$$
\|\mathcal{R}[\Psi](x) - \mathbb{I}_K(x)\|_{L_p} \le C(r_0, \beta, d, B, p) 2^{-\frac{\beta}{p(d-1+\beta)}(2m+1)},
$$

where $C(r_0, \beta, d, B, p) = 2^{q+r_0 d}(4d + 2^{\frac{1+p}{p}} C'(d-1, n, \beta, B)^{1/p})$ and the proof is completed. □

Appendix 6.D Proof of Theorem 6.1

Proof. For $K \in \mathcal{K}_{r,\beta,d,B}$, let θ_K be a ReLU network θ_K from Theorem 6.C.2 that approximates $\mathbb{I}_K(x)$. For $g \in \mathcal{F}_{\beta',d,B}$, let θ_g be a deep ReLU network from Theorem 6.B.1. Without loss of generality, let us assume that the number of layers of θ_g and θ_K are the same. Let $\tilde{\times}$ be the multiplication network from Lemma 6.A.11. We then define $\Phi = \tilde{\times} \bullet \mathrm{P}_{\mathrm{jt}}(\theta_g, \theta_K)$. It then can be checked that

$$
\begin{aligned}
&\|\mathcal{R}[\Phi](x) - \mathbb{I}_K(x) g(x)\|_{L_p} \\
&\le \|\mathcal{R}[\tilde{\times}](\mathcal{R}[\theta_g](x), \mathcal{R}[\theta_K](x)) - \mathcal{R}[\theta_K](x)\mathcal{R}[\theta_g](x)\|_{L_p} \\
&\qquad + \|\mathcal{R}[\theta_K](x)\mathcal{R}[\theta_g](x) - \mathbb{I}_K(x) g(x)\|_{L_p} \\
&\le 2^{-2m} + \|\mathcal{R}[\theta_K](x) \left(\mathcal{R}[\theta_g](x) - g(x)\right)\|_{L_p} + \|g(x)\left(\mathcal{R}[\theta_K](x) - \mathbb{I}_K(x)\right)\|_{L_p}.
\end{aligned}
$$

Note that $\|g\|_{C^0} \le B$ and $|\mathcal{R}[\theta_K](x)| \le 1$. Thus, it follows from Theorems 6.B.1 and 6.C.2 that

$$
\begin{aligned}
\|\mathcal{R}[\theta_K](x) \left(\mathcal{R}[\theta_g](x) - g(x)\right)\|_{L_p} &\le \|\mathcal{R}[\theta_g](x) - g(x)\|_{L_p} \\
&\le C'(d, n, \beta', B) 2^{-\frac{\beta'}{d+\beta'}(2m+1)}, \\
\|g(x)\left(\mathcal{R}[\theta_K](x) - \mathbb{I}_K(x)\right)\|_{L_p} &\le C(r, \beta, d, B, p) 2^{-\frac{\beta}{p(d-1+\beta)}(2m+1)},
\end{aligned}
$$

which shows that

$$
\begin{aligned}
&\|\mathcal{R}[\Phi](x) - \mathbb{I}_K(x) g(x)\|_{L_p} \\
&\le 3^{-1}\tilde{C}\left[2^{-2m} + 2^{-\frac{\beta'}{d+\beta'}(2m+1)} + 2^{-\frac{\beta}{p(d-1+\beta)}(2m+1)}\right],
\end{aligned}
$$

where $\tilde{C}(r, \beta, d, B, p) = 3\max\{C(r, \beta, d, B, p), C'(d, n, \beta', B), 1\}$. Since $\frac{\beta'}{d+\beta'} = \frac{\beta}{p(d-1+\beta)+\beta} \leq \frac{\beta}{p(d-1+\beta)} \leq 1$, we have

$$\|\mathcal{R}[\Phi](x) - \mathbb{I}_K(x)g(x)\|_{L_p} \leq \tilde{C} \cdot 2^{-\frac{\beta}{p(d-1+\beta)+\beta}(2m+1)}.$$

The architecture of Φ is $\vec{n}_\Phi = (\vec{n}_{\theta_K} + \vec{n}_{\theta_g}, \vec{n}_{\tilde{\times}})$. $\square$

References

Ainsworth, M., Dong, J., 2021. Galerkin neural networks: a framework for approximating variational equations with error control. SIAM Journal on Scientific Computing 43 (4), A2474–A2501.

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

Bochev, P., Gunzburger, M., 1998. Finite element methods of least-squares type. SIAM Review 40 (4), 789–837.

Bramble, J.H., Schatz, A.H., 1970. Rayleigh-Ritz-Galerkin methods for Dirichlet's problem using subspaces without boundary conditions. Communications on Pure and Applied Mathematics 23, 653–675.

Burman, E., Oksanen, L., 2018. Weakly consistent regularisation methods for ill-posed problems. In: Numerical Methods for PDEs. In: SEMA SIMAI Springer Ser., vol. 15. Springer, Cham, pp. 171–202.

Chen, T., 1998. A unified approach for neural network-like approximation of non-linear functionals. Neural Networks 11 (6), 981–983.

Chen, T., Chen, H., 1993. Approximations of continuous functionals by neural networks with application to dynamic systems. IEEE Transactions on Neural Networks 4 (6), 910–918.

Chen, T., Chen, H., 1995a. Approximation capability to functions of several variables, nonlinear functionals, and operators by radial basis function neural networks. IEEE Transactions on Neural Networks 6 (4), 904–910.

Chen, T., Chen, H., 1995b. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. IEEE Transactions on Neural Networks 6 (4), 911–917.

Cho, J., Nam, S., Yang, H., Yun, S.-B., Hong, Y., Park, E., 2022. Separable PINN: mitigating the curse of dimensionality in physics-informed neural networks. arXiv preprint. arXiv: 2211.08761.

Dahmen, W., Monsuur, H., Stevenson, R., 2023. Least squares solvers for ill-posed PDEs that are conditionally stable. ESAIM: Mathematical Modelling and Numerical Analysis 57 (4), 2227–2255.

De Ryck, T., Jagtap, A.D., Mishra, S., 2023. Error estimates for physics-informed neural networks approximating the Navier–Stokes equations. IMA Journal of Numerical Analysis, drac085.

Deng, B., Shin, Y., Lu, L., Zhang, Z., Karniadakis, G.E., 2022. Approximation rates of DeepONets for learning operators arising from advection–diffusion equations. Neural Networks 153, 411–426.

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

Franco, N.R., Manzoni, A., Zunino, P., 2023. Mesh-informed neural networks for operator learning in finite element spaces. Journal of Scientific Computing 97 (2), 35.

Glorot, X., Bengio, Y., 2010. Understanding the Difficulty of Training Deep Feedforward Neural Networks. Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, vol. 9. PMLR, pp. 249–256.

Goswami, S., Bora, A., Yu, Y., Karniadakis, G.E., 2023. Physics-informed deep neural operator networks. In: Machine Learning in Modeling and Simulation: Methods and Applications. Springer, pp. 219–254.

He, K., Zhang, X., Ren, S., Sun, J., 2015. Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In: Proc. IEEE Int. Conf. Comput. Vis, pp. 1026–1034.

Hu, Z., Jagtap, A.D., Karniadakis, G.E., Kawaguchi, K., 2022. When do extended physics-informed neural networks (XPINNs) improve generalization? SIAM Journal on Scientific Computing 44 (5), A3158–A3182.

Hu, Z., Shukla, K., Karniadakis, G.E., Kawaguchi, K., 2023. Tackling the curse of dimensionality with physics-informed neural networks. arXiv preprint. arXiv:2307.12306.

Jagtap, A.D., Karniadakis, G.E., 2020. Extended physics-informed neural networks (XPINNs): a generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. Communications in Computational Physics 28 (5), 2002–2041.

Jagtap, A.D., Kharazmi, E., Karniadakis, G.E., 2020. Conservative physics-informed neural networks on discrete domains for conservation laws: applications to forward and inverse problems. Computer Methods in Applied Mechanics and Engineering 365, 113028.

Jagtap, A.D., Mao, Z., Adams, N., Karniadakis, G.E., 2022a. Physics-informed neural networks for inverse problems in supersonic flows. Journal of Computational Physics 466, 111402.

Jagtap, A.D., Shin, Y., Kawaguchi, K., Karniadakis, G.E., 2022b. Deep Kronecker neural networks: a general framework for neural networks with adaptive activation functions. Neurocomputing 468, 165–180.

Kharazmi, E., Zhang, Z., Karniadakis, G.E., 2019. Variational physics-informed neural networks for solving partial differential equations. arXiv preprint. arXiv:1912.00873.

Kharazmi, E., Zhang, Z., Karniadakis, G.E., 2021. hp-vpinns: variational physics-informed neural networks with domain decomposition. Computer Methods in Applied Mechanics and Engineering 374, 113547.

Khodayi-Mehr, R., Zavlanos, M., 2020. Varnet: variational neural networks for the solution of partial differential equations. In: Learning for Dynamics and Control. PMLR, pp. 298–307.

Kim, H.H., Yang, H.J., 2023. Domain decomposition algorithms for physics-informed neural networks. In: Domain Decomposition Methods in Science and Engineering XXVI. Springer, pp. 697–704.

Kim, Y., Choi, Y., Widemann, D., Zohdi, T., 2022. A fast and accurate physics-informed neural network reduced order model with shallow masked autoencoder. Journal of Computational Physics 451, 110841.

Klibanov, M.V., Yamamoto, M., 2006. Lipschitz stability of an inverse problem for an acoustic equation. Applicable Analysis 85 (5), 515–538.

Kopaničáková, A., Kothari, H., Karniadakis, G.E., Krause, R., 2023. Enhancing training of physics-informed neural networks using domain-decomposition based preconditioning strategies. arXiv preprint. arXiv:2306.17648.

Kovachki, N., Lanthaler, S., Mishra, S., 2021a. On universal approximation and error bounds for Fourier neural operators. Journal of Machine Learning Research 22 (1), 13237–13312.

Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., Anandkumar, A., 2021b. Neural operator: learning maps between function spaces. arXiv preprint. arXiv:2108.08481.

Lagaris, I.E., Likas, A., Fotiadis, D.I., 1998. Artificial neural networks for solving ordinary and partial differential equations. IEEE Transactions on Neural Networks 9 (5), 987–1000.

Lanthaler, S., Mishra, S., Karniadakis, G.E., 2021. Error estimates for DeepOnets: a deep learning framework in infinite dimensions.

Lee, S., Shin, Y., 2023. On the training and generalization of deep operator networks. arXiv preprint. arXiv:2309.01020.

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

Lu, L., Jin, P., Karniadakis, G.E., 2019. Deeponet: learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. arXiv preprint. arXiv:1910.03193.

Lu, L., Jin, P., Pang, G., Zhang, Z., Karniadakis, G.E., 2021a. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. Nature Machine Intelligence 3 (3), 218–229.

Lu, L., Meng, X., Cai, S., Mao, Z., Goswami, S., Zhang, Z., Karniadakis, G.E., 2022. A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data. Computer Methods in Applied Mechanics and Engineering 393, 114778.

Lu, L., Pestourie, R., Yao, W., Wang, Z., Verdugo, F., Johnson, S.G., 2021b. Physics-informed neural networks with hard constraints for inverse design. SIAM Journal on Scientific Computing 43 (6), B1105–B1132.

Lu, L., Shin, Y., Su, Y., Karniadakis, G., 2020. Dying ReLU and initialization: theory and numerical examples. Communications in Computational Physics 28, 1671–1706.

Luo, D., O'Leary-Roseberry, T., Chen, P., Ghattas, O., 2023a. Efficient pde-constrained optimization under high-dimensional uncertainty using derivative-informed neural operators.

Luo, Y., Chen, Y., Zhang, Z., 2023b. Cfdbench: a comprehensive benchmark for machine learning methods in fluid dynamics. arXiv preprint. arXiv:2310.05963.

Marcati, C., Schwab, C., 2023. Exponential convergence of deep operator networks for elliptic partial differential equations. SIAM Journal on Numerical Analysis 61 (3), 1513–1545.

Meng, X., Li, Z., Zhang, D., Karniadakis, G.E., 2020. PPINN: parareal physics-informed neural network for time-dependent PDEs. Computer Methods in Applied Mechanics and Engineering 370, 113250.

Mhaskar, H.N., Poggio, T., 2016. Deep vs. shallow networks: an approximation theory perspective. Analysis and Applications 14 (06), 829–848.

Mishra, S., Molinaro, R., 2022. Estimates on the generalization error of physics-informed neural networks for approximating a class of inverse problems for PDEs. IMA Journal of Numerical Analysis 42 (2), 981–1022.

Mishra, S., Molinaro, R., 2023. Estimates on the generalization error of physics-informed neural networks for approximating PDEs. IMA Journal of Numerical Analysis 42 (1), 1–43.

Penwarden, M., Jagtap, A.D., Zhe, S., Karniadakis, G.E., Kirby, R.M., 2023. A unified scalable framework for causal sweeping strategies for physics-informed neural networks (PINNs) and their temporal decompositions. Journal of Computational Physics 493, 112464.

Petersen, P., Voigtlaender, F., 2018. Optimal approximation of piecewise smooth functions using deep ReLU neural networks. Neural Networks 108, 296–330.

Raissi, M., Perdikaris, P., Karniadakis, G.E., 2019. 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.

Richter-Powell, J., Lipman, Y., Chen, R.T., 2022. Neural conservation laws: a divergence-free perspective. Advances in Neural Information Processing Systems 35, 38075–38088.

Shang, Y., Wang, F., Sun, J., 2022. Deep Petrov-Galerkin method for solving partial differential equations. arXiv preprint. arXiv:2201.12995.

Sheng, H., Yang, C., 2022. PFNN-2: a domain decomposed penalty-free neural network method for solving partial differential equations. Communications in Computational Physics 5, 19–25.

Shin, Y., Karniadakis, G.E., 2020. Trainability of ReLU networks and data-dependent initialization. Journal of Machine Learning for Modeling and Computing 1 (1), 39–74.

Shin, Y., Zhang, Z., Karniadakis, G.E., 2023. Error estimates of residual minimization using neural networks for linear PDEs. Journal of Machine Learning for Modeling and Computing 4 (4).

Shukla, K., Jagtap, A.D., Karniadakis, G.E., 2021. Parallel physics-informed neural networks via domain decomposition. Journal of Computational Physics 447, 110683.

Sirignano, J., Spiliopoulos, K., 2018. DGM: a deep learning algorithm for solving partial differential equations. Journal of Computational Physics 375, 1339–1364.

Son, H., Cho, S.W., Hwang, H.J., 2023. Enhanced physics-informed neural networks with augmented Lagrangian relaxation method (AL-PINNs). Neurocomputing, 126424.

Son, H., Jang, J.W., Han, W.J., Hwang, H.J., 2021. Sobolev training for physics informed neural networks. arXiv preprint. arXiv:2101.08932.

Sukumar, N., Srivastava, A., 2022. Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. Computer Methods in Applied Mechanics and Engineering 389, 114333.

Sun, Q., Xu, X., Yi, H., 2023. Dirichlet-Neumann learning algorithm for solving elliptic interface problems. arXiv preprint. arXiv:2301.07361.

Telgarsky, M., 2017. Neural networks and rational functions. In: International Conference on Machine Learning. PMLR, pp. 3387–3393.

Venturi, S., Casey, T., 2023. Svd perspectives for augmenting deeponet flexibility and interpretability. Computer Methods in Applied Mechanics and Engineering 403, 115718.

Wang, S., Wang, H., Perdikaris, P., 2021. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. Science Advances 7 (40), eabi8605.

Wang, Y., Jin, P., Xie, H., 2022a. Tensor neural network and its numerical integration. arXiv preprint. arXiv:2207.02754.

Wang, Y., Liao, Y., Xie, H., 2022b. Solving Schrödinger equation using tensor neural network. arXiv preprint. arXiv:2209.12572.

Yarotsky, D., 2022. Universal approximations of invariant maps by neural networks. Constructive Approximation 55 (1), 407–474.

Yu, J., Lu, L., Meng, X., Karniadakis, G.E., 2022. Gradient-enhanced physics-informed neural networks for forward and inverse pde problems. Computer Methods in Applied Mechanics and Engineering 393, 114823.

Zang, Y., Bao, G., Ye, X., Zhou, H., 2020. Weak adversarial networks for high-dimensional partial differential equations. Journal of Computational Physics 411, 109409.

Zhang, H., Xu, Y., Liu, Q., Li, Y., 2023a. Deep learning framework for solving Fokker-Planck equations with low-rank separation representation. Engineering Applications of Artificial Intelligence 121, 106036.

Zhang, Z., Wing Tat, L., Schaeffer, H., 2023b. Belnet: basis enhanced learning, a mesh-free neural operator. Proceedings of the Royal Society A 479 (2276), 20230043.