SIAM J. SCI. COMPUT.
Vol. 47, No. 3, pp. C601–C629

THE HARD-CONSTRAINT PINNS FOR INTERFACE OPTIMAL CONTROL PROBLEMS*

MING-CHIH LAI†, YONGCUN SONG‡,
XIAOMING YUAN§, HANGRUI YUE¶, AND TIANYOU ZENG§

Abstract. We show that the physics-informed neural networks (PINNs), in combination with some recently developed discontinuity capturing neural networks, can be applied to solve optimal control problems subject to partial differential equations (PDEs) with interfaces and some control constraints. The resulting algorithm is mesh-free and scalable to different PDEs, and it ensures the control constraints rigorously. Since the boundary and interface conditions, as well as the PDEs, are all treated as soft constraints by lumping them into a weighted loss function, it is necessary to learn them simultaneously and there is no guarantee that the boundary and interface conditions can be satisfied exactly. This immediately causes difficulties in tuning the weights in the corresponding loss function and training the neural networks. To tackle these difficulties and guarantee the numerical accuracy, we propose to impose the boundary and interface conditions as hard constraints in PINNs by developing a novel neural network architecture. The resulting hard-constraint PINNs approach guarantees that both the boundary and the interface conditions can be satisfied exactly or with a high degree of accuracy, and they are decoupled from the learning of the PDEs. Its efficiency is promisingly validated by some elliptic and parabolic interface optimal control problems.

Key words. optimal control, interface problems, physics-informed neural networks, discontinuity capturing neural networks, hard constraints

MSC codes. 49M41, 68T07, 35Q90, 35Q93, 90C25

DOI. 10.1137/23M1601249

1. Introduction.

Partial differential equations (PDEs) with interfaces capture important applications in science and engineering such as fluid mechanics [25], biological science [8], and material science [17]. Typically, PDEs with interfaces are modeled as piecewise-defined PDEs in different regions coupled together with interface conditions, e.g., jumps in solution and flux across the interface, and hence nonsmooth or even discontinuous solutions. Numerical methods for solving PDEs with interfaces have been extensively studied in the literature; see, e.g., [10, 14, 18, 27, 28]. In addition to numerical simulation of PDEs with interfaces, one very often considers how to control them with certain goals. As a result, optimal control problems of PDEs with interfaces (or interface optimal control problems, for short) arise in various fields. To mention a few, see applications in crystal growth [34] and composite materials [56].

*Submitted to the journal's Machine Learning Methods for Scientific Computing section September 13, 2023; accepted for publication (in revised form) January 14, 2025; published electronically May 6, 2025.

https://doi.org/10.1137/23M1601249

Funding: The work of first author was supported by the NSTC of Taiwan (grant 113-2115-M-A49-014-MY3). The work of third author was supported by RGC TRS project T32-707/22-N. The work of forth author was supported by the National Natural Science Foundation of China (grant 12301399) and the Natural Science Foundation of Tianjin (grant 22JCQNJC01120). The work of fifth author was supported by the Hong Kong PhD Fellowship Scheme.

†Department of Applied Mathematics, National Yang Ming Chiao Tung University, Hsinchu, 30010 Taiwan (mclai@math.nctu.edu.tw).

‡Department of Mathematics, City University of Hong Kong, Kowloon, Hong Kong, China (ysong307@gmail.com).

§Department of Mathematics, The University of Hong Kong, Hong Kong, China (xmyuan@hku.hk, logic@connect.hku.hk).

¶School of Mathematical Sciences, Nankai University, Tianjin, 300071 China (yuehangrui@gmail.com).

In this paper, we consider interface optimal control problems that can be abstractly written as

$$\min_{y\in Y, u\in U} \quad J(y,u) \quad \text{s.t.} \quad \mathcal{I}(y,u)=0,\ u\in U_{ad}. \tag{1.1}$$

Above, Y and U are Banach spaces, $J : Y \times U \to \mathbb{R}$ is the objective functional to minimize, and $y \in Y$ and $u \in U$ are the state variable and the control variable, respectively. The operator $\mathcal{I} : Y \times U \to Z$ with Z Banach space defines a PDE with interface. Throughout, we assume that $\mathcal{I}(y, u) = 0$ is well-posed. That is, for each $u \in U_{ad}$, there exists a unique y that solves $\mathcal{I}(y, u) = 0$ and varies continuously with respect to u. The control constraint $u \in U_{ad}$ imposes pointwise boundedness constraints on u with the admissible set U_{ad} a nonempty closed subset of U. Problem (1.1) aims to find an optimal control $u^* \in U_{ad}$, which determines a state y^* through $\mathcal{I}(y^*, u^*) = 0$, such that $J(y, u)$ is minimized by the pair (y^*, u^*).

Problem (1.1) is challenging from both the theoretical analysis and the algorithmic design perspectives. First, solving problem (1.1) entails appropriate discretization schemes due to the presence of interfaces. For instance, direct applications of standard finite element or finite difference methods fail to produce satisfactory solutions because of the difficulty in enforcing the interface conditions into numerical discretization; see, e.g., [1]. Moreover, similar to the typical optimal control problems with PDE constraints studied in [6, 15, 29, 46], the resulting algebraic systems after discretization are high-dimensional and ill-conditioned and hence difficult to be solved. Finally, the presence of the control constraint $u \in U_{ad}$ leads to problem (1.1) being a nonsmooth optimization problem. Consequently, the well-known gradient-type methods like gradient descent methods, conjugate gradient methods, and quasi-Newton methods cannot be applied directly. All these obvious difficulties imply that meticulously designed algorithms are required for solving problem (1.1).

1.1. State-of-the-art. Numerical methods for solving some optimal control problems modeled by (1.1) have been studied in the literature. These methods combine mesh-based numerical discretization schemes and optimization algorithms that can respectively enforce the interface conditions and tackle the nonsmoothness caused by the constraint $u \in U_{ad}$. For the numerical discretization of elliptic interface optimal control problems, we refer the reader to the immersed finite element methods in [43, 55], the interface-unfitted finite element method based on Nitsche's approach in [54], and the interface concentrated finite element method in [49]. Moreover, an immersed finite element method is proposed in [57] for parabolic interface optimal control problems. Although these finite element methods have shown to be effective to some extent, their practical implementation is not easy, especially for interfaces with complex geometries in high-dimensional spaces. Meanwhile, when the shape of the domain is complicated, generating a suitable mesh is even a nontrivial task, which imposes additional difficulty in solving the problems.

Moreover, various optimization methods have been developed in the context of optimal control problems, such as the semismooth Newton methods [15, 48], the inexact Uzawa method [40], the alternating direction method of multipliers (ADMM) [9], and the primal-dual methods [3, 41]. All these optimization methods can be applied to solve (1.1). It is notable that, to implement the above methods, two PDEs with interfaces ($\mathcal{I}(y, u) = 0$ and its adjoint system) or a saddle point problem are usually required to be solved repeatedly. After some proper numerical discretization, such as the aforementioned immersed and interface-unfitted finite element methods, the resulting systems are large-scale and ill-conditioned, and the computation cost

for solving the PDEs with interfaces or the saddle point problem repeatedly could be extremely high in practice.

1.2. Physics-informed neural networks. In the past few years, thanks to the universal approximation property [5, 11, 16] and the great expressivity [36] of deep neural networks (DNNs), some deep learning methods have been proposed to solve various PDEs, such as the physics-informed neural networks (PINNs) [37], the deep Ritz method [7], the deep Galerkin method [39], and the neural Q-learning method [4]. Compared with the traditional numerical methods for PDEs, deep learning methods are usually mesh-free, easy to implement, scalable to different PDE settings, and able to overcome the curse of dimensionality. Among them, PINN methods have become one of the most prominent deep learning methods and have been extensively studied in, e.g., [20, 21, 30, 37]. However, in general, these PINN methods require the smoothness of the solutions to the PDEs, mainly because the activation functions used in a DNN are in general smooth (e.g., the sigmoid function) or at least continuous (e.g., the rectified linear unit (ReLU) function). Consequently, the above PINN methods cannot be directly used to solve PDEs with interfaces whose solutions are only piecewise-smooth.

To overcome the aforementioned difficulty, some PINN methods tailored for PDEs with interfaces are proposed in, e.g., [14, 18, 47, 53], and these methods primarily focused on developing new ways of using DNNs to approximate the underlying nonsmooth or discontinuous solution. In [14], it is suggested to approximate the solution by two neural networks corresponding to the two distinct subdomains determined by the interface, so that the solution remains smooth in each subdomain. A similar idea can also be found in [53]. In this way, the numerical results obtained by PINNs are satisfactory, but one has to train two neural networks, which requires more computational effort. To alleviate this issue, a discontinuity capturing shallow neural network (DCSNN) is proposed in [18]. The DCSNN allows a single neural network to approximate piecewise-smooth functions by augmenting a coordinate variable, which labels different pieces of each subdomain, as a feature input of the neural network. Since the neural network can be shallow, the resulting number of trainable parameters is moderate and thus the neural network is relatively easier to train. Inspired by [18], a cusp-capturing neural network is proposed in [47] to solve elliptic PDEs with interfaces whose solutions are continuous but have discontinuous first-order derivatives on the interfaces. The cusp-capturing neural network contains the absolute value of the zero level set function of the interface as a feature input and can capture the solution cusps (where the derivatives are discontinuous) sharply. Finally, for completeness, we mention that other deep learning methods for solving PDEs with interfaces can be seen in [12, 19, 45, 51] and the references therein.

In addition to solving PDEs, various PINNs for solving optimal control problems of PDEs have been proposed in the literature; see [2, 31, 35, 42]. In [35], the vanilla PINN method [37] is extended to optimal control problems by approximating the control variable with another neural network in addition to the one for the state variable. Then, these two neural networks are simultaneously trained by minimizing a loss function defined by a weighted sum of the objective functional and the residuals of the PDE constraint. Then PINNs with hard constraints are proposed in [31] for solving optimal design problems, where the PDE and additional inequality constraints are treated as hard constraints by an augmented Lagrangian method. In [2], it is suggested to solve an optimal control problem by deriving the first-order optimality system and approximating the control variable, the state variable, and the

corresponding adjoint variable by different DNNs, respectively. Then a stationary point of the optimal control problem can be computed by minimizing a loss function that consists of the residuals of the first-order optimality system. Recently, the ADMM-PINNs algorithmic framework is proposed in [42], which applies to a general class of optimal control problems with nonsmooth objective functionals. It is worth noting that all the above-mentioned PINN methods are designed for only optimal control problems with smooth PDE constraints, and they cannot be directly applied to interface optimal control problems modeled by (1.1). To the best of our knowledge, there is still no literature for studying the application of PINNs on the interface optimal control problems modeled by (1.1).

1.3. Main contributions. Inspired by the great success of PINNs in solving various PDEs and optimal control problems, we develop some PINNs methods in this paper for solving problem (1.1). We first show that following the ideas in [2], the PINN method [37] can be applied to solve the first-order optimality system of problem (1.1) with the variables approximated by DCSNNs. The resulting PINN method is mesh-free and scalable to different PDEs with interfaces and ensures the control constraint $u \in U_{ad}$ rigorously. However, as shown in section 3.1, this method treats the underlying PDEs and the boundary and interface conditions as soft constraints by penalizing them in the loss function with constant penalty parameters. Hence, the boundary and interface conditions cannot be satisfied exactly, and the numerical errors are mainly accumulated on the boundary and the interface as validated by the numerical results in section 4. Moreover, such a soft-constraint PINN method treats the PDE and the boundary and interface conditions together during the training process, and its effectiveness strongly depends on the choices of the weights in the loss function. Typically, there is no established rule or principle to systematically determine the weights, and setting them manually by trial and error is extremely challenging and time-consuming.

To tackle the above issues, we propose the hard-constraint PINNs, where the boundary and interface conditions are imposed as hard constraints and can be treated separately from the PDEs in the training process. In this context, the term "hard constraints" refers to the boundary and interface conditions being able to be satisfied exactly or with a high degree of accuracy by the designed neural networks. For this purpose, we develop a novel neural network architecture by generalizing the DCSNN to approximate the first-order optimality system of (1.1). To be concrete, we first follow the ideas in [31, 38] to modify the output of the neural network to impose the boundary condition. Then, to impose the interface condition as hard constraints, we propose to construct an auxiliary function for the interface as an additional feature input of the neural network. Such an auxiliary function depends on the geometrical property of the interface, and its construction is nontrivial. To address this issue, we elaborate on the methods for constructing appropriate auxiliary functions for interfaces with different geometrical properties. This ensures that the hard-constraint PINNs are highly implementable. Numerical results for different types of interface optimal control problems are reported to validate the effectiveness and flexibility of the hard-constraint PINNs. Finally, we mention that the proposed hard-constraint PINNs can be directly applied to solve $\mathcal{I}(y, u) = 0$ per se since it is involved as a part of problem (1.1) and its first-order optimality system.

1.4. Organization. The rest of the paper is organized as follows. In section 2, for the convenience of further discussion, we specify problem (1.1) as a distributed elliptic interface optimal control problem, where the control arises as a source term

in the model. Then we review some existing results on the DCSNN. In section 3, we first demonstrate the combination of the DCSNN and the PINN methods for solving the distributed elliptic interface optimal control problem and then propose the hard-constraint PINN method to impose the boundary and interface conditions as hard constraints. We test several elliptic optimal control problems in section 4 to validate the efficiency and effectiveness of the proposed hard-constraint PINN method. In section 5, we showcase how to extend the hard-constraint PINN method by an elliptic interface optimal control problem where the control acts on the interface and a distributed parabolic interface optimal control problem. Some related numerical experiments are also presented to validate the effectiveness. Finally, we make some conclusions and comments for future work in section 6.

2. Preliminaries. In this section, we present some preliminaries that will be used throughout the following discussions. First, to impose our ideas clearly, we specify the generic model (1.1) as a distributed elliptic interface optimal control problem and summarize some existing results. We then review the DCSNN proposed in [18] for elliptic PDEs with interfaces.

2.1. A distributed elliptic interface optimal control problem. Let $\Omega \subset \mathbb{R}^d$ $(d = 2, 3)$ be a bounded domain with Lipschitz continuous boundary $\partial\Omega$, and let $\Gamma \subset \Omega$ be an oriented embedded interface, which divides Ω into two nonoverlapping subdomains Ω^- (inside) and Ω^+ (outside) such that $\Omega = \Omega^- \cup \Omega^+ \cup \Gamma$ and $\overline{\Omega^+} \cap \overline{\Omega^-} = \Gamma$; see Figure 1 for an illustration. We consider the following optimal control problem:

$$\min_{y \in L^2(\Omega), u \in L^2(\Omega)} J(y,u) := \frac{1}{2}\int_\Omega (y - y_d)^2 dx + \frac{\alpha}{2}\int_\Omega u^2 dx, \tag{2.1}$$

subject to the state equation

$$-\nabla \cdot (\beta \nabla y) = u + f \text{ in } \Omega\backslash\Gamma,\ [y]_\Gamma = g_0,\ [\beta \partial_{\boldsymbol{n}} y]_\Gamma = g_1 \text{ on } \Gamma,\ y = h_0 \text{ on } \partial\Omega, \tag{2.2}$$

and the control constraint $u \in U_{ad}$ with

$$u \in U_{ad} := \{u \in L^2(\Omega) : u_a(x) \le u(x) \le u_b(x) \text{ a.e. in } \Omega\} \subset L^2(\Omega), \tag{2.3}$$

where $u_a, u_b \in L^2(\Omega)$.

Above, the function $y_d \in L^2(\Omega)$ is the target and the constant $\alpha > 0$ is a regularization parameter. The functions $f \in L^2(\Omega), g_0 \in H^{\frac{1}{2}}(\Gamma), g_1 \in L^2(\Gamma)$, and $h_0 \in H^{\frac{1}{2}}(\partial\Omega)$ are given, and β is a positive piecewise-constant in $\Omega\backslash\Gamma$ such that $\beta = \beta^-$ in Ω^- and $\beta = \beta^+$ in Ω^+. The bracket $[\cdot]_\Gamma$ denotes the jump discontinuity across the interface Γ and is defined by

$$[y]_\Gamma(x) := \lim_{\tilde{x} \to x \text{ in } \Omega^+} y(\tilde{x}) - \lim_{\tilde{x} \to x \text{ in } \Omega^-} y(\tilde{x}) \quad \forall x \in \Gamma.$$

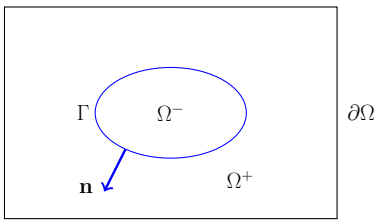


FIG. 1. *The geometry of an interface problem: an illustration.*

The operator $\partial_{\boldsymbol{n}}$ stands for the normal derivative on Γ, i.e., $\partial_{\boldsymbol{n}} y(x) = \boldsymbol{n} \cdot \nabla y(x)$ with $\boldsymbol{n} \in \mathbb{R}^d$ the outward unit normal vector of Γ. In particular, we have

$$[\beta \partial_{\boldsymbol{n}} y]_\Gamma(x) := \beta^+ \lim_{\tilde{x} \to x \text{ in } \Omega^+} \boldsymbol{n} \cdot \nabla y(\tilde{x}) - \beta^- \lim_{\tilde{x} \to x \text{ in } \Omega^-} \boldsymbol{n} \cdot \nabla y(\tilde{x}) \quad \forall x \in \Gamma.$$

Moreover, $y = h_0$ on $\partial\Omega$ is called the *boundary condition,* $[y]_\Gamma = g_0$ on Γ is called the *interface condition,* and $[\beta \partial_{\boldsymbol{n}} y]_\Gamma = g_1$ on Γ is called the *interface-gradient condition.* For (2.1)–(2.3), we have the following results.

THEOREM 2.1 (cf. [55]). *Problem* (2.1)–(2.3) *admits a unique solution* $(u^*, y^*)^\top \in U_{ad} \times L^2(\Omega)$*, and the following first-order optimality system holds:*

$$u^* = \mathcal{P}_{U_{ad}}\Big(-\frac{1}{\alpha} p^* \Big), \tag{2.4}$$

where $\mathcal{P}_{U_{ad}}(\cdot)$ *denotes the projection onto* U_{ad}*, and* p^* *is the adjoint variable associated with* u^**, which is obtained from the successive solution of the following two equations:*

$$-\nabla \cdot (\beta \nabla y^*) = u^* + f \ \text{ in } \Omega\backslash\Gamma,\ [y^*]_\Gamma = g_0,\ [\beta \partial_{\boldsymbol{n}} y^*]_\Gamma = g_1 \ \text{ on } \Gamma,\ y^* = h_0 \ \text{ on } \partial\Omega, \tag{2.5}$$

$$-\nabla \cdot (\beta \nabla p^*) = y^* - y_d \ \text{ in } \Omega\backslash\Gamma,\ [p^*]_\Gamma = 0,\ [\beta \partial_{\boldsymbol{n}} p^*]_\Gamma = 0 \ \text{ on } \Gamma,\ p^* = 0 \ \text{ on } \partial\Omega. \tag{2.6}$$

Since problem (2.1)–(2.3) is known to be convex [55], the optimality conditions (2.4)–(2.6) are also sufficient. Furthermore, substituting (2.4) into (2.5) yields

$$\begin{aligned} -\nabla \cdot (\beta \nabla y^*) &= \mathcal{P}_{U_{ad}}\Big(-\frac{p^*}{\alpha} \Big) + f \text{ in } \Omega\backslash\Gamma,\ [y^*]_\Gamma = g_0,\ [\beta \partial_{\boldsymbol{n}} y^*]_\Gamma = g_1 \text{ on } \Gamma, \\ y^* &= h_0 \text{ on } \partial\Omega. \end{aligned} \tag{2.7}$$

Therefore, solving (2.1)–(2.3) is equivalent to solving (2.6) and (2.7) simultaneously.

2.2. Discontinuity capturing shallow neural networks. In this subsection, we briefly review and explain the idea of the DCSNN [18] using problem (2.2).

First, note that although the solution y to (2.2) is only a d-dimensional piecewise-smooth function, it can be extended to a $(d+1)$-dimensional function $\tilde{y}(x, z)$, which is smooth on the domain $\Omega \times \mathbb{R}$ and satisfies

$$y(x) = \begin{cases} \tilde{y}(x, 1) & \text{if } x \in \Omega^+, \\ \tilde{y}(x, -1) & \text{if } x \in \Omega^-, \end{cases} \tag{2.8}$$

where the additional input $z \in \mathbb{R}$ is the augmented coordinate variable that labels Ω^+ and Ω^-. Note that such a smooth extension $\tilde{y}$ always exists since the function y can be viewed as a smooth function defined on a closed subset of $\mathbb{R}^{d+1}$; see [26]. The extension (2.8) is not unique since there are infinitely many choices of $\tilde{y}(x, z)$ for $z \neq \pm 1$.

Substituting (2.8) into (2.2), it is easy to show that $\tilde{y}(x, z)$ satisfies the following equation:

$$\begin{cases} -\Delta_x \tilde{y}(x, z) = \begin{cases} \dfrac{1}{\beta^+}\left(u(x) + f(x)\right) & \text{if } x \in \Omega^+, z = 1, \\ \dfrac{1}{\beta^-}\left(u(x) + f(x)\right) & \text{if } x \in \Omega^-, z = -1, \end{cases} \\ \tilde{y}(x, 1) - \tilde{y}(x, -1) = g_0(x) \quad \text{if } x \in \Gamma, \\ \beta^+ \boldsymbol{n} \cdot \nabla \tilde{y}(x, 1) - \beta^- \boldsymbol{n} \cdot \nabla \tilde{y}(x, -1) = g_1(x) \quad \text{if } x \in \Gamma, \\ \tilde{y}(x, 1) = h_0(x) \quad \text{if } x \in \partial\Omega. \end{cases} \tag{2.9}$$

Hence, the solution y to problem (2.2) can be obtained from (2.8) with $\tilde{y}(x,z)$ computed by solving (2.9). For solving problem (2.9), we note that the extended function $\tilde{y}$ is smooth and one can construct a neural network $\hat{y}(x,z;\theta)$ with $d+1$ inputs, which is referred to as the DCSNN [18], to approximate $\tilde{y}$. Since $\tilde{y}$ is continuous, it follows from the universal approximation theorem [5] that one can choose $\hat{y}$ as a shallow neural network. Then the PINN method [37] can be applied to solve (2.9) and we refer the reader to [18] for the details.

3. The hard-constraint PINN method for (2.1)–(2.3). In this section, we first demonstrate that, combined with the DCSNN, the PINN method [37] can be applied to solve the reduced optimality conditions (2.6)–(2.7) and hence to solve (2.1)–(2.3). Then we impose the boundary and interface conditions in (2.6)–(2.7) as hard constraints by designing two novel neural networks to approximate y and p, and we propose the hard-constraint PINN method for solving (2.1)–(2.3).

3.1. A soft-constraint PINN method for (2.1)–(2.3). We recall that, for solving (2.1)–(2.3), it is sufficient to solve (2.6) and (2.7) simultaneously. First, we apply two DCSNNs to approximate y and p. To this end, let $\tilde{y}:\Omega\times\mathbb{R}\to\mathbb{R}$ and $\tilde{p}:\Omega\times\mathbb{R}\to\mathbb{R}$ be two smooth extensions of y and p, respectively, which satisfy (2.8) and

$$p(x)=\begin{cases}\tilde{p}(x,1) & \text{if } x\in\Omega^+,\\ \tilde{p}(x,-1) & \text{if } x\in\Omega^-.\end{cases} \tag{3.1}$$

Then, substituting (2.8) and (3.1) into (2.7) and (2.6), we obtain that $\tilde{y}$ and $\tilde{p}$ satisfy the following system:

(3.2)
$$\begin{cases}
-\Delta_x\tilde{y}(x,z)=\begin{cases}\dfrac{1}{\beta^+}\left(f(x)+\mathcal{P}_{[u_a(x),u_b(x)]}\left(-\dfrac{1}{\alpha}\tilde{p}(x,z)\right)\right) & \text{if } x\in\Omega^+, z=1,\\ \dfrac{1}{\beta^-}\left(f(x)+\mathcal{P}_{[u_a(x),u_b(x)]}\left(-\dfrac{1}{\alpha}\tilde{p}(x,z)\right)\right) & \text{if } x\in\Omega^-, z=-1,\end{cases}\\
\tilde{y}(x,1)-\tilde{y}(x,-1)=g_0(x) \quad \text{if } x\in\Gamma,\\
\beta^+\boldsymbol{n}\cdot\nabla_x\tilde{y}(x,1)-\beta^-\boldsymbol{n}\cdot\nabla_x\tilde{y}(x,-1)=g_1(x) \quad \text{if } x\in\Gamma,\\
\tilde{y}(x,1)=h_0(x) \quad \text{if } x\in\partial\Omega,\\
-\Delta_x\tilde{p}(x,z)=\begin{cases}\dfrac{1}{\beta^+}(\tilde{y}(x,z)-y_d(x)) & \text{if } x\in\Omega^+, z=1,\\ \dfrac{1}{\beta^-}(\tilde{y}(x,z)-y_d(x)) & \text{if } x\in\Omega^-, z=-1,\end{cases}\\
\tilde{p}(x,1)-\tilde{p}(x,-1)=0 \quad \text{if } x\in\Gamma,\\
\beta^+\boldsymbol{n}\cdot\nabla_x\tilde{p}(x,1)-\beta^-\boldsymbol{n}\cdot\nabla_x\tilde{p}(x,-1)=0 \quad \text{if } x\in\Gamma,\\
\tilde{p}(x,1)=0 \quad \text{if } x\in\partial\Omega.
\end{cases}$$

Once $\tilde{y}$ and $\tilde{p}$ are computed by solving (3.2), the solutions y and p to (2.6) and (2.7) can be obtained using (2.8) and (3.1). Next, we solve (3.2) by the PINN method [37]. For this purpose, we first sample training sets $\mathcal{T}:=\{(x^i,z^i)\}_{i=1}^M\subset(\Omega^+\times\{1\})\cup(\Omega^-\times\{-1\})$, $\mathcal{T}_B:=\{x_B^i\}_{i=1}^{M_B}\subset\partial\Omega$, and $\mathcal{T}_\Gamma:=\{x_\Gamma^i\}_{i=1}^{M_\Gamma}\subset\Gamma$. We then apply two DCSNNs $\hat{y}(x,z;\theta_y)$ and $\hat{p}(x,z;\theta_p)$ to approximate $\tilde{y}(x,z)$ and $\tilde{p}(x,z)$, respectively, and train the neural networks by minimizing the following loss function:

(3.3)

$$
\begin{aligned}
&\mathcal{L}(\theta_y,\theta_p)\\
&= \frac{w_{y,r}}{M}\sum_{i=1}^{M}\left|-\Delta_x\hat{y}(x^i,z^i;\theta_y)-\frac{\mathcal{P}_{[u_a(x^i),u_b(x^i)]}(-\frac{1}{\alpha}\hat{p}(x^i,z^i;\theta_p))+f(x^i)}{\beta^{\pm}}\right|^2\\
&\quad+\frac{w_{y,b}}{M_b}\sum_{i=1}^{M_b}|\hat{y}(x_B^i,1;\theta_y)-h_0(x_B^i)|^2+\frac{w_{y,\Gamma}}{M_\Gamma}\sum_{i=1}^{M_\Gamma}\left|\hat{y}(x_\Gamma^i,1;\theta_y)-\hat{y}(x_\Gamma^i,-1;\theta_y)-g_0(x_\Gamma^i)\right|^2\\
&\quad+\frac{w_{y,\Gamma_n}}{M_\Gamma}\sum_{i=1}^{M_\Gamma}\left|\beta^+\boldsymbol{n}\cdot\nabla_x\hat{y}(x_i^\Gamma,1;\theta_y)-\beta^-\boldsymbol{n}\cdot\nabla_x\hat{y}(x_i^\Gamma,-1;\theta_y)-g_1(x_\Gamma^i)\right|^2\\
&\quad+\frac{w_{p,r}}{M}\sum_{i=1}^{M}\left|-\Delta_x\hat{p}(x_i,z_i;\theta_p)-\frac{\hat{y}(x_i,z_i;\theta_y)-y_d(x_i)}{\beta^{\pm}}\right|^2+\frac{w_{p,b}}{M_b}\sum_{i=1}^{M_b}|\hat{p}(x_B^i,1;\theta_p)|^2\\
&\quad+\frac{w_{p,\Gamma}}{M_\Gamma}\sum_{i=1}^{M_\Gamma}\left|\hat{p}(x_\Gamma^i,1;\theta_p)-\hat{p}(x_\Gamma^i,-1;\theta_p)\right|^2\\
&\quad+\frac{w_{p,\Gamma_n}}{M_\Gamma}\sum_{i=1}^{M_\Gamma}\left|\beta^+\boldsymbol{n}\cdot\nabla_x\hat{p}(x_i^\Gamma,1;\theta_p)-\beta^-\boldsymbol{n}\cdot\nabla_x\hat{p}(x_i^\Gamma,-1;\theta_p)\right|^2,
\end{aligned}
$$

where $w_{y,*}$ and $w_{p,*}$ are the weights for each term.

Note that the loss function (3.3) is nonnegative, and if $\mathcal{L}(\theta_y,\theta_p)$ goes to zero, then the resulting $(\hat{y},\hat{p})$ gives an approximate solution to (3.2). Moreover, for any function $v:\Omega\to\mathbb{R}$, we have

$$\mathcal{P}_{[u_a(x),u_b(x)]}(v(x))=(L_1\circ L_2\circ v)(x),$$

where $L_1(v(x)) := \mathrm{ReLU}(v(x)-u_a(x))+u_a(x)$ and $L_2(v(x)) := -\mathrm{ReLU}(u_b(x)-v(x))+u_b(x)$ with $\mathrm{ReLU}(v(x))=\max\{v(x),0\}$ the ReLU function. This implies that the projection $\mathcal{P}_{[u_a(x^i),u_b(x^i)]}(-\frac{1}{\alpha}\hat{p}(x^i,z^i;\theta_p))$ can be viewed as a composition of $-\frac{1}{\alpha}\hat{p}(x,z;\theta_p)$ and a two-layer neural network with ReLU as the activation functions. As a result, the loss function $\mathcal{L}(\theta_y,\theta_p)$ in (3.3) can be minimized by a stochastic optimization method with all the derivatives $\Delta_x\hat{y}$, $\nabla_x\hat{y}$, $\Delta_x\hat{p}$, $\nabla_x\hat{p}$ and the gradients $\frac{\partial\mathcal{L}}{\partial\theta_y}$, $\frac{\partial\mathcal{L}}{\partial\theta_p}$ computed by automatic differentiation.

We summarize the above PINN method in Algorithm 3.1.

It is easy to see that the computed control u satisfies the control constraint $u\in U_{ad}$ strictly. Additionally, Algorithm 3.1 is mesh-free and is very flexible in terms of the geometries of the domain and the interface. However, note that in Algorithm 3.1 the boundary and interface conditions are penalized in the loss function (3.3) with constant penalty parameters. Hence, these conditions are treated as soft constraints and cannot be satisfied rigorously by the solutions y and p computed by Algorithm 3.1. Moreover, such a soft-constraint approach treats the PDE and the boundary and interface conditions together during the training process and its effectiveness strongly depends on the choices of the weights in the loss function (3.3). Manually determining these weights through trial and error is extremely challenging and time-demanding. The numerical results in section 4 also show that this soft-constraint approach generates solutions with numerical errors mainly accumulated on the boundaries and the interfaces. To tackle the above issues, we consider imposing the boundary and interface conditions as hard constraints that can be treated separately from the PDE in the training of the neural networks.

Algorithm 3.1. A soft-constraint PINN method for (2.1)–(2.3).

Input: Weights $w_{y,*}, w_{p,*}$ in (3.3).

1: Initialize the neural networks $\hat{y}(x,z;\theta_y)$ and $\hat{p}(x,z;\theta_p)$ with θ_y^0 and θ_p^0.

2: Sample training sets $\mathcal{T} = \{x^i, z^i\}_{i=1}^M \subset \Omega \times \{\pm 1\}$, $\mathcal{T}_B = \{x_B^i\}_{i=1}^{M_B} \subset \partial\Omega$, and $\mathcal{T}_\Gamma = \{x_\Gamma^i\}_{i=1}^{M_\Gamma} \subset \Gamma$.

3: Calculate the values of f, y_d over $\mathcal{T}$, the value of h over $\mathcal{T}_B$, and the value of g_0, g_1 over $\mathcal{T}_\Gamma$.

4: Train the neural networks $\hat{y}(x,z;\theta_y)$ and $\hat{p}(x,z;\theta_p)$ to identify the optimal parameters θ_y^* and θ_p^* by minimizing (3.3).

5: $y(x) \leftarrow \begin{cases} \hat{y}(x,1;\theta_y^*) & \text{if } x \in \Omega^+, \\ \hat{y}(x,-1;\theta_y^*) & \text{if } x \in \Omega^-, \end{cases}$ $p(x) \leftarrow \begin{cases} \hat{p}(x,1;\theta_p^*) & \text{if } x \in \Omega^+, \\ \hat{p}(x,-1;\theta_p^*) & \text{if } x \in \Omega^-, \end{cases}$

$u(x) \leftarrow \mathcal{P}_{[u_a(x),u_b(x)]}\left(-\frac{1}{\alpha}p(x)\right)$.

Output: Approximate solutions $u(x)$ and $y(x)$ to (2.1)–(2.3).

3.2. Hard-constraint boundary and interface conditions. In this subsection, we elaborate on the construction of new neural networks to approximate the state variable y and the adjoint variable p by modifying the DSCNNs $\hat{y}(x,z;\theta_y)$ and $\hat{p}(x,z;\theta_p)$ in Algorithm 3.1 so that the boundary and interface conditions in (2.6) and (2.7) are imposed as hard constraints. In the following discussions, for the sake of simplicity, we still denote θ_y and θ_p the parameters of the neural networks with hard-constraint boundary and interface conditions.

Let $y \in L^2(\Omega)$ be the solution of (2.7); then it satisfies

$$y = h_0 \text{ on } \partial\Omega, \quad [y]_\Gamma = g_0, \quad [\beta\partial_{\boldsymbol{n}} y]_\Gamma = g_1 \text{ on } \Gamma$$

for some functions $g_0, g_1 : \Gamma \to \mathbb{R}$ and $h_0 : \partial\Omega \to \mathbb{R}$. We first introduce two functions $g, h : \overline{\Omega} \to \mathbb{R}$ satisfying

$$g|_{\partial\Omega} = h_0, \quad [g]_\Gamma = g_0, \quad g|_{\Omega^+} \in C^2(\overline{\Omega^+}), \quad g|_{\Omega^-} \in C^2(\overline{\Omega^-}), \tag{3.4}$$

$$h \in C^2(\overline{\Omega}), \quad h(x) = 0 \text{ if and only if } x \in \partial\Omega. \tag{3.5}$$

If the functions g_0 and h_0, the interface Γ, and the boundary $\partial\Omega$ admit analytic forms, it is usually easy to construct g and h with analytic expressions. Some discussions can be found in [23, 24, 31]. Otherwise, we can either adopt the method in [38] or construct g and h by training two neural networks. For instance, we can train a DCSNN $\hat{g}(x,z;\theta_g)$ and a neural network $\hat{h}(x;\theta_h)$ with smooth activation functions (e.g., the sigmoid function or the hyperbolic tangent function) by minimizing the following loss functions:

$$\frac{w_{1g}}{M_b}\sum_{i=1}^{M_b} |\hat{g}(x_B^i, 1;\theta_y) - h_0(x_B^i)|^2 + \frac{w_{2g}}{M_\Gamma}\sum_{i=1}^{M_\Gamma} \left|\hat{g}(x_\Gamma^i, 1;\theta_g) - \hat{g}(x_\Gamma^i, -1;\theta_g) - g_0(x_\Gamma^i)\right|^2, \tag{3.6}$$

$$\frac{w_{1h}}{M_b}\sum_{i=1}^{M_b} |\hat{h}(x_B^i;\theta_h)|^2 + \frac{w_{2h}}{M}\sum_{i=1}^{M} |\hat{h}(x^i;\theta_h) - \bar{h}(x^i)|^2, \tag{3.7}$$

where w_{1g}, w_{2g}, w_{1h}, and $w_{2h} > 0$ are the weights, $\{x^i\}_{i=1}^{M} \subset \Omega$, $\{x_B^i\}_{i=1}^{M_B} \subset \partial\Omega$, and $\{x_\Gamma^i\}_{i=1}^{M_\Gamma} \subset \Gamma$ are the training points, and $\bar{h}(x) \in C^2(\Omega)$ is a known function satisfying $\bar{h}(x) \neq 0$ in Ω, e.g., $\bar{h}(x) = \min_{\hat{x}\in\partial\Omega}\{\|x - \hat{x}\|_2^4\}$.

With the functions g and h satisfying (3.4) and (3.5), we approximate y by

$$\hat{y}(x;\theta_y) = g(x) + h(x)\mathcal{N}_y(x, \phi(x); \theta_y), \tag{3.8}$$

where $\mathcal{N}_y(x, \phi(x); \theta_y)$ is a neural network with smooth activation functions and parameterized by θ_y, and $\phi : \overline{\Omega} \to \mathbb{R}$ satisfying

$$\phi \in C(\overline{\Omega}), \quad \phi|_{\Omega^+} \in C^2(\overline{\Omega^+}), \quad \phi|_{\Omega^-} \in C^2(\overline{\Omega^-}), \quad [\phi]_\Gamma = 0, \quad [\beta\partial_{\boldsymbol{n}}\phi]_\Gamma \neq 0 \text{ a.e. on } \Gamma \tag{3.9}$$

is an auxiliary function for the interface Γ. It follows from (3.5) and (3.9) that $h(x)\mathcal{N}_y(x, \phi(x); \theta_y)$ is a continuous function of x over $\overline{\Omega}$.

For the neural network $\hat{y}(x;\theta_y)$ given by (3.8), it is easy to verify that

$$\begin{aligned}
&[\hat{y}]_\Gamma(x) = [g]_\Gamma(x) + [h(\cdot)\mathcal{N}_y(\cdot, \phi(\cdot))]_\Gamma(x) = g_0(x) \quad \forall x \in \Gamma,\\
&\hat{y}|_{\partial\Omega}(x) = g|_{\partial\Omega}(x) + h|_{\partial\Omega}(x)\left(\mathcal{N}_y(\cdot, \phi(\cdot))|_{\partial\Omega}\right)(x) = h_0(x) \quad \forall x \in \partial\Omega.
\end{aligned}$$

Hence, the interface condition $[y]_\Gamma = g_0$ and the boundary condition $y|_{\partial\Omega} = h_0$ are satisfied exactly by $\hat{y}(x;\theta_y)$ if functions g, h, and ϕ are given in analytic expressions and can be satisfied with a high degree of accuracy if g, h, and ϕ are approximated by pretrained neural networks.

Furthermore, we have

$$\begin{aligned}
[\beta\partial_{\boldsymbol{n}}\hat{y}]_\Gamma(x) &= [\beta\partial_{\boldsymbol{n}}g]_\Gamma(x) + [\beta\partial_{\boldsymbol{n}}h\mathcal{N}_y(\cdot, \phi(\cdot))]_\Gamma(x)\\
&= [\beta\partial_{\boldsymbol{n}}g]_\Gamma(x) + (\beta^+ - \beta^-)\big(\mathcal{N}_y(x, \phi(x))(\boldsymbol{n}\cdot\nabla h(x)) + h(x)(\boldsymbol{n}\cdot\nabla_x\mathcal{N}(x, \phi(x)))\big)\\
&\quad + \frac{\partial\mathcal{N}_y}{\partial\phi}\left(h(x)[\beta\partial_{\boldsymbol{n}}\phi]_\Gamma(x)\right) \quad \forall x \in \Gamma,
\end{aligned} \tag{3.10}$$

which implies that the interface-gradient condition $[\beta\partial_{\boldsymbol{n}}y]_\Gamma = g_1$ cannot be exactly satisfied by $\hat{y}(x;\theta_y)$ and has to be treated as a soft constraint; see (3.18) for the details.

Remark 3.1. Note that the neural network $\hat{y}(x;\theta_y)$ given by (3.8) reduces to the DSCNN $\hat{y}(x, z;\theta_y)$ used in Algorithm 3.1 by taking $g = 0, h = 1$ and replacing ϕ with the piecewise constant

$$z(x) = \begin{cases} 1 & \text{if } x \in \Omega^+,\\ -1 & \text{if } x \in \Omega^-. \end{cases}$$

However, this auxiliary variable z does not satisfy the assumptions in (3.9). Hence, the auxiliary function ϕ is a nontrivial generalization of the augmented coordinate variable z introduced in the DSCNN.

Given $\mathcal{N}_y$ a neural network with smooth activation functions, we have that $\mathcal{N}_y \in C^\infty(\overline{\Omega})$. Moreover, it follows from the smooth assumptions of g and ϕ in (3.4) and (3.9) that $\hat{y}|_{\Omega^+} \in C^2(\overline{\Omega^+})$ and $\hat{y}|_{\Omega^-} \in C^2(\overline{\Omega^-})$. Hence, the second-order derivatives of $\hat{y}$ are well-defined and continuous on Ω^- and Ω^+. In particular, we have that

$$
\begin{cases}
\dfrac{\partial \hat{y}}{\partial x_i} = \dfrac{\partial g}{\partial x_i} + h\left(\dfrac{\partial \mathcal{N}_y}{\partial x_i} + \dfrac{\partial \mathcal{N}_y}{\partial \phi}\dfrac{\partial \phi}{\partial x_i}\right) + \mathcal{N}_y \dfrac{\partial h}{\partial x_i}, \\
\dfrac{\partial^2 \hat{y}}{\partial x_i^2} = \dfrac{\partial^2 g}{\partial x_i^2} + h\left(\dfrac{\partial^2 \mathcal{N}_y}{\partial x_i^2} + 2\dfrac{\partial^2 \mathcal{N}_y}{\partial x_i \partial \phi}\dfrac{\partial \phi}{\partial x_i} + \dfrac{\partial^2 \mathcal{N}_y}{\partial \phi^2}\left(\dfrac{\partial \phi}{\partial x_i}\right)^2 + \dfrac{\partial \mathcal{N}_y}{\partial \phi}\dfrac{\partial^2 \phi}{\partial x_i^2}\right) \\
\qquad\qquad + 2\dfrac{\partial h}{\partial x_i}\left(\dfrac{\partial \mathcal{N}_y}{\partial x_i} + \dfrac{\partial \mathcal{N}_y}{\partial \phi}\dfrac{\partial \phi}{\partial x_i}\right) + \mathcal{N}_y \dfrac{\partial^2 h}{\partial x_i^2}.
\end{cases} \tag{3.11}
$$

Similar to what we have done for the state variable y, we can also approximate p by a neural network with the boundary and interface conditions in (2.6) as hard constraints. To be concrete, since the boundary and interface conditions for p are homogeneous, we approximate it by

$$
\hat{p}(x;\theta_p) = h(x)\mathcal{N}_p(x,\phi(x);\theta_p), \tag{3.12}
$$

where $\mathcal{N}_p(x,\phi(x);\theta_p)$ is a neural network with smooth activation functions and parameterized by θ_p, the functions h and ϕ satisfy (3.5) and (3.9), respectively. In particular, the functions h and ϕ for $\hat{p}$ can be the same as the ones for $\hat{y}$. The derivatives of $\hat{p}$ can be calculated in the same ways as those in (3.10) and (3.11), that is,

$$
\begin{cases}
\dfrac{\partial \hat{p}}{\partial x_i} = h\left(\dfrac{\partial \mathcal{N}_p}{\partial x_i} + \dfrac{\partial \mathcal{N}_p}{\partial \phi}\dfrac{\partial \phi}{\partial x_i}\right) + \mathcal{N}_p \dfrac{\partial h}{\partial x_i}, \\
\dfrac{\partial^2 \hat{p}}{\partial x_i^2} = h\left(\dfrac{\partial^2 \mathcal{N}_p}{\partial x_i^2} + 2\dfrac{\partial^2 \mathcal{N}_p}{\partial x_i \partial \phi}\dfrac{\partial \phi}{\partial x_i} + \dfrac{\partial^2 \mathcal{N}_p}{\partial \phi^2}\left(\dfrac{\partial \phi}{\partial x_i}\right)^2 + \dfrac{\partial \mathcal{N}_p}{\partial \phi}\dfrac{\partial^2 \phi}{\partial x_i^2}\right) \\
\qquad\qquad + 2\dfrac{\partial h}{\partial x_i}\left(\dfrac{\partial \mathcal{N}_p}{\partial x_i} + \dfrac{\partial \mathcal{N}_p}{\partial \phi}\dfrac{\partial \phi}{\partial x_i}\right) + \mathcal{N}_p \dfrac{\partial^2 h}{\partial x_i^2}, \\
[\beta \partial_{\boldsymbol{n}} \hat{p}]_\Gamma = (\beta^+ - \beta^-)\left(\mathcal{N}_p \boldsymbol{n}\cdot\nabla h + h(\boldsymbol{n}\cdot\nabla_x \mathcal{N}_p)\right) + \dfrac{\partial \mathcal{N}_p}{\partial \phi}\left(h[\beta \partial_{\boldsymbol{n}}\phi]_\Gamma\right).
\end{cases} \tag{3.13}
$$

3.3. The choice of ϕ. We note that the abstract and general neural networks $\hat{y}(x;\theta_y)$ and $\hat{p}(x;\theta_p)$ given in (3.8) and (3.12) can be used in practice only when the auxiliary function $\phi(x)$ satisfying (3.9) is chosen appropriately. In this subsection, we illustrate how to choose $\phi(x)$ for interfaces with different geometrical properties. In particular, we shall show that if the shapes of Ω^+, Ω^-, and Γ are regular enough and their analytic expressions are known, then we can construct an auxiliary function $\phi(x)$ analytically.

First, if Γ is the regular zero level set of a function $\psi \in C^2(\overline{\Omega})$,[1] then we can define $\phi(x)$ as follows:

$$
\phi(x) = \begin{cases} \psi(x) & \text{if } x \in \Omega^-, \\ 0 & \text{if } x \in \Omega^+ \cup \Gamma \cup \partial\Omega, \end{cases}
$$

which is smooth and clearly satisfies (3.9). We present an example below for further explanations.

Example 3.2 (circle-shaped interfaces). Consider a domain $\Omega \subset \mathbb{R}^d$ and the interface $\Gamma \subset \Omega$ given by the circle $\Gamma = \{x \in \Omega : \|x\|_2 = r_0\}$, with $r_0 > 0$. The domain Ω is divided into two parts: $\Omega^- = \{x \in \Omega : \|x\|_2 < r_0\}$ and $\Omega^+ = \{x \in \Omega : \|x\|_2 > r_0\}$.

[1]The zero level set of ψ is regular means that it does not contain any point where $\nabla\psi$ vanishes.

In this case, the interface Γ is the regular zero level set of $\psi(x) = r_0^2 - \|x\|_2^2$ and the auxiliary function ϕ can be defined as

$$\phi(x) = \begin{cases} r_0^2 - \|x\|_2^2 & \text{if } x \in \Omega^-, \\ 0 & \text{if } x \in \Omega^+ \cup \Gamma \cup \partial\Omega. \end{cases}$$

The above idea can be easily extended to the case where Γ is a finite union of the regular zero level sets of some functions $\psi_1, \ldots, \psi_n \in C^2(\overline{\Omega^-})(n \geq 1)$. See Example 3.3 for a concrete explanation.

Example 3.3 (box-shaped interfaces). Consider a domain $\Omega \subset \mathbb{R}^d$ containing the box $B := [a_1, b_1] \times \cdots \times [a_d, b_d] \in \mathbb{R}^d$. The interface $\Gamma := \partial B$ divides Ω into $\Omega^- = (a_1, b_1) \times \cdots \times (a_d, b_d)$ and $\Omega^+ = \Omega \backslash B$. Here the subdomain Ω^- is the intersection of $2d$ half-spaces, whose corresponding hyperplanes are the zero level sets of $\psi_i(x) = x_i - a_i,\ i = 1, \ldots, d$, and $\psi_i(x) = b_{i-d} - x_{i-d},\ i = d+1, \ldots, 2d$, respectively. When each ψ_i is treated as a function defined on Ω^-, we can see that $\psi_i \in C^2(\overline{\Omega^-})$ and Γ is indeed characterized by the union of the regular zero level sets of $\psi_1, \ldots, \psi_{2d}$. In this case, we define

$$\phi(x) = \begin{cases} \prod_{i=1}^d (x_i - a_i)(b_i - x_i) & \text{if } x \in \Omega^-, \\ 0 & \text{if } x \in \Omega^+ \cup \Gamma \cup \partial\Omega. \end{cases} \tag{3.14}$$

This is a smooth function satisfying (3.9). In particular, the pairwise intersections of the zero level sets of $\{\psi_i(x)\}_{i=1}^{2d}$ have measure zero, and the regularity of these zero level sets ensures that $[\beta \partial_{\boldsymbol{n}} \phi]_\Gamma \neq 0$ almost everywhere.

Next, we consider a more general case, where Γ is a finite union of the regular zero level sets of functions $\psi_1, \psi_2, \ldots, \psi_n$, which are of class C^2 only in an open neighborhood of Γ. Such a situation arises, for instance, when Γ is defined by the zero level set of a function represented in polar coordinates since the angle parameter is not differentiable at the origin. Due to the lack of global smoothness, we cannot simply define $\phi(x)$ as in (3.14); otherwise, the resulting ϕ may not satisfy the assumption $\phi \in C^2(\Omega^-)$ in (3.9). To tackle this issue, we propose to set $\phi(x) = 0$ when $x \in \overline{\Omega^+}$ and $\phi(x)$ to be a nonzero constant over the region inside Ω^- where $\psi_1, \ldots, \psi_n$ fail to be C^2 functions. Then, in the rest of the domain, we define $\phi(x)$ by using $\psi_1, \ldots, \psi_n$ so that the resulting piecewise function $\phi(x)$ is well-defined and satisfies (3.9). We shall elaborate on the above ideas in the remainder of this section, and for this purpose, we make the following assumptions.

Assumption 3.4. The subdomain Ω^- is the intersection of the interior of finitely many oriented, smooth, and embedded manifolds $M_1, M_2, \ldots, M_n$, where $M_i \cap M_j$ is of measure zero whenever $i \neq j$ and $i, j \in \{1, \ldots, n\}$.

Assumption 3.5. There exists an open neighborhood $U \subset \mathbb{R}^d$ of Γ, such that for each $i \in \{1, \ldots, n\}$ and manifold M_i, there exist functions $\psi_i : U \to \mathbb{R}$ satisfying $\psi_i \in C^2(\overline{U})$ and

$$\psi_i(x) = 0 \text{ if } x \in M_i \cap \Gamma, \quad \psi_i(x) > 0 \text{ if } x \in U \cap \Omega^-, \quad \partial_{\boldsymbol{n}} \psi_i \neq 0 \text{ on } M_i \cap \Gamma.$$

Assumption 3.6. There exist positive constants $c_1, \ldots, c_n$ such that $\psi_i(x) > c_i$ for all $x \in \partial U \cap \overline{\Omega^-}$ and for all $i \in \{1, \ldots, n\}$.

Under Assumptions 3.4–3.6, we have the following result.

THEOREM 3.7. *Suppose Assumptions* 3.4–3.6 *hold and we define* $\psi : U \to \mathbb{R}$ *as* $\psi(x) = \prod_{i=1}^{n} \psi_i(x)$. *For any constant* c *such that* $0 < c < \prod_{i=1}^{n} c_i$, *let* $L_c := \{x \in U : \psi(x) \geq c\}$. *Then the function* $\phi : \bar{\Omega} \to \mathbb{R}$ *given by*

$$\phi(x) = \begin{cases} c^3 & \text{if } x \in (\overline{\Omega^-} \backslash U) \cup (\overline{\Omega^-} \cap L_c), \\ c^3 - (c - \psi(x))^3 & \text{if } x \in (U \cap \overline{\Omega^-}) \backslash L_c, \\ 0 & \text{if } x \in \overline{\Omega^+} \end{cases} \tag{3.15}$$

is well-defined and satisfies (3.9).

Proof. Let

$$D_1 := (\overline{\Omega^-} \backslash U) \cup (\overline{\Omega^-} \cap L_c), D_2 := (U \cap \overline{\Omega^-}) \backslash L_c, D_3 := \overline{\Omega^+};$$

then we shall show that ϕ is a well-defined piecewise function, i.e., $\bigcup_{i=1}^{3} D_i = \overline{\Omega}$ and ϕ has consistent function values over $\overline{D_1} \cap \overline{D_2}$, $\overline{D_2} \cap \overline{D_3}$, and $\overline{D_3} \cap \overline{D_1}$.

First, observe that $D_1 \cup D_2 = \overline{\Omega^-}$, so $D_1 \cup D_2 \cup D_3 = \overline{\Omega}$. Moreover, D_1 and D_3 are closed in $\mathbb{R}^d$. We next prove that (1) $\overline{D_1} \cap \overline{D_3} = \varnothing$, (2) $\overline{D_2} \cap \overline{D_3} \subset \Gamma$, and (3) $\overline{D_1} \cap \overline{D_2} \subset \partial L_c$.

(1) Since Ω^- is a subdomain of Ω, it is open and connected. We thus have $\overline{D_1} = D_1 \subset \overline{\Omega^-}$ and $\overline{D_3} = D_3 \subset \overline{\Omega^+}$ such that $\overline{D_1} \cap \overline{D_3} \subset \Gamma$. Then, since $\Gamma \subset U$ and $\Gamma \cap L_c = \varnothing$, we have $\overline{D_1} \cap \Gamma = D_1 \cap \Gamma = \varnothing$. Hence, $\overline{D_1} \cap \overline{D_3} = \varnothing$.

(2) Note that

$$\overline{D_2} \subset \overline{U} \cap \overline{\Omega^-} \cap \overline{\mathbb{R}^d \backslash L_c} \subset \overline{\Omega^-} \tag{3.16}$$

and $\overline{D_3} \subset \overline{\Omega^+}$; then we have $\overline{D_2} \cap \overline{D_3} \subset \Gamma$.

(3) Denote $c_0 := \prod_{i=1}^{n} c_i$. If $x \in \overline{D_1} \cap \overline{D_2} = D_1 \cap \overline{D_2}$, then by the decomposition of $\overline{D_2}$ in (3.16), we have $x \in \overline{U} \backslash U \subset \partial U$ or $x \in \overline{(\mathbb{R}^d \backslash L_c)} \cap L_c \subset \partial L_c$. If $x \in \partial U$, by Assumption 3.6 we have $\psi(x) > c_0$. But $x \in \overline{D_2}$ implies that $x \in \overline{\mathbb{R}^d \backslash L_c}$, i.e., $\psi(x) \leq c < c_0$, contradicting $\psi(x) > c_0$. We thus have that $x \in \partial L_c$.

By the above claims (1)–(3), we have

$$\begin{aligned} x \in \overline{D_1} \cap \overline{D_2} \implies x \in \partial L_c \implies \psi(x) = c \implies c^3 = c^3 - (c - \psi(x))^3, \\ x \in \overline{D_2} \cap \overline{D_3} \implies x \in \Gamma \implies \psi(x) = 0 \implies c^3 - (c - \psi(x))^3 = 0. \end{aligned}$$

Moreover, $\overline{D_1} \cap \overline{D_3} = \varnothing$. Therefore, the piecewise definition of ϕ is consistent, and hence ϕ is well-defined.

Since ϕ is smooth in D_1, D_2, D_3 and clearly continuous on Γ and ∂L_c, we have $\phi|_{\Omega^+} \in C^2(\overline{\Omega^+})$ and $\phi \in C(\overline{\Omega})$. The first- and second-order derivatives of $\phi(x)$ all tend to zero as x approaches ∂L_c in D_2, so $\phi|_{\Omega^-} \in C^2(\overline{\Omega^-})$.

By Assumption 3.5 and (3.15), it is clear that $\phi(x) = 0$ for all $x \in \Gamma$ and $\phi(x) > 0$ on Ω^-. Finally, we have

$$[\beta \partial_{\boldsymbol{n}} \phi]_\Gamma(x) = -3\beta^-(c - \psi(x))^2 \left(\sum_{i=1}^{n} (\partial_{\boldsymbol{n}} \psi_i)(x) \prod_{j=1, j \neq i}^{n} \psi_j(x) \right) \quad \forall x \in \Gamma.$$

Since $c > 0$, by Assumption 3.5, $[\beta \partial_{\boldsymbol{n}} \phi]_\Gamma(x)$ equals 0 if and only if there exist at least two distinct indexes $i, j \in \{1, \ldots, n\}$ such that $\psi_i(x) = \psi_j(x) = 0$. However, it follows from Assumption 3.4 that $M_i \cap M_j$ is of measure zero for all i, j with $i \neq j$, so the set

$\{x \in \Gamma : \psi_i(x) = \psi_j(x) = 0\}$ is also of measure zero. This shows that $[\beta \partial_{\boldsymbol{n}} \phi]_\Gamma \neq 0$ a.e. on Γ. ∎

Theorem 3.7 provides a generic method for constructing an auxiliary function $\phi(x)$ when Assumptions 3.4–3.6 are satisfied. This method is independent of the PDE and is only related to the shape of the interface Γ, and it can be easily applied to other types of interface problems with slight modifications.

Note that although Assumptions 3.4–3.6 look complicated, they can be satisfied by a large class of interfaces, which are of great practical interest. We present an example in Example 3.8 below for explanations. Note that the interfaces in Example 3.8 have complex geometry and are challenging to be addressed by traditional mesh-based numerical methods.

Example 3.8 (star-shaped interfaces). Let $\Omega \subset \mathbb{R}^2$ be a bounded domain and the star-shaped interface $\Gamma \subset \Omega$ be defined by the zero level set of the following function in polar coordinates: $\psi(r,\theta) = r - a - b\sin(5\theta)$ with constants $b < a$. The domain Ω is divided into $\Omega^- = \{(r,\theta) \in \mathbb{R}^2 : r < a + b\sin(5\theta)\}$ and $\Omega^+ = \{(r,\theta) \in \Omega : r > a + b\sin(5\theta)\}$. Note that $\psi(r,\theta)$ is not differentiable on Ω since the polar angle is not differentiable at the origin. In this case, it follows from Theorem 3.7 that we can define

$$\phi(r,\theta) = \begin{cases} \left(\dfrac{a-b}{2}\right)^3 & \text{if } a + b\sin(5\theta) - r \geq \frac{a-b}{2}, \\ \left(\dfrac{a-b}{2}\right)^3 - \left(\dfrac{a-b}{2} + \psi(r,\theta)\right)^3 & \text{if } 0 < a + b\sin(5\theta) - r < \frac{a-b}{2}, \\ 0 & \text{otherwise,} \end{cases}$$

and one can check that ϕ satisfies (3.9).

Remark 3.9. We mention that if it is difficult to construct an auxiliary function $\phi(x)$ with an analytic form, we can train a DCSNN to represent $\phi(x)$. To this end, we impose the constraints $[\phi]_\Gamma = 0$, $[\beta \partial_{\boldsymbol{n}} \phi]_\Gamma = \gamma$, where the function $\gamma : \Gamma \to \mathbb{R}$ is nonzero almost everywhere. Then we train a DCSNN $\hat{\phi}(x, z; \theta_\phi)$ with smooth activation functions by minimizing the following loss function:

$$\begin{aligned} &\frac{w_1}{M_\Gamma} \sum_{i=1}^{M_\Gamma} \left| \hat{\phi}(x_\Gamma^i, 1; \theta_\phi) - \hat{\phi}(x_\Gamma^i, -1; \theta_\phi) \right|^2 \\ &\quad + \frac{w_2}{M_\Gamma} \sum_{i=1}^{M_\Gamma} \left| \boldsymbol{n} \cdot (\beta^+ \nabla_x \hat{\phi}(x_i^\Gamma, 1; \theta_\phi) - \beta^- \nabla_x \hat{\phi}(x_i^\Gamma, -1; \theta_\phi)) - \gamma(x_\Gamma^i) \right|^2, \end{aligned} \tag{3.17}$$

where $\{x_\Gamma^i\}_{i=1}^{M_\Gamma} \subset \Gamma$ and $w_1, w_2 > 0$ are the weights. It is clear that the trained $\hat{\phi}$ satisfies the smoothness requirements in (3.9).

3.4. The hard-constraint PINN method for (2.1)–(2.3). In this subsection, we propose a hard-constraint PINN method for problem (2.1)–(2.3) based on the discussions in subsections 3.2 and 3.3. For this purpose, we first approximate the state variable y and the adjoint variable p by the neural networks $\hat{y}(x; \theta_y)$ and $\hat{p}(x; \theta_p)$ given in (3.8) and (3.12), respectively. As a result, the boundary and interface conditions in (2.6) and (2.7) are satisfied automatically. Then (2.6) and (2.7) can be solved by minimizing the following loss function:

(3.18)
$$\begin{aligned}\mathcal{L}_{HC}(\theta_y,\theta_p) = \frac{w_{y,r}}{M}\sum_{i=1}^{M}\left|-\Delta_x\hat{y}(x^i;\theta_y) - \frac{\mathcal{P}_{[u_a(x^i),u_b(x^i)]}(-\frac{1}{\alpha}\hat{p}(x^i;\theta_p)) + f(x^i)}{\beta^{\pm}}\right|^2 \\ + \frac{w_{y,\Gamma_n}}{M_\Gamma}\sum_{i=1}^{M_\Gamma}\left|[\beta\partial_{\boldsymbol{n}}\hat{y}]_\Gamma(x_\Gamma^i;\theta_y) - g_1(x_\Gamma^i)\right|^2 \\ + \frac{w_{p,r}}{M}\sum_{i=1}^{M}\left|-\Delta_x\hat{p}(x_i;\theta_p) - \frac{\hat{y}(x_i;\theta_y) - y_d(x_i)}{\beta^{\pm}}\right|^2 \\ + \frac{w_{p,\Gamma_n}}{M_\Gamma}\sum_{i=1}^{M_\Gamma}\left|[\beta\partial_{\boldsymbol{n}}\hat{p}]_\Gamma(x_i^\Gamma;\theta_p)\right|^2.\end{aligned}$$

Clearly, the loss function $\mathcal{L}_{HC}$ can be calculated by (3.8) and (3.10)–(3.13). Similar to (3.3), the loss function $\mathcal{L}_{HC}(\theta_y,\theta_p)$ in (3.18) can be minimized by a stochastic optimization method, where all the derivatives $\Delta_x\hat{y}$, $\nabla_x\hat{y}$, $\Delta_x\hat{p}$, $\nabla_x\hat{p}$ and the gradients $\frac{\partial\mathcal{L}_{HC}}{\partial\theta_y}$, $\frac{\partial\mathcal{L}_{HC}}{\partial\theta_p}$ are computed by automatic differentiation.

We remark that if g, h, and ϕ cannot be constructed with analytical forms, then we can represent them by training three neural networks $\hat{g}$, $\hat{h}$, and $\hat{\phi}$ as shown in (3.6), (3.7), and (3.17). Note that these neural networks are expected to be easy to train due to the simple structures of the loss functions (3.6), (3.7), and (3.17). More importantly, with the pretrained $\hat{g}$, $\hat{h}$, and $\hat{\phi}$, the boundary and interface conditions are decoupled from the learning of the PDE. Hence, this hard-constraint approach is still superior since it can reduce the training difficulty and improve the numerical accuracy of Algorithm 3.1.

We summarize the proposed hard-constraint PINN method for (2.1)–(2.3) in Algorithm 3.2. We provide two options for inputting the functions g, h, and ϕ in (3.8) and (3.12): In Option I, the input functions g, h, and ϕ have closed-form expressions, while in Option II they are numerically approximated by pretrained neural networks. We reiterate that the boundary and interface conditions for y and p are imposed as hard constraints in the neural networks $\hat{y}(x;\theta_y)$ and $\hat{p}(x;\theta_p)$. Hence, compared with Algorithm 3.1, Algorithm 3.2 reduces the numerical error at the boundary and interface and is easier and cheaper to implement.

Algorithm 3.2. The hard-constraint PINN method for (2.1)–(2.3).

Input (Option I): Weights $w_{y,r}, w_{p,\Gamma_n}, w_{p,r}, w_{p,\Gamma_n}$, functions g, h, and ϕ with closed-form expressions and satisfying (3.4), (3.5), (3.9), respectively.

Input (Option II): Weights $w_{y,r}, w_{p,\Gamma_n}, w_{p,r}, w_{p,\Gamma_n}$, pretrained neural networks $\hat{g}$, $\hat{h}$, and $\hat{\phi}$ that approximate any functions satisfying (3.4), (3.5), (3.9), respectively.

1: Initialize the neural networks $\mathcal{N}_y(x,\phi(x);\theta_y)$ and $\mathcal{N}_p(x,\phi(x);\theta_p)$ with parameters θ_y^0 and θ_p^0.
2: Sample training sets $\mathcal{T} = \{x^i\}_{i=1}^M \subset \Omega$ and $\mathcal{T}_\Gamma = \{x_\Gamma^i\}_{i=1}^{M_\Gamma} \subset \Gamma$.
3: Calculate the function values of g, h, ϕ and their first- and second-order derivatives over $\mathcal{T}$, and the values of $[\beta\partial_{\boldsymbol{n}}g]_\Gamma, [\beta\partial_{\boldsymbol{n}}\phi]_\Gamma$ over $\mathcal{T}_\Gamma$.
4: Train the neural networks $\hat{y}(x;\theta_y)$ and $\hat{p}(x;\theta_p)$ to identify the optimal parameters θ_y^* and θ_p^* by minimizing (3.18).
5: $\hat{u}(x) \leftarrow \min\{u_b(x), \max\{u_a(x), -\frac{1}{\alpha}\hat{p}(x;\theta_p^*)\}\}$.

Output: Approximate solutions $\hat{y}(x;\theta_y^*)$ and $\hat{u}(x)$.

4. Numerical results. In this section, we report some numerical results of Algorithms 3.1 and 3.2 for solving problem (2.1)–(2.3) and numerically verify the superiority of Algorithm 3.2 to Algorithm 3.1. All the codes of our numerical experiments were written with PyTorch (version 1.13) and are available at https://github.com/tianyouzeng/PINNs-interface-optimal-control. In particular, we use the hyperbolic tangent function as the active functions in all the neural networks.

To test the accuracy of the results computed by Algorithms 3.1 and 3.2, we select 256×256 testing points $\{x^i\}_{i=1}^{M_T} \subset \Omega$ following the Latin hypercube sampling [33]. We then compute

$$\varepsilon_{\text{abs}} = \sqrt{\frac{1}{M_T}\sum_{i=1}^{M_T}(\hat{u}(x^i) - u^*(x^i))^2} \text{ and } \varepsilon_{rel} = \varepsilon_{abs}\sqrt{A(\Omega)}/||u^*||_{L^2(\Omega)} \tag{4.1}$$

as the absolute and relative L^2-errors of $\hat{u}$, where $A(\Omega)$ is the area of Ω (i.e., the Lebesgue measure of Ω), and $||u^*||_{L^2(\Omega)}$ is computed using the numerical integration function `dblquad` implemented in the SciPy library of Python.

Example 1. We first demonstrate an example of (2.1)–(2.3), where $U_{ad} = \{u \in L^2(\Omega) : -1 \le u \le 1 \text{ a.e. in } \Omega\}$. We set $\Omega = (-1,1)\times(-1,1)$, $\Gamma = \{x \in \Omega : \|x\|_2 \le r_0\} \subset \Omega$ with $r_0 = 0.5$, $\alpha = 1$, $\beta^- = 1$, and $\beta^+ = 10$. We further choose $g_0 = 0$, $g_1 = 0$, and $h_0 = \frac{(x_1^2+x_2^2)^{3/2}}{\beta^+} + (\frac{1}{\beta^-} - \frac{1}{\beta^+})r_0^3$. Following [55], we let

$$p^*(x_1,x_2) = \begin{cases} -5(x_1^2+x_2^2-r_0^2)(x_1^2-1)(x_2^2-1)/\beta^- \text{ in } \Omega^-, \\ -5(x_1^2+x_2^2-r_0^2)(x_1^2-1)(x_2^2-1)/\beta^+ \text{ in } \Omega^+, \end{cases} \tag{4.2}$$

$$y^*(x_1,x_2) = \begin{cases} (x_1^2+x_2^2)^{3/2}/\beta^- \text{ in } \Omega^-, \\ (x_1^2+x_2^2)^{3/2}/\beta^+ + (1/\beta^- - 1/\beta^+)r_0^3 \text{ in } \Omega^+, \end{cases} \tag{4.3}$$

$$u^*(x_1,x_2) = \max\left\{-1, \min\left\{1, -\frac{1}{\alpha}p^*(x_1,x_2)\right\}\right\}, \tag{4.4}$$

and

$$\begin{aligned} f(x_1,x_2) &= -u^*(x_1,x_2) - \nabla\cdot(\nabla\beta^{\pm}y^*(x_1,x_2)) \text{ in } \Omega^{\pm}, \\ y_d(x_1,x_2) &= y^*(x_1,x_2) + \nabla\cdot(\nabla\beta^{\pm}p^*(x_1,x_2)) \text{ in } \Omega^{\pm}. \end{aligned}$$

Then it is easy to verify that $(u^*, y^*)^\top$ is the unique solution of this example.

To implement the soft-constraint PINN method in Algorithm 3.1, we approximate y and p, respectively, by two fully connected neural networks $\hat{y}(x,z;\theta_y)$ and $\hat{p}(x,z;\theta_p)$, where z is the augmented coordinate variable in the DCSNN (see section 2.2). All the neural networks consist of only one hidden layer with 100 neurons and `tanh` activation functions. To implement the hard-constraint PINN method in Algorithm 3.2, we define two neural networks $\mathcal{N}_y(x,\phi(x);\theta_y)$ and $\mathcal{N}_p(x,\phi(x);\theta_p)$ with the same structures as those of $\hat{y}$ and $\hat{p}$. Then the state and adjoint variables are approximated by the neural networks given in (3.8) and (3.12). We consider both input Options I and II in Algorithm 3.2. For Option I, we choose $g(x) = \frac{(x_1^2x_2^2+1)^{3/2}}{\beta^+} + (\frac{1}{\beta^-} - \frac{1}{\beta^+})r_0^3$ and $h(x) = (x_1^2-1)(x_2^2-1)$. The auxiliary function ϕ is defined analytically by

$$\phi(x) = \begin{cases} 4\|x\|_2^2 & \text{if } x \in \Omega^-, \\ 1 & \text{otherwise.} \end{cases} \tag{4.5}$$

It is easy to check that these functions satisfy the requirements in (3.4), (3.5), and (3.9). For Option II, we first randomly sample the training sets $\mathcal{T} = \{x^i\}_{i=1}^{M} \subset \Omega$, $\mathcal{T}_B = \{x_B^i\}_{i=1}^{M_B} \subset \partial\Omega$, and $\mathcal{T}_\Gamma = \{x_\Gamma^i\}_{i=1}^{M_\Gamma} \subset \Gamma$ with $M = 1024$ and $M_B = M_\Gamma = 256$. Then we train two fully connected shallow neural networks[2] $\hat{g}(x;\theta_g)$ and $\hat{h}(x;\theta_h)$ with one hidden layer, 500 neurons, and `tanh` activation functions by minimizing the loss functions

$$\mathcal{L}_g(\theta_g) = \frac{1}{M_B}\sum_{i=1}^{M_B}\left|\hat{g}(x_B^i;\theta_g) - h_0(x_B^i)\right|^2 \tag{4.6}$$

and

$$\mathcal{L}_h(\theta_h) = \frac{0.01}{M}\sum_{i=1}^{M}\left|\hat{h}(x^i;\theta_h) - \bar{h}(x^i)\right|^2 + \frac{1}{M_B}\sum_{i=1}^{M_B}\left|\hat{h}(x_B^i;\theta_h)\right|^2, \tag{4.7}$$

where $\bar{h}(x) := \cos(\frac{\pi}{2}x_1)\cos(\frac{\pi}{2}x_2)$. Each neural network is trained with the L-BFGS optimizer with stepsize 1 and strong Wolfe condition for 200 iterations. Then we train a DCSNN $\hat{\phi}(x,z;\theta_\phi)$ with one hidden layer, 200 neurons, and `tanh` activation functions by minimizing the loss function

$$\begin{aligned}\mathcal{L}_\phi(\theta_\phi) = {} & \frac{1}{M_\Gamma}\sum_{i=1}^{M_\Gamma}\left|\hat{\phi}(x_\Gamma^i,1;\theta_\phi) - \hat{\phi}(x_\Gamma^i,-1;\theta_\phi)\right|^2 \\ & + \frac{1}{M_\Gamma}\sum_{i=1}^{M_\Gamma}\left|\boldsymbol{n}\cdot\nabla_x\hat{\phi}(x_\Gamma^i,1;\theta_\phi) - 5\right|^2 + \frac{1}{M_\Gamma}\sum_{i=1}^{M_\Gamma}\left|\boldsymbol{n}\cdot\nabla_x\hat{\phi}(x_\Gamma^i,-1;\theta_\phi)\right|^2.\end{aligned} \tag{4.8}$$

The Adam optimizer [22] is applied to train $\hat{\phi}(x,z;\theta_\phi)$ for 30,000 iterations. The learning rate is initially set to be 5×10^{-4} and finally reduces to 10^{-4} by a preset scheduler. It can be verified that the functions $\hat{g}$, $\hat{h}$, and $\hat{\phi}$ are the approximations of some functions g, h, and ϕ satisfying the requirements (3.4), (3.5), and (3.9).

To train the neural networks $\hat{y}(x;\theta_y)$ and $\hat{p}(x;\theta_p)$, we adopt the same training sets $\mathcal{T}$, $\mathcal{T}_B$, and $\mathcal{T}_\Gamma$ as sampled above. All the neural networks are initialized using the default initializer of PyTorch. The weights are tuned so that the magnitude of each term in the loss function (3.3) or (3.18) is balanced. In particular, by adjusting the weights carefully, we set $w_{y,r} = w_{y,\Gamma} = w_{y,\Gamma_n} = w_{p,r} = w_{p,\Gamma} = w_{p,\Gamma_n} = 1, w_{y,b} = 2$, $w_{p,b} = 10$ for Algorithm 3.1; $w_{y,r} = 3$, $w_{y,\Gamma_n} = w_{p,r} = w_{p,\Gamma_n} = 1$ for Algorithm 3.2 with Option I; and $w_{y,r} = w_{y,\Gamma_n} = w_{p,\Gamma_n} = 1$,$w_{p,r} = 3$ for Algorithm 3.2 with Option II.

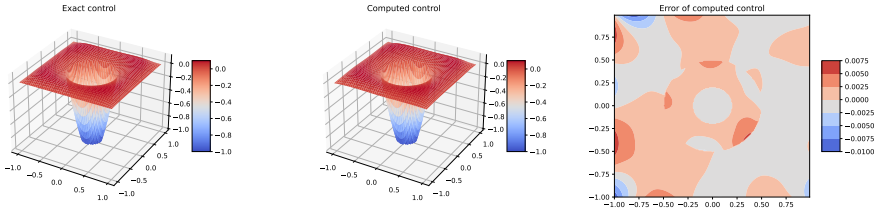
The Adam optimizer is used to train the neural networks $\hat{y}(x;\theta_y)$ and $\hat{p}(x;\theta_p)$ in Algorithms 3.1 and 3.2. We fix the number of Adam iterations to 60,000 for Algorithm 3.1 and Algorithm 3.2 with Option II and to 40,000 for Algorithm 3.2 with Option I. For Algorithm 3.1, the learning rate is initialized to be 10^{-2} and is reduced to 5×10^{-4} during the training by a preset scheduler. For Option I of Algorithm 3.2, the learning rate is initially set to be 10^{-3} and finally reduces to 3×10^{-5}. For Option II of Algorithm 3.2, the learning rate is initially set to be 5×10^{-3} and finally reduces to 3×10^{-4}.

The numerical results for Algorithms 3.1 and 3.2 are shown in Figures 2, 3, and 4, respectively. All the figures are plotted over a 200×200 uniform grid in $\overline{\Omega}$. The L^2-

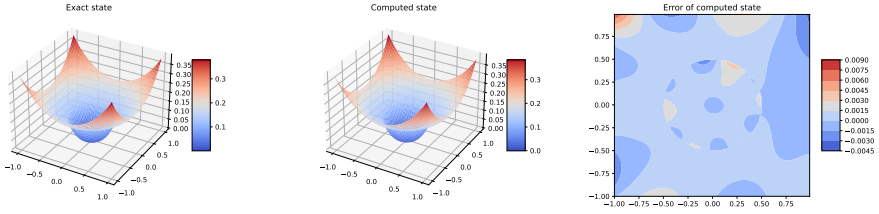
[2]Here, due to the homogeneous interface condition $g_0 = 0$, it suffices to approximate $\hat{g}$ by a fully connected neural network, rather than by a DSCNN as described in the previous section.

TABLE 1
The L^2-errors of the computed control u of Example 1 for Algorithms 3.1 and 3.2.

	ε_{abs}	ε_{rel}
Algorithm 3.1	6.6853×10^{-4}	2.3559×10^{-3}
Algorithm 3.2 with Option I	6.7087×10^{-5}	2.3062×10^{-4}
Algorithm 3.2 with Option II	2.0921×10^{-4}	7.1919×10^{-4}

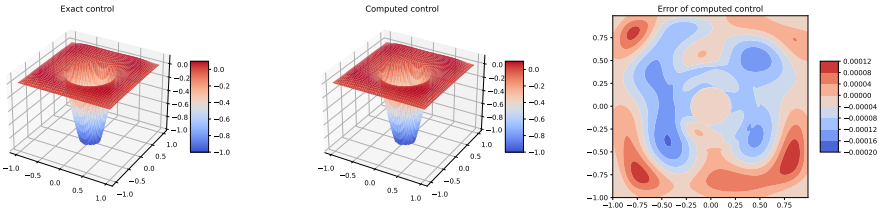


(a) Exact control u. (b) Computed control u. (c) Error of control u.

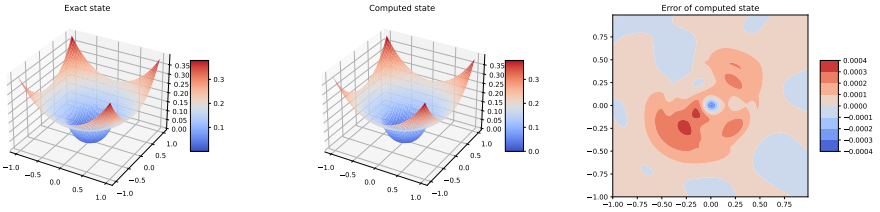


(d) Exact state y. (e) Computed state y. (f) Error of state y.

FIG. 2. *Numerical results of Algorithm 3.1 for Example 1.*



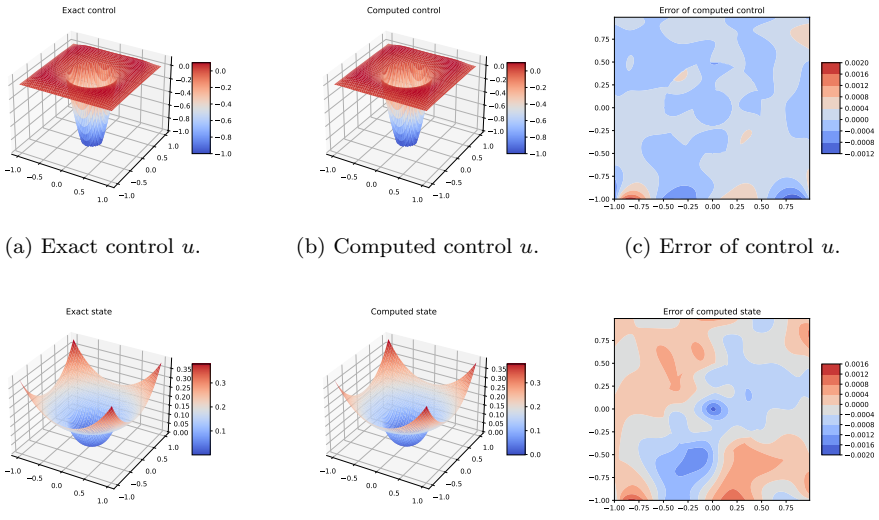
(a) Exact control u. (b) Computed control u. (c) Error of control u.



(d) Exact state y. (e) Computed state y. (f) Error of state y.

FIG. 3. *Numerical results of Algorithm 3.2 with Option I for Example 1.*

errors of the computed control are summarized in Table 1, where the results imply that the controls computed by Algorithm 3.2 are far more accurate than the one by Algorithm 3.1.



(a) Exact control u. (b) Computed control u. (c) Error of control u.

(d) Exact state y. (e) Computed state y. (f) Error of state y.

FIG. 4. *Numerical results of Algorithm* 3.2 *with Option* II *for Example* 1.

Moreover, it can be seen from Figures 2, 3, and 4 that, for Algorithm 3.1, the numerical errors are mainly accumulated on $\partial\Omega$ and Γ, and this issue is significantly alleviated in Algorithm 3.2 with Options I and II. We can also see that Algorithms 3.1 and 3.2 are effective in dealing with the control constraint $u \in U_{ad}$. All these results validate the advantage of Algorithm 3.2 over Algorithm 3.1 and the necessity of imposing the boundary conditions and interface conditions as hard constraints.

To further validate the effectiveness of Algorithm 3.2, we compare the numerical results with the immersed finite element method (IFEM) in [55], which is a benchmark mesh-based traditional numerical algorithm for solving elliptic interface optimal control problems. For this purpose, we consider another set of PDE coefficients $\beta^- = 1$ and $\beta^+ = 5$. With the updated β value, the numerical example becomes identical to the one in [55, Example 2], enabling us to directly compare the errors of the computed solutions with the ones reported in [55, Table 4]. In [55], the solutions are computed on an $N \times N$ uniform grid over $\overline{\Omega}$ with different mesh resolutions N. Here we fix the training resolution $N = 32$ for Algorithm 3.2 throughout the remaining experiments.

Again, we randomly sample the training sets $\mathcal{T} = \{x^i\}_{i=1}^{M} \subset \Omega$, $\mathcal{T}_B = \{x_B^i\}_{i=1}^{M_B} \subset \partial\Omega$, and $\mathcal{T}_\Gamma = \{x_\Gamma^i\}_{i=1}^{M_\Gamma} \subset \Gamma$ with $M = N^2$, $M_B = M_\Gamma = 8 \times N$, and $N = 32$. We adopt the same configurations for the neural networks $\hat{g}(x;\theta_g)$, $\hat{h}(x;\theta_h)$, $\hat{\phi}(x,z;\theta_\phi)$, $\hat{y}(x;\theta_y)$, and $\hat{p}(x;\theta_p)$ as above. We also apply the same optimization algorithms and stepsizes for training $\hat{g}(x;\theta_g)$, $\hat{h}(x;\theta_h)$, and $\hat{\phi}(x,z;\theta_\phi)$. To train the neural networks $\hat{y}(x;\theta_y)$ and $\hat{p}(x;\theta_p)$ in Algorithm 3.2, we fix the number of Adam iterations to 40,000 for Option I and to 60,000 for Option II. For Option I, the learning rate is initially set to be 1×10^{-3} and finally decreases to 3×10^{-5} by a preset scheduler. For Option II, the learning rate is initially set to be 5×10^{-3} and finally reduces to 3×10^{-4}. After the above iterations, we fix the parameters in $\hat{p}(x;\theta_p)$ and train $\hat{y}(x;\theta_y)$ for 300 L-BFGS iterations with the stepsize 1 to further improve the accuracy of $\hat{y}$

We use the L^2-errors defined in [55] to evaluate and compare the numerical accuracy of the computed solutions. Following [55], we evaluate the L^2-errors of the solution on an $N \times N$ uniform grid over $\overline{\Omega}$ with $N = 16$, 32, 64, 128, and 256. The results are reported in Table 2.

TABLE 2

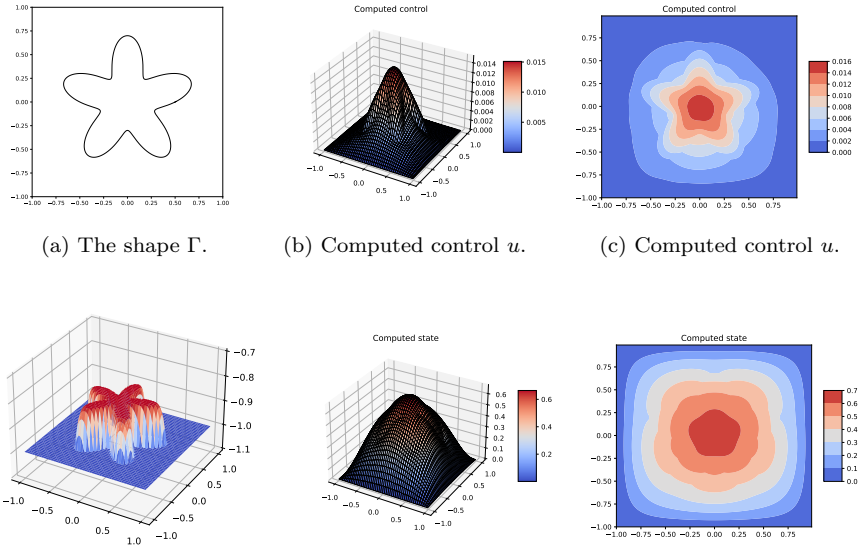
The L^2-errors of the computed solutions for Example 1 with $\beta^- = 1$ and $\beta^+ = 5$ evaluated on different grid resolutions N. Here u^*, y^*, and p^* are the exact solutions given in (4.2)–(4.4), and $\hat{u}$, $\hat{y}$, and $\hat{p}$ are the computed solutions by each corresponding algorithm. The results of IFEM [55] are computed and evaluated with each mesh resolution N, while the results of Algorithm 3.2 are computed with fixed sampling resolution $N = 32$ and evaluated with each mesh resolution N.

	$\|\hat{u} - u^*\|_{L^2(\Omega)}$		
N	IFEM [55]	Algorithm 3.2 with Option I	Algorithm 3.2 with Option II
16	2.0049×10^{-2}	7.8972×10^{-5}	9.4242×10^{-4}
32	5.8477×10^{-3}	8.0359×10^{-5}	7.8161×10^{-4}
64	1.4215×10^{-3}	8.1759×10^{-5}	7.1714×10^{-4}
128	3.6148×10^{-4}	8.2311×10^{-5}	6.8934×10^{-4}
256	9.6419×10^{-5}	8.2656×10^{-5}	6.7645×10^{-4}
	$\|\hat{p} - p^*\|_{L^2(\Omega)}$		
N	IFEM [55]	Algorithm 3.2 with Option I	Algorithm 3.2 with Option II
16	2.5823×10^{-2}	7.8993×10^{-5}	9.4317×10^{-4}
32	6.6744×10^{-3}	8.1344×10^{-5}	7.8366×10^{-4}
64	1.6418×10^{-3}	8.2633×10^{-5}	7.1928×10^{-4}
128	4.0256×10^{-4}	8.3286×10^{-5}	6.9175×10^{-4}
256	1.0293×10^{-4}	8.3614×10^{-5}	6.7891×10^{-4}
	$\|\hat{y} - y^*\|_{L^2(\Omega)}$		
N	IFEM [55]	Algorithm 3.2 with Option I	Algorithm 3.2 with Option II
16	5.6594×10^{-3}	1.3021×10^{-4}	7.9867×10^{-4}
32	1.4803×10^{-3}	1.3877×10^{-4}	6.8886×10^{-4}
64	3.6993×10^{-4}	1.4114×10^{-4}	6.4501×10^{-4}
128	9.4048×10^{-5}	1.4226×10^{-4}	6.2584×10^{-4}
256	2.2873×10^{-5}	1.4282×10^{-4}	6.1696×10^{-4}

When $N = 32$, the L^2-errors of the computed $\hat{u}$, $\hat{y}$, and $\hat{p}$ by Algorithm 3.2 with Options I and II are significantly lower than those by IFEM. Even if the mesh resolution increases to $N = 256$, Algorithm 3.2 with Option I and Option II is still comparable with IFEM. Moreover, note that after training the neural networks at the resolution $N = 32$, the evaluation of Algorithm 3.2 for a new resolution requires only a forward pass of these neural networks. In contrast, for each resolution, IFEM requires solving the elliptic interface optimal control problem from scratch, which is much more computationally expensive. These results validate that the mesh-free nature and the generalization ability of Algorithm 3.2 make it effective and numerically favorable for elliptic interface optimal control problems.

Example 2. To further validate the effectiveness of Algorithm 3.2 with Option I, we consider problem (2.1)–(2.3) with a complicated interface. In particular, we take $\Omega = (-1, 1) \times (-1, 1) \subset \mathbb{R}^2$, and the interface Γ is the curve defined by the polar coordinate equation $r = 0.5 + 0.2\sin(5\theta)$. The shape of Γ, Ω^-, and Ω^+ is illustrated in Figure 5(a). We then set $\alpha = 1$, $\beta^- = 1$, $\beta^+ = 10$, $g_0 = g_1 = 0$, $h_0 = 0$, $y_d(x) = (x_1^2 - 1)(x_2^2 - 1)$, and $f(x) = 2\beta^{\pm}(2 - x_1^2 - x_2^2)$ if $x \in \Omega^{\pm}$. Compared with Example 1, this example is more general and its exact solution is unknown.

To implement Algorithm 3.2 with Option I, we first define two fully connected neural networks $\mathcal{N}_y(x, \phi(x); \theta_y)$ and $\mathcal{N}_p(x, \phi(x); \theta_p)$, which consist of three hidden layers with 100 neurons per hidden layer. Then y and p are respectively approximated by the $\hat{y}(x; \theta_y)$ and $\hat{p}(x; \theta_p)$ given in (3.8) and (3.12) but with $g = 0$ and $h(x) = (x_1^2 - 1)(x_2^2 - 1)$. The auxiliary function ϕ can be constructed by Theorem 3.7 (see Example 3.8). Here we first define



(a) The shape Γ. (b) Computed control u. (c) Computed control u.

(d) The graph of $-\phi$. (e) Computed state y. (f) Computed state y.

FIG. 5. *Numerical settings and results for Example* 2. (a) *and* (d): *Star-shaped* Γ *and the auxiliary function* ϕ. (b) *and* (c): *The computed control* u. (e) *and* (f): *The computed state* y.

$$\phi_0(r,\theta) = \begin{cases} 0.2^3 & \text{if } r - 0.2\sin(5\theta) \le 0.3, \\ 0.2^3 - (0.2 - f(r,\theta))^3 & \text{if } 0.3 \le r - 0.2\sin(5\theta) \le 0.5, \\ 0 & \text{if } r - 0.2\sin(5\theta) \ge 0.5, \end{cases}$$

where $f(r,\theta) = 0.5 + 0.2\sin(5\theta) - r$. Then it is easy to check that $\phi := 1 - 20 \cdot \phi_0$ satisfies (3.9). The graph of $-\phi$ is shown in Figure 5(d).

We randomly sample the training sets $\mathcal{T} = \{x^i\}_{i=1}^{16384} \subset \Omega$ and $\mathcal{T}_\Gamma = \{x_\Gamma^i\}_{i=1}^{1024} \subset \Gamma$ with respect to the polar angle. The neural networks are initialized randomly following the default settings of PyTorch. We set $w_{p,\Gamma_n} = 3$ and $w_{y,r} = w_{y,\Gamma_n} = w_{p,r} = 1$. The neural networks are trained with $40,000$ Adam iterations, where θ_y and θ_p are updated simultaneously in each iteration. The initial learning rate is 10^{-2} in the first $5,000$ iterations, then 3×10^{-3} in $5,001$ to $10,000$ iterations, then 10^{-3} in $10,001$ to $20,000$ iterations, and finally 3×10^{-3} in $20,001$ to $40,000$ iterations.

The computed u and y are presented in Figure 5. We can see that the computed control and state by Option I of Algorithm 3.2 capture the nonsmoothness across the interface even if the geometry of the interface is complicated.

5. Extensions. In this section, we show that Algorithm 3.2 can be easily extended to other types of interface optimal control problems. For this purpose, we investigate an elliptic interface optimal control problem, where the control variable acts on the interface, and a parabolic interface optimal control problem.

5.1. Control on the interface. We consider the optimal control problem

$$\min_{y \in L^2(\Omega), u \in L^2(\Gamma)} J(y,u) := \frac{1}{2}\int_\Omega (y - y_d)^2 dx + \frac{\alpha}{2}\int_\Gamma u^2 dx \tag{5.1}$$
$$\text{s.t. } -\nabla \cdot (\beta \nabla y) = f \text{ in } \Omega\backslash\Gamma,\ [y]_\Gamma = g_0,\ [\beta \partial_{\boldsymbol{n}} y]_\Gamma = u + g_1 \text{ on } \Gamma,\ y = h_0 \text{ on } \partial\Omega,$$

together with the control constraint

$$u \in U_{ad} := \{u \in L^2(\Gamma) : u_a \le u \le u_b \text{ a.e. on } \Gamma\}, \tag{5.2}$$

where $u_a, u_b \in L^2(\Gamma)$. Above, all the notations are the same as those in (2.1)–(2.2), but the control variable $u \in L^2(\Gamma)$ in (5.1)–(5.2) acts on the interface rather than the source term. Existence and uniqueness of the solution to problem (5.1) can be found in [54], and we have the following results.

THEOREM 5.1 (cf. [54]). *Problem* (5.1)–(5.2) *admits a unique optimal control* $(u^*, y^*)^\top \in U_{ad} \times L^2(\Omega)$*, and the following first-order optimality system holds:*

$$u^* = \mathcal{P}_{U_{ad}}\Big(-\frac{p^*}{\alpha}\Big|_\Gamma (x)\Big), \tag{5.3}$$

where $\mathcal{P}_{U_{ad}}(\cdot)$ *denotes the projection onto* U_{ad}*, and* p^* *is the adjoint variable associated with* u^**, which is obtained from the successive solution of the following two equations:*

$$-\nabla\cdot(\beta\nabla y^*) = f \text{ in } \Omega\backslash\Gamma,\ [y^*]_\Gamma = g_0,\ [\beta\partial_{\boldsymbol{n}} y^*]_\Gamma = g_1 + u^* \text{ on } \Gamma,\ y^* = h_0 \text{ on } \partial\Omega, \tag{5.4}$$

$$-\nabla\cdot(\beta\nabla p^*) = y^* - y_d \text{ in } \Omega\backslash\Gamma,\ [p^*]_\Gamma = 0,\ [\beta\partial_{\boldsymbol{n}} p^*]_\Gamma = 0 \text{ on } \Gamma,\ p^* = 0 \text{ on } \partial\Omega. \tag{5.5}$$

It is easy to see that problem (5.1) is convex and hence the optimality system (5.3) is also sufficient. Substituting (5.3) into (5.4) yields

$$\begin{aligned} -\nabla\cdot(\beta\nabla y^*) &= f \text{ in } \Omega\backslash\Gamma,\ [y^*]_\Gamma = g_0,\ [\beta\partial_{\boldsymbol{n}} y^*]_\Gamma \\ &= g_1 + \mathcal{P}_{U_{ad}}\Big(-\frac{p^*}{\alpha}\Big|_\Gamma (x)\Big) \text{ on } \Gamma,\ y^* = h_0 \text{ on } \partial\Omega. \end{aligned} \tag{5.6}$$

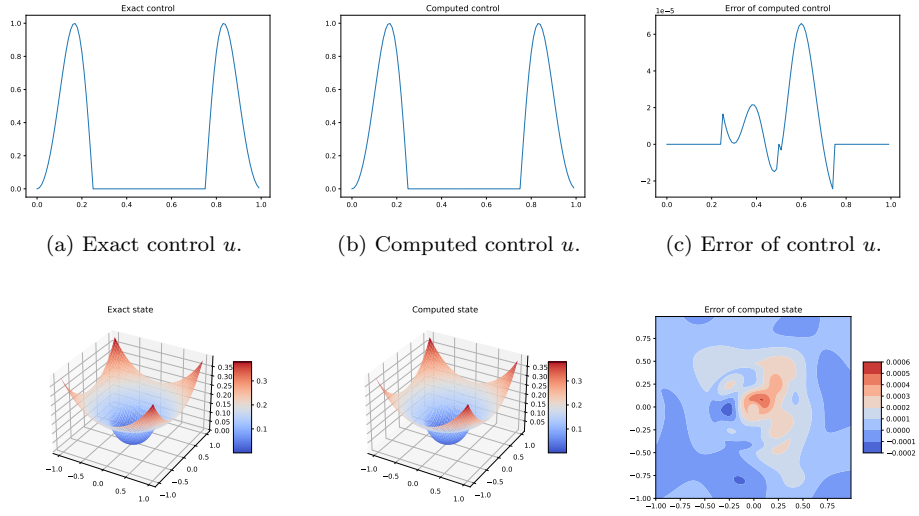
Therefore, solving (5.1)–(5.2) is equivalent to solving (5.5) and (5.6) simultaneously.

Next, we demonstrate the extension of Algorithm 3.2 to solve problem (5.1)–(5.2). First, the neural networks $\hat{y}$ and $\hat{p}$ for approximating y and p are constructed in the same way as that in section 3; see (3.8) and (3.12). The loss function is now defined as

$$\begin{aligned} \mathcal{L}_{HC}(\theta_y, \theta_p) = &\frac{w_{y,r}}{M}\sum_{i=1}^{M}\Big| -\Delta_x\hat{y}(x^i;\theta_y) - \frac{f(x^i)}{\beta^\pm}\Big|^2 \\ &+ \frac{w_{y,\Gamma_n}}{M_\Gamma}\sum_{i=1}^{M_\Gamma}\Big|[\beta\partial_{\boldsymbol{n}}\hat{y}]_\Gamma(x_\Gamma^i;\theta_y) - g_1(x_\Gamma^i) - \mathcal{P}_{[u_a(x_\Gamma^i),u_b(x_\Gamma^i)]}(-\frac{1}{\alpha}\hat{p}(x^i;\theta_p))\Big|^2 \\ &+ \frac{w_{p,r}}{M}\sum_{i=1}^{M}\Big| -\Delta_x\hat{p}(x_i;\theta_p) - \frac{\hat{y}(x_i;\theta_y) - y_d(x_i)}{\beta^\pm}\Big|^2 \\ &+ \frac{w_{p,\Gamma_n}}{M_\Gamma}\sum_{i=1}^{M_\Gamma}\big|[\beta\partial_{\boldsymbol{n}}\hat{p}]_\Gamma(x_i^\Gamma;\theta_p)\big|^2. \end{aligned} \tag{5.7}$$

Then we can easily obtain the hard-constraint PINN method for (5.1)–(5.2) and we omit the details for succinctness.

Example 3. We consider an example of (5.1)–(5.2) with $\alpha = 1$, $\beta^- = 1$, $\beta^+ = 10$, $u \in U_{ad} = \{u \in L^2(\Gamma) : \sin(2\pi x_1) \le u(x) \le 1 \text{ a.e. on } \Gamma\}$, and $u^*(x_1, x_2) =$



(a) Exact control u. (b) Computed control u. (c) Error of control u.

(d) Exact state y. (e) Computed state y. (f) Error of state y.

FIG. 6. *Numerical results of the hard-constraint PINN method for Example* 3.

$\max\{\sin(2\pi x_1), \min\{1, -\frac{1}{\alpha}p^*|_\Gamma(x_1,x_2)\}\}$. The rest of the settings are the same as those in Example 1. Then we can see that $(u^*, y^*)^\top$, with y^* defined in (4.3), is the solution to this problem.

The neural networks $\mathcal{N}_y(x, \phi(x); \theta_y), \mathcal{N}_p(x, \phi(x); \theta_p)$ and the functions g, h, and ϕ are the same as those in Example 1. Moreover, we randomly sample the training sets $\mathcal{T} = \{x^i\}_{i=1}^{1024} \subset \Omega$ and $\mathcal{T}_\Gamma = \{x_\Gamma^i\}_{i=1}^{256} \subset \Gamma$. We initialize the neural network parameters θ_y, θ_p following the default settings of PyTorch. All the weights in (5.7) are set to be 1.

We employ 40,000 iterations of Adam to train the neural networks, and the parameters θ_y and θ_p are updated simultaneously in each iteration. The learning rate is set to be 5×10^{-3} in the first $5,000$ iterations, then 1×10^{-3} in the $5,001$ to $15,000$ iterations, then 5×10^{-4} in the $15,001$ to $30,000$ iterations, and finally 1×10^{-4} in the $30,001$ to $40,000$ iterations. The strategy for computing the L^2-error of the control is similar to (4.1), except that now we sample the testing points from the interface Γ. We use the same method as that in Example 1 for visualizing y and its error. For the computed control u, we present its graph along the interface circle $\{(\cos(2\pi\theta), \sin(2\pi\theta)) \in \mathbb{R}^2 : 0 \le \theta \le 1\}$ with respect to the angle parameter θ.

The numerical results are shown in Figure 6. It can be observed that the maximum absolute errors of y and u obtained by the hard-constraint PINN method are approximately 6×10^{-4} and 7×10^{-5}, respectively. Moreover, the L^2-errors of the computed control u are $\varepsilon_{\text{abs}} = 2.0386 \times 10^{-5}$ and $\varepsilon_{\text{rel}} = 7.8737 \times 10^{-5}$. These results show that the proposed hard-constraint PINN method is also efficient for (5.1)–(5.2), producing numerical solutions with high accuracy.

5.2. A parabolic interface optimal control problem. In this subsection, we discuss the extension of Algorithm 3.2 to time-dependent problems. To this end, let $\Omega \subset \mathbb{R}^d$ $(d = 2, 3)$ be a bounded domain and $\Gamma \subset \Omega$ be the same interface as the one defined in section 2. Consider the following optimal control problem:

(5.8)
$$\min_{y\in L^2(\Omega\times(0,T)),u\in L^2(\Omega\times(0,T))} J(y,u) := \frac{1}{2}\int_0^T\int_\Omega (y-y_d)^2dx + \frac{\alpha}{2}\int_0^T\int_\Omega u^2dx$$
$$\text{s.t.}\begin{cases} \dfrac{\partial y}{\partial t} - \nabla\cdot(\beta\nabla y) = u+f & \text{in } (\Omega\backslash\Gamma)\times(0,T), \quad y=h_0 \quad \text{on } \partial\Omega\times(0,T),\\ [y]_\Gamma = g_0,\ [\beta\partial_{\boldsymbol{n}} y]_\Gamma = g_1 & \text{on } \Gamma\times(0,T), \quad y(0)=y_0 \quad \text{in } \Omega. \end{cases}$$

Above, the final time $T>0$ is a fixed constant, the function $y_d\in L^2(\Omega\times(0,T))$ is the target, and the constant $\alpha>0$ is a regularization parameter. The functions f, g_0, g_1, h_0, and y_0 are given, and β is a positive piecewise-constant function like the one defined in section 2. The interface Γ is assumed to be time-invariant. We also set the following constraint for the control variable:

$$u\in U_{ad} := \{u\in L^2(\Omega\times(0,T)) : u_a\le u\le u_b \text{ a.e. in } \Omega\times(0,T)\}, \tag{5.9}$$

where $u_a, u_b\in L^2(\Omega\times(0,T))$. By [57], we have the following results.

THEOREM 5.2 (cf. [57]). *Problem* (5.8)–(5.9) *admits a unique optimal control* $(u^*,y^*)^\top\in U_{ad}\times L^2(\Omega\times(0,T))$*, and the following first-order optimality system holds:*

$$u^* = \mathcal{P}_{U_{ad}}\Big(-\frac{1}{\alpha}p^*\Big), \tag{5.10}$$

where $\mathcal{P}_{U_{ad}}(\cdot)$ *denotes the projection onto* U_{ad}*, and* p^* *is the adjoint variable associated with* u^* *which is obtained from the successive solution of the following two equations:*

(5.11)
$$\begin{cases} \dfrac{\partial y^*}{\partial t} - \nabla\cdot(\beta\nabla y^*) = u^*+f & \text{in } (\Omega\backslash\Gamma)\times(0,T), \quad y^*=h_0 \quad \text{on } \partial\Omega\times(0,T),\\ [y^*]_\Gamma = g_0, \quad [\beta\partial_{\boldsymbol{n}} y^*]_\Gamma = g_1 & \text{on } \Gamma\times(0,T), \quad y^*(0)=y_0 \quad \text{in } \Omega, \end{cases}$$

(5.12)
$$\begin{cases} -\dfrac{\partial p^*}{\partial t} - \nabla\cdot(\beta\nabla p^*) = y^*-y_d & \text{in } (\Omega\backslash\Gamma)\times(0,T), \quad p^*=0 \quad \text{on } \partial\Omega\times(0,T),\\ [p^*]_\Gamma = 0, \quad [\beta\partial_{\boldsymbol{n}} p^*]_\Gamma = 0 & \text{on } \Gamma\times(0,T), \quad p^*(T)=0 \quad \text{in } \Omega. \end{cases}$$

Problem (5.8)–(5.9) is convex, and hence the solution to (5.8)–(5.9) can be obtained by simultaneously solving (5.11) and (5.12). Next, we delineate the extension of Algorithm 3.2 to problem (5.8)–(5.9). For this purpose, let $\mathcal{N}_y(x,t,\phi(x);\theta_y)$ and $\mathcal{N}_p(x,t,\phi(x);\theta_p)$ be two neural networks with smooth activation functions; we then approximate the solutions of (5.11) and (5.12) by

$$\begin{aligned} \hat{y}(x,t;\theta_y) &= g(x,t) + th(x)\mathcal{N}_y(x,t,\phi(x);\theta_y),\\ \hat{p}(x,t;\theta_p) &= (T-t)h(x)\mathcal{N}_p(x,t,\phi(x);\theta_p). \end{aligned} \tag{5.13}$$

Here $\phi(x)$ is an auxiliary function satisfying (3.9), $h(x)$ is a function satisfying (3.5), and both of them are independent of the variable t since the interface Γ and the boundary $\partial\Omega$ are time-invariant. The function g satisfies

$$g = h_0 \text{ on } \partial\Omega\times(0,T), \quad g(0)=y_0 \text{ in } \Omega, \quad [g]_\Gamma = g_0 \text{ on } \Gamma\times(0,T). \tag{5.14}$$

Then, using the same arguments as those in subsection 3.2, it is easy to check that $\hat{y}(x,t;\theta_y)$ and $\hat{p}(x,t;\theta_p)$ strictly satisfy the interface, boundary, and initial conditions

in (5.11)–(5.12) if functions g, h, and ϕ are given in analytic expressions. Moreover, we reiterate that, following the discussions in subsection 3.2, the functions g, h, and ϕ can be constructed in analytic forms or by neural networks.

To train the neural networks $\hat{y}(x,t;\theta_y)$ and $\hat{p}(x,t;\theta_p)$, we sample the training sets $\mathcal{T} = \{(x^i,t^i)\}_{i=1}^{M} \subset \Omega \times (0,T)$ and $\mathcal{T}_\Gamma = \{x_\Gamma^i, t_\Gamma^i\}_{i=1}^{M_\Gamma} \subset \Gamma \times (0,T)$, and we consider the following loss function:

(5.15)

$$
\begin{aligned}
&\mathcal{L}_{HC}(\theta_y,\theta_p)\\
&\quad= \frac{w_{y,r}}{M}\sum_{i=1}^{M}\left|\frac{\partial \hat{y}(x^i,t^i;\theta_y)}{\partial t} - \Delta_x \hat{y}(x^i,t^i;\theta_y)\right.\\
&\qquad\left. - \frac{\mathcal{P}_{[u_a(x^i,t^i),u_b(x^i,t^i)]}(-\frac{\hat{p}}{\alpha}(x^i,t^i;\theta_p)+f(x^i,t^i))}{\beta^{\pm}}\right|^2\\
&\qquad+ \frac{w_{y,\Gamma_n}}{M_\Gamma}\sum_{i=1}^{M_\Gamma}\left|[\beta\partial_{\boldsymbol{n}}\hat{y}]_\Gamma(x_\Gamma^i,t_\Gamma^i;\theta_y) - g_1(x_\Gamma^i,t_\Gamma^i)\right|^2 + \frac{w_{p,\Gamma_n}}{M_\Gamma}\sum_{i=1}^{M_\Gamma}\left|[\beta\partial_{\boldsymbol{n}}\hat{p}]_\Gamma(x_\Gamma^i,t_\Gamma^i;\theta_p)\right|^2\\
&\qquad+ \frac{w_{p,r}}{M}\sum_{i=1}^{M}\left|-\frac{\partial \hat{p}(x^i,t^i;\theta_p)}{\partial t} - \Delta_x \hat{p}(x_i,t^i;\theta_p) - \frac{\hat{y}(x^i,t^i;\theta_y)-y_d(x_i,t^i)}{\beta^{\pm}}\right|^2.
\end{aligned}
$$

Then we can easily obtain the hard-constraint PINN method for solving (5.11)–(5.12) and hence problem (5.8)–(5.9).

Example 4. We test the hard-constraint PINN method for solving (5.8)–(5.9) with $\Omega = (-1,1)\times(-1,1) \subset \mathbb{R}^2$, $\Gamma = \{x \in \Omega : \|x\|_2 \le r_0\}$, $r_0 = 0.5$, and $T = \pi/2$. The admissible set $U_{ad} = \{u \in L^2(\Omega\times(0,T)) : -1 \le u \le 1 \text{ a.e. in } \Omega\times(0,T)\}$. We further set $\alpha = 1$, $\beta^- = 1$, $\beta^+ = 3$, $g_0 = g_1 = 0$, $h_0 = 0$, and $y_0 = 0$.

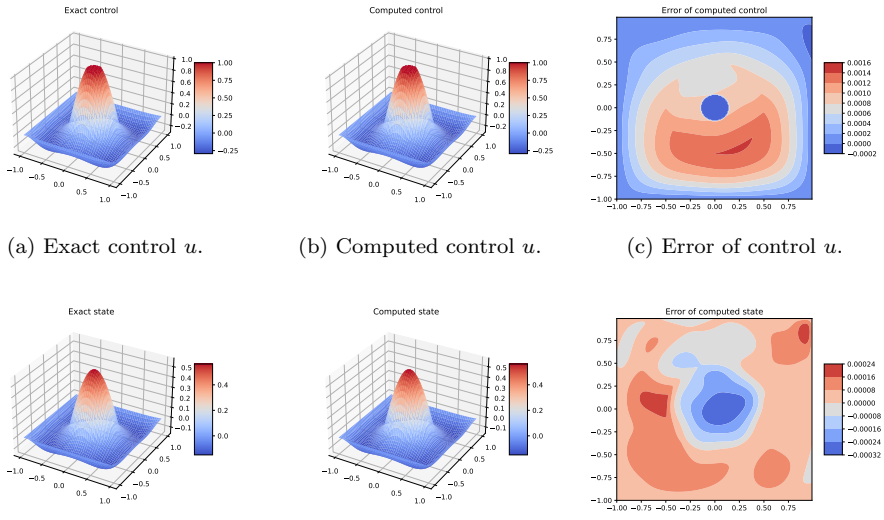
Following [57], we let

$$
\begin{cases}
p^*(x_1,x_2,t) = 5\sin(T-t)(x_1^2+x_2^2-r_0^2)(x_1^2-1)(x_2^2-1)/\beta^{\pm} \quad \text{in } \Omega^{\pm},\\
y^*(x_1,x_2,t) = 5\cos(t-T)(x_1^2+x_2^2-r_0^2)(x_1^2-1)(x_2^2-1)/\beta^{\pm} \quad \text{in } \Omega^{\pm},\\
u^*(x_1,x_2,t) = \max\left\{-1, \min\left\{1, -\dfrac{1}{\alpha}p^*(x_1,x_2,t)\right\}\right\},\\
f(x_1,x_2,t) = \dfrac{\partial y^*}{\partial t}(x_1,x_2,t) - u^*(x_1,x_2,t) - \nabla\cdot(\beta^{\pm}\nabla y^*(x_1,x_2,t)) \text{ in } \Omega^{\pm},\\
y_d(x_1,x_2,t) = \dfrac{\partial p^*}{\partial t}(x_1,x_2,t) + y^*(x_1,x_2,t) + \nabla\cdot(\beta^{\pm}\nabla p^*(x_1,x_2,t)) \text{ in } \Omega^{\pm}.
\end{cases}
$$

Then it is easy to verify that $(u^*,y^*,p^*)^\top$ satisfies the optimality system (5.10)–(5.12), and hence $(u^*,y^*)^\top$ is the solution of (5.8)–(5.9).

We construct two neural networks $\mathcal{N}_y(x,t,\phi(x);\theta_y)$ and $\mathcal{N}_p(x,t,\phi(x);\theta_p)$ consisting of three hidden layers with 100 neurons. The state and adjoint variables are approximated by (5.13) with $g(x,t) = 0$ and $h(x) = (x_1^2-1)(x_2^2-1)$. The auxiliary function ϕ is chosen like that in (4.5).

To evaluate the loss function (5.15), we first select $\{t_i\}_{i=1}^{16}$ by the Chebyshev sampling over $(0,T)$. Then we randomly sample $\{x^i\}_{i=1}^{256} \subset \Omega$ and $\{x_\Gamma^i\}_{i=1}^{64} \subset \Gamma$. After that, we take the Cartesian product of $\{t_i\}_{i=1}^{16}$ and $\{x^i\}_{i=1}^{256}$ and $\{x_\Gamma^i\}_{i=1}^{64}$, respectively, to generate the training sets $\mathcal{T} = \{(x^i,t^i)\}_{i=1}^{4096} \subset \Omega\times(0,T)$ and $\mathcal{T}_\Gamma = \{(x_\Gamma^i,t_\Gamma^i)\}_{i=1}^{1024} \subset \Gamma\times(0,T)$. We initialize the neural network parameters θ_y and θ_p following the default settings in PyTorch. The weights in (5.15) are all taken to be 1. We implement 40,000



(a) Exact control u. (b) Computed control u. (c) Error of control u.

(d) Exact state y. (e) Computed state y. (f) Error of state y.

FIG. 7. *Numerical results of the hard-constraint PINN method for Example* 4.

iterations of the Adam to train the neural networks, and the parameters θ_y and θ_p are optimized simultaneously in each iteration. The learning rate is set to be 3×10^{-3} in the first $3,000$ iterations, then 1×10^{-3} in the $3,001$ to $10,000$ iterations, 5×10^{-4} in the $10,001$ to $20,000$ iterations, and finally 1×10^{-4} in the $20,001$ to $40,000$ iterations.

The computed results at $t = 0.3T$ are presented in Figure 7. We can see that the hard-constraint PINN method is capable of dealing with time-dependent problems and a high-accurate numerical solution can be pursued.

6. Conclusions and perspectives. This paper explores the application of the physics-informed neural networks (PINNs) to optimal control problems subject to PDEs with interfaces and control constraints. We first demonstrate that leveraged by the discontinuity capturing neural networks [18], PINNs can effectively solve such problems. However, the boundary and interface conditions, along with the PDE, are treated as soft constraints by incorporating them into a weighted loss function. Hence, the boundary and interface conditions cannot be satisfied exactly and must be simultaneously learned with the PDE. This makes it difficult to fine-tune the weights and to train the neural networks, resulting in a loss of numerical accuracy. To overcome these issues, we propose a novel neural network architecture designed to impose the boundary and interface conditions as hard constraints. The resulting hard-constraint PINNs guarantee that both the boundary and the interface conditions are satisfied exactly or with a high degree of accuracy, while being independent of the learning process for the PDEs. This hard-constraint approach significantly simplifies the training process and enhances the numerical accuracy. Moreover, the hard-constraint PINNs are mesh-free, easy to implement, and scalable to different PDEs, and they can ensure rigorous satisfaction of the control constraints. To validate the effectiveness of the proposed hard-constraint PINNs, we conduct extensive tests on various elliptic and parabolic interface optimal control problems.

Our work leaves some important questions for future research. For instance, the high efficiency of the hard-constraint PINNs for interface optimal control problems

emphasizes the necessity for convergence analysis and error estimates. In the numerical experiments, we adopted fixed weights in the loss function and nonadaptive sampling methods for the training points, which may not be optimal. It is worth investigating adaptive weighting and sampling strategies (see, e.g., [13, 30, 32, 50, 52]) to further improve the numerical accuracy of the hard-constraint PINNs. In subsection 5.2, we discuss the extension of the hard-constraint PINNs to parabolic interface optimal control problems, where the interface is assumed to be time-invariant. A natural question is how to extend our discussions to the interfaces whose shape changes over time. Recall (3.10), where the interface-gradient condition $[\beta\partial_{\boldsymbol{n}} y]_\Gamma$ is still treated as a soft constraint. It is thus worth designing some more sophisticated neural networks such that this condition can also be imposed as a hard constraint and the numerical efficiency of PINNs can be further improved. Finally, note that we focus on Dirichlet boundary conditions and it would be interesting to design some novel neural networks such that other types of boundary conditions (e.g., periodic conditions and Neumann conditions) can be treated as hard constraints; the ideas in [31] and [44] could be useful.

REFERENCES

[1] I. Babuška, *The finite element method for elliptic equations with discontinuous coefficients*, Computing, 5 (1970), pp. 207–213, https://doi.org/10.1007/BF02248021.

[2] J. Barry-Straume, A. Sarshar, A. A. Popov, and A. Sandu, *Physics-Informed Neural Networks for PDE-constrained Optimization and Control*, preprint, arXiv:2205.03377, 2022.

[3] U. Biccari, Y. Song, X. Yuan, and E. Zuazua, *A two-stage numerical approach for the sparse initial source identification of a diffusion–advection equation*, Inverse Problems, 39 (2023), 095003, https://doi.org/10.1088/1361-6420/ace548.

[4] S. N. Cohen, D. Jiang, and J. Sirignano, *Neural Q-learning for solving PDEs*, J. Mach. Learn. Res., 24 (2023), pp. 1–49.

[5] G. Cybenko, *Approximation by superpositions of a sigmoidal function*, Math. Control Signals Systems, 2 (1989), pp. 303–314, https://doi.org/10.1007/BF02551274.

[6] J. C. De los Reyes, *Numerical PDE-Constrained Optimization*, Springer, 2015.

[7] W. E and B. Yu, *The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems*, Commun. Math. Stat., 6 (2018), pp. 1–12, https://doi.org/10.1007/s40304-018-0127-z.

[8] W. Geng, S. Yu, and G. Wei, *Treatment of charge singularities in implicit solvent models*, J. Chem. Phys., 127 (2007), 114106, https://doi.org/10.1063/1.2768064.

[9] R. Glowinski, Y. Song, X. Yuan, and H. Yue, *Application of the alternating direction method of multipliers to control constrained parabolic optimal control problems and beyond*, Ann. Appl. Math., 38 (2022), pp. 115–158.

[10] Y. Gong, B. Li, and Z. Li, *Immersed-interface finite-element methods for elliptic interface problems with nonhomogeneous jump conditions*, SIAM J. Numer. Anal., 46 (2008), pp. 472–495, https://doi.org/10.1137/060666482.

[11] G. Gripenberg, *Approximation by neural networks with a bounded number of nodes at each level*, J. Approx. Theory, 122 (2003), pp. 260–266, https://doi.org/10.1016/S0021-9045(03)00078-9.

[12] H. Guo and X. Yang, *Deep unfitted Nitsche method for elliptic interface problems*, Commun. Comput. Phys., 31 (2022), pp. 1162–1179, https://doi.org/10.4208/cicp.OA-2021-0201.

[13] Z. Hao, S. Liu, Y. Zhang, C. Ying, Y. Feng, H. Su, and J. Zhu, *Physics-Informed Machine Learning: A Survey on Problems, Methods and Applications*, preprint, arXiv:2211.08064, 2022.

[14] C. He, X. Hu, and L. Mu, *A mesh-free method using piecewise deep neural network for elliptic interface problems*, J. Comput. Appl. Math., 412 (2022), 114358, https://doi.org/10.1016/j.cam.2022.114358.

[15] M. Hinze, R. Pinnau, M. Ulbrich, and S. Ulbrich, *Optimization with PDE Constraints*, Math. Model. Theory Appl. 23, Springer, New York, 2009.

[16] K. Hornik, *Approximation capabilities of multilayer feedforward networks*, Neural Networks, 4 (1991), pp. 251–257, https://doi.org/10.1016/0893-6080(91)90009-T.

[17] T. Y. Hou, Z. Li, S. Osher, and H. Zhao, *A hybrid method for moving interface problems with application to the Hele–Shaw flow*, J. Comput. Phys., 134 (1997), pp. 236–252, https://doi.org/10.1006/jcph.1997.5689.

[18] W.-F. Hu, T.-S. Lin, and M.-C. Lai, *A discontinuity capturing shallow neural network for elliptic interface problems*, J. Comput. Phys., 469 (2022), 111576, https://doi.org/10.1016/j.jcp.2022.111576.

[19] W.-F. Hu, T.-S. Lin, Y.-H. Tseng, and M.-C. Lai, *An efficient neural-network and finite-difference hybrid method for elliptic interface problems with applications*, Commun. Comput. Phys., 33 (2023), pp. 1090–1105, https://doi.org/10.4208/cicp.OA-2022-0284.

[20] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, *Physics-informed machine learning*, Nature Rev. Phys., 3 (2021), pp. 422–440, https://doi.org/10.1038/s42254-021-00314-5.

[21] E. Kharazmi, Z. Zhang, and G. E. Karniadakis, *Variational Physics-Informed Neural Networks for Solving Partial Differential Equations*, preprint, arXiv:1912.00873, 2019.

[22] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, in Proceedings of the 3rd International Conference on Learning Representations, 2015.

[23] P. L. Lagari, L. H. Tsoukalas, S. Safarkhani, and I. E. Lagaris, *Systematic construction of neural forms for solving partial differential equations inside rectangular domains, subject to initial, boundary and interface conditions*, Int. J. Artif. Intell. Tools, 29 (2020), 2050009, https://doi.org/10.1142/S0218213020500098.

[24] I. E. Lagaris, A. Likas, and D. I. Fotiadis, *Artificial neural networks for solving ordinary and partial differential equations*, IEEE Trans. Neural Networks, 9 (1998), pp. 987–1000, https://doi.org/10.1109/72.712178.

[25] A. T. Layton, *Using integral equations and the immersed interface method to solve immersed boundary problems with stiff forces*, Comput. Fluids, 38 (2009), pp. 266–272, https://doi.org/10.1016/j.compfluid.2008.02.003.

[26] J. M. Lee, *Introduction to Smooth Manifolds*, Grad. Texts in Math. 218, Springer, New York, 2013.

[27] Z. Li, *A fast iterative algorithm for elliptic interface problems*, SIAM J. Numer. Anal., 35 (1998), pp. 230–254, https://doi.org/10.1137/S0036142995291329.

[28] Z. Li and K. Ito, *The Immersed Interface Method: Numerical Solutions of PDEs Involving Interfaces and Irregular Domains*, SIAM, Philadelphia, 2006, https://doi.org/10.1137/1.9780898717464.

[29] J. L. Lions, *Optimal Control of Systems Governed by Partial Differential Equations*, Grundlehren Math. Wiss. 170, Springer-Verlag, New York, Berlin, 1971.

[30] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, *DeepXDE: A deep learning library for solving differential equations*, SIAM Rev., 63 (2021), pp. 208–228, https://doi.org/10.1137/19M1274067.

[31] L. Lu, R. Pestourie, W. Yao, Z. Wang, F. Verdugo, and S. G. Johnson, *Physics-informed neural networks with hard constraints for inverse design*, SIAM J. Sci. Comput., 43 (2021), pp. B1105–B1132, https://doi.org/10.1137/21M1397908.

[32] L. D. McClenny and U. M. Braga-Neto, *Self-Adaptive physics-informed neural networks*, J. Comput. Phys., 474 (2023), 111722, https://doi.org/10.1016/j.jcp.2022.111722.

[33] M. D. McKay, R. J. Beckman, and W. J. Conover, *A comparison of three methods for selecting values of input variables in the analysis of output from a computer code*, Technometrics, 42 (2000), pp. 55–61, https://doi.org/10.1080/00401706.2000.10485979.

[34] C. Meyer, P. Philip, and F. Tröltzsch, *Optimal control of a semilinear PDE with nonlocal radiation interface conditions*, SIAM J. Control Optim., 45 (2006), pp. 699–721, https://doi.org/10.1137/040617753.

[35] S. Mowlavi and S. Nabi, *Optimal control of PDEs using physics-informed neural networks*, J. Comput. Phys., 473 (2023), 111731, https://doi.org/10.1016/j.jcp.2022.111731.

[36] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein, *On the expressive power of deep neural networks*, in Proceedings of the 34th International Conference on Machine Learning, PMLR, 2017, pp. 2847–2854.

[37] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, J. Comput. Phys., 378 (2019), pp. 686–707, https://doi.org/10.1016/j.jcp.2018.10.045.

[38] H. Sheng and C. Yang, *PFNN: A penalty-free neural network method for solving a class of second-order boundary-value problems on complex geometries*, J. Comput. Phys., 428 (2021), 110085, https://doi.org/10.1016/j.jcp.2020.110085.

[39] J. Sirignano and K. Spiliopoulos, *DGM: A deep learning algorithm for solving partial differential equations*, J. Comput. Phys., 375 (2018), pp. 1339–1364, https://doi.org/10.1016/j.jcp.2018.08.029.

[40] Y. Song, X. Yuan, and H. Yue, *An inexact Uzawa algorithmic framework for nonlinear saddle point problems with applications to elliptic optimal control problem*, SIAM J. Numer. Anal., 57 (2019), pp. 2656–2684, https://doi.org/10.1137/19M1245736.

[41] Y. Song, X. Yuan, and H. Yue, *Accelerated Primal-Dual Methods with Enlarged Step Sizes and Operator Learning for Nonsmooth Optimal Control Problems*, preprint, arXiv:2307.00296, 2023.

[42] Y. Song, X. Yuan, and H. Yue, *The ADMM-PINNs algorithmic framework for nonsmooth PDE-constrained optimization: A deep learning approach*, SIAM J. Sci. Comput., 46 (2024), pp. C659–C687, https://doi.org/10.1137/23M1566935.

[43] M. Su and Z. Zhang, *Numerical approximation based on immersed finite element method for elliptic interface optimal control problem*, Commun. Nonlinear Sci., 120 (2023), 107195, https://doi.org/10.1016/j.cnsns.2023.107195.

[44] N. Sukumar and A. Srivastava, *Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks*, Comput. Methods Appl. Mech. Engrg., 389 (2022), 114333, https://doi.org/10.1016/j.cma.2021.114333.

[45] Q. Sun, X. Xu, and H. Yi, *Dirichlet-Neumann Learning Algorithm for Solving Elliptic Interface Problems*, preprint, arXiv:2301.07361, 2023.

[46] F. Tröltzsch, *Optimal Control of Partial Differential Equations: Theory, Methods, and Applications*, Grad. Stud. Math. 112, American Mathematical Society, Providence, RI, 2010.

[47] Y.-H. Tseng, T.-S. Lin, W.-F. Hu, and M.-C. Lai, *A cusp-capturing PINN for elliptic interface problems*, J. Comput. Phys., 491 (2023), 112359, https://doi.org/10.1016/j.jcp.2023.112359.

[48] M. Ulbrich, *Semismooth Newton Methods for Variational Inequalities and Constrained Optimization Problems in Function Spaces*, SIAM, Philadelphia, 2011, https://doi.org/10.1137/1.9781611970692.

[49] D. Wachsmuth and J.-E. Wurst, *Optimal control of interface problems with hp-finite elements*, Numer. Funct. Anal. Optim., 37 (2016), pp. 363–390, https://doi.org/10.1080/01630563.2016.1149014.

[50] S. Wang, Y. Teng, and P. Perdikaris, *Understanding and mitigating gradient flow pathologies in physics-informed neural networks*, SIAM J. Sci. Comput., 43 (2021), pp. A3055–A3081, https://doi.org/10.1137/20M1318043.

[51] Z. Wang and Z. Zhang, *A mesh-free method for interface problems using the deep learning approach*, J. Comput. Phys., 400 (2020), 108963, https://doi.org/10.1016/j.jcp.2019.108963.

[52] C. Wu, M. Zhu, Q. Tan, Y. Kartha, and L. Lu, *A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks*, Comput. Methods Appl. Mech. Engrg., 403 (2023), 115671, https://doi.org/10.1016/j.cma.2022.115671.

[53] S. Wu and B. Lu, *INN: Interfaced neural networks as an accessible meshless approach for solving interface PDE problems*, J. Comput. Phys., 470 (2022), 111588, https://doi.org/10.1016/j.jcp.2022.111588.

[54] C. Yang, T. Wang, and X. Xie, *An interface-unfitted finite element method for elliptic interface optimal control problems*, Numer. Math. Theor. Meth. Appl., 12 (2019), pp. 727–749, https://doi.org/10.4208/nmtma.OA-2018-0031.

[55] Q. Zhang, K. Ito, Z. Li, and Z. Zhang, *Immersed finite elements for optimal control problems of elliptic PDEs with interfaces*, J. Comput. Phys., 298 (2015), pp. 305–319, https://doi.org/10.1016/j.jcp.2015.05.050.

[56] Q. Zhang, T. Zhao, and Z. Zhang, *Unfitted finite element for optimal control problem of the temperature in composite media with contact resistance*, Numer. Algorithms, 84 (2020), pp. 165–180, https://doi.org/10.1007/s11075-019-00750-6.

[57] Z. Zhang, D. Liang, and Q. Wang, *Immersed finite element method and its analysis for parabolic optimal control problems with interfaces*, Appl. Numer. Math., 147 (2020), pp. 174–195, https://doi.org/10.1016/j.apnum.2019.08.024.