Journal of Computational Physics 491 (2023) 112359

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

A cusp-capturing PINN for elliptic interface problems

Yu-Hau Tseng [a], Te-Sheng Lin [b,d,*], Wei-Fan Hu [c,d], Ming-Chih Lai [b]

[a] Department of Applied Mathematics, National University of Kaohsiung, Kaohsiung 81148, Taiwan
[b] Department of Applied Mathematics, National Yang Ming Chiao Tung University, Hsinchu 30010, Taiwan
[c] Department of Mathematics, National Central University, Taoyuan 32001, Taiwan
[d] National Center for Theoretical Sciences, National Taiwan University, Taipei 10617, Taiwan

ARTICLE INFO

Article history:
Received 15 October 2022
Received in revised form 16 April 2023
Accepted 6 July 2023
Available online 13 July 2023

Keywords:
Fluid structure interaction problem
Elliptic interface problem
Level set function
Cusp capturing

ABSTRACT

In this paper, we propose a cusp-capturing physics-informed neural network (PINN) to solve discontinuous-coefficient elliptic interface problems whose solution is continuous but has discontinuous first derivatives on the interface. To find such a solution using neural network representation, we introduce a cusp-enforced level set function as an additional feature input to the network to retain the inherent solution properties; that is, capturing the solution cusps (where the derivatives are discontinuous) sharply. In addition, the proposed neural network has the advantage of being mesh-free, so it can easily handle problems in irregular domains. We train the network using the physics-informed framework in which the loss function comprises the residual of the differential equation together with certain interface and boundary conditions. We conduct a series of numerical experiments to demonstrate the effectiveness of the cusp-capturing technique and the accuracy of the present network model. Numerical results show that even using a one-hidden-layer (shallow) network with a moderate number of neurons and sufficient training data points, the present network model can achieve prediction accuracy comparable with traditional methods. Besides, if the solution is discontinuous across the interface, we can simply incorporate an additional supervised learning task for solution jump approximation into the present network without much difficulty.

1. Introduction

The study of fluid-structure interaction (FSI) problems has been an important research topic in fluid dynamics for centuries, with applications ranging from, for example, fundamental physics, engineering, geophysics, and biomedicine. Typical small-scale examples include collisions between droplets in interfacial flows [28,37], the dynamics of red blood cells flowing in pulsating arteries [16,36], and the electrophoretic motion of colloidal particles in electrically charged fluids [11,27]. The key components in these examples are fluid flow, deformable interfaces, and the complex mechanisms behind them. Moreover, physical parameters (such as viscosity or density) for each subregion of the domain may be different, resulting in lower regularity of the solution across the interfaces, thus requiring additional treatments for accurate simulations.

For instance, when the no-slip boundary condition is applied to a fluid-structure interface, the velocity field in the FSI problem is continuous in the entire domain, but its derivative is discontinuous across the interface. Among many classical

* Corresponding author at: Department of Applied Mathematics, National Yang Ming Chiao Tung University, Hsinchu 30010, Taiwan.
E-mail addresses: yhtseng@go.nuk.edu.tw (Y.-H. Tseng), tslin@math.nctu.edu.tw (T.-S. Lin), wfhu@math.ncu.edu.tw (W.-F. Hu), mclai@math.nctu.edu.tw (M.-C. Lai).

https://doi.org/10.1016/j.jcp.2023.112359
0021-9991/

numerical methods for solving such problems, Peskin proposed the immersed boundary (IB) formulation [29,31], which transforms the core of solving the velocity field into an elliptic problem with singular forces. The IB method adopts a regularized version of the Dirac delta function to discretize the singular forces directly, resulting in only first-order solution accuracy [23]. Another way to write the velocity equations is to impose jump conditions directly on the interface. So the problem becomes an elliptic interface problem in which the solution is continuous, but its normal derivative has jump discontinuity across the interface, which is exactly the formulation we aim to solve in this work.

Since the introduction of the IB formulation, several jump-capturing and high-order methods have been proposed for elliptic interface problems with discontinuous coefficients. For instance, LeVeque and Li introduced the immersed interface method (IIM) [20], incorporating the jump conditions via local coordinates into the finite difference scheme to achieve the overall second-order accuracy in maximum norm. A simple implementation version of IIM that directly uses the jump conditions without introducing local coordinates was developed in [13,19] to achieve second-order accuracy in maximum norm as well. Liu et al. [22] introduced a boundary condition capturing method (also known as the ghost fluid method (GFM)) that is able to solve the elliptic interface problems in a dimension-by-dimension manner, and can capture the solution and its normal derivative jumps sharply. However, the original GFM smoothes its tangential derivative, so the method is only first-order accurate in the maximum norm. Egan and Gibou [6] extended the original GFM by recovering the convergence of the gradients to achieve second-order accuracy without modifying the resultant linear system. There are many other Cartesian grid-based methods to solve the above elliptic interface problems accurately and robustly; however, we do not intend to have an exhaustive review here.

Besides the grid-based methods described above, the scientific computing community has shown an increased interest in solving elliptic interface problems using shallow or deep neural networks. Notice that the neural network approach for solving the interface problems has one apparent advantage over the grid-based methods; namely, it is completely mesh-free and can easily handle problems with complex interfaces or irregular domains. One obstacle for the neural network approach is that most of the network has a smooth activation function, so the resulting network is inherently smooth and is not a suitable ansatz for the interface problem. We list some related works in literature as follows. A deep Nitsche-type method [21] to solve elliptic interface problems with high-contrast discontinuous coefficients was developed in [39]. To deal with inhomogeneous boundary conditions, a shallow neural network to approximate the boundary conditions must be employed in advance. In [9], the authors proposed a deep unfitted Nitsche method for solving elliptic interface problems with high contrasts in high dimensions. Unlike using a single network, Wu and Lu [40] proposed an interfaced neural network that decomposes the computational domain into two subdomains (one interface case), and each network is responsible for the solution on each subdomain. Then an extended multiple-gradient descent method was introduced to train the network. A similar piecewise deep neural network for elliptic interface problems was also introduced earlier in [10]. In the above neural network approaches, the network architectures usually have deep structures. Recently, the authors have proposed a discontinuity capturing shallow neural network (DCSNN) [14] for solving elliptic interface problems with discontinuous solutions. By augmenting a coordinate variable to label different pieces of each subdomain, the DCSNN can be trained in a single physics-informed neural network (PINN) framework [34]. Meanwhile, we also used the idea proposed by E and Yu [7] and developed a completely shallow Ritz network for solving the elliptic interface problems by augmenting the level set function as an extra feature input in [18]. We found that it significantly improves the training effectiveness and accuracy. Notice that the major difference between DCSNN [14] and the shallow Ritz network [18] is that the former inherently represents a discontinuous function while the latter represents a continuous one.

In this paper, we propose a cusp-capturing physics-informed neural network for solving discontinuous-coefficient elliptic interface problems. The specific aim of this study is to introduce a network that can present continuous solutions, but with discontinuous first derivatives on interfaces. The smooth level set function augmented input in [18] cannot capture the derivative discontinuity sharply; thus, we augment a cusp-enforced level set function input to the network instead. Notice that, this new modified level set function does not change the interface position (i.e., zero level set). The rest of the paper is organized as follows. We present the formulation of the discontinuous-coefficient elliptic interface problems in Section 2. In Section 3, we propose a cusp-capturing neural network to solve the model problems. Numerical experiments are shown in Section 4 to demonstrate the effectiveness of the proposed cusp-capturing technique and the accuracy of the present network, followed by some concluding remarks in Section 5.

2. Discontinuous-coefficient elliptic interface problems

We consider a d-dimensional discontinuous-coefficient second-order elliptic interface problem [2]. Let $\Omega \subset \mathbb{R}^d$ be a bounded domain and Γ be an embedded $(d-1)$-dimensional C^1-interface separating Ω into two subdomains, Ω^- and Ω^+, so $\Omega = \Omega^- \cup \Omega^+ \cup \Gamma$. The equations of the problem subjected to the interface and boundary conditions are given as follows:

$$\nabla \cdot (\beta(\mathbf{x})\nabla u(\mathbf{x})) - \alpha(\mathbf{x})u(\mathbf{x}) = f(\mathbf{x}), \quad \mathbf{x} \in \Omega^- \cup \Omega^+, \tag{1}$$

$$[\![u]\!](\mathbf{x}_\Gamma) = 0, \quad [\![\beta \partial_n u]\!](\mathbf{x}_\Gamma) = \rho(\mathbf{x}_\Gamma), \quad \mathbf{x}_\Gamma \in \Gamma, \tag{2}$$

$$u(\mathbf{x}_B) = g(\mathbf{x}_B), \quad \mathbf{x}_B \in \partial\Omega, \tag{3}$$

where $u(\mathbf{x})$ is the function to be solved, $\rho(\mathbf{x}_\Gamma)$ and $g(\mathbf{x}_B)$ are given smooth functions, $\alpha(\mathbf{x}) \geq 0$, $f(\mathbf{x})$ and $\beta(\mathbf{x}) > 0$ are also given but defined in a piecewise smooth manner across the interface Γ. We use $\partial_n u$ to denote the shorthand of normal

derivative $\nabla u \cdot \mathbf{n}$, where $\mathbf{n}$ is the unit normal vector pointing from Ω^- to Ω^+ along the interface Γ. The notation $[\![\cdot]\!]$ represents the jump of a quantity across the interface (the one-sided limiting value approaching from Ω^+ minus the one from Ω^-). For example,

$$[\![\beta]\!](\mathbf{x}_\Gamma) = \lim_{\mathbf{x}\in\Omega^+,\, \mathbf{x}\to\mathbf{x}_\Gamma} \beta(\mathbf{x}) - \lim_{\mathbf{x}\in\Omega^-,\, \mathbf{x}\to\mathbf{x}_\Gamma} \beta(\mathbf{x}) = \beta^+(\mathbf{x}_\Gamma) - \beta^-(\mathbf{x}_\Gamma), \tag{4}$$

where the superscripts "$\pm$" represent the limits of the function value on the interface. Under this notation, the second interface condition in Eq. (2) can be written explicitly as

$$\begin{aligned}[\![\beta\partial_n u]\!](\mathbf{x}_\Gamma) &= \beta^+(\mathbf{x}_\Gamma)\partial_n u^+(\mathbf{x}_\Gamma) - \beta^-(\mathbf{x}_\Gamma)\partial_n u^-(\mathbf{x}_\Gamma) \\ &= \beta^+(\mathbf{x}_\Gamma)\partial_n u^+(\mathbf{x}_\Gamma) - \beta^-(\mathbf{x}_\Gamma)\partial_n u^+(\mathbf{x}_\Gamma) + \beta^-(\mathbf{x}_\Gamma)\partial_n u^+(\mathbf{x}_\Gamma) - \beta^-(\mathbf{x}_\Gamma)\partial_n u^-(\mathbf{x}_\Gamma) \\ &= [\![\beta]\!](\mathbf{x}_\Gamma)\partial_n u^+(\mathbf{x}_\Gamma) + \beta^-(\mathbf{x}_\Gamma)[\![\partial_n u]\!](\mathbf{x}_\Gamma) = \rho(\mathbf{x}_\Gamma).\end{aligned} \tag{5}$$

One can immediately see that even with the case of $[\![\beta]\!](\mathbf{x}_\Gamma) = 0$, the solution u always has the property of $[\![\partial_n u]\!](\mathbf{x}_\Gamma) \neq 0$ as long as $\rho(\mathbf{x}_\Gamma) \neq 0$. Along with the first interface condition $[\![u]\!](\mathbf{x}_\Gamma) = 0$ in Eq. (2), we can conclude that the solution u is continuous over the domain Ω but its normal derivative has jump discontinuity across the interface Γ.

We would also like to point out that although here we focus only on the Dirichlet-type boundary condition (3), one can apply the present method to the Neumann or Robin-type boundary condition with no difficulty. In this paper, we aim to find the solution to Eqs. (1)-(3) using machine learning techniques in the spirit of physics-informed neural networks [34], as introduced in the next section.

3. A cusp-capturing physics-informed neural network

As mentioned before, the solution of Eqs. (1)-(3) is continuous in the domain Ω but has a jump discontinuity to its normal derivative on the interface Γ. The universal approximation theorems [4,12,32] guarantee the applicability of approximating such continuous solutions using artificial neural networks. However, a neural network with differentiable activation functions is undoubtedly smooth, thus it is unlikely to capture the present solution with cusps (the partial derivatives are not continuous) in an accurate manner. More precisely, locating and fitting derivative discontinuities in neural network solutions is challenging. Since the partial derivative jumps occur at the interface, it is natural to include the interface position as a feature input in the network architecture. In [18], we proposed a shallow Ritz-type method to solve similar interface problems (taking $\beta = 1$) as Eqs. (1)-(3) in which we add the level set function of the interface as a feature input to the network. That is, we use a neural network of the form $U(\mathbf{x}, z = \phi(\mathbf{x}))$ to approximate the solution $u(\mathbf{x})$ of the problem, where $\phi(\mathbf{x})$ is the level set function defined in the whole domain Ω. Here, the interior and exterior region are defined as $\Omega^- = \{\mathbf{x} \in \mathbb{R}^d \,|\phi(\mathbf{x}) < 0\}$ and $\Omega^+ = \{\mathbf{x} \in \mathbb{R}^d \,|\phi(\mathbf{x}) > 0\}$, respectively, and the zero level set gives the position of the interface Γ, i.e., $\Gamma = \{\mathbf{x} \in \mathbb{R}^d \,|\, \phi(\mathbf{x}) = 0\}$. With this level set function augmentation, we found that it significantly improves the training effectiveness and accuracy. However, since the level set function is smooth, and the neural network function U is smooth due to the use of a smooth activation function, the resulting neural network solution $u(\mathbf{x}) = U(\mathbf{x}, z) = U(\mathbf{x}, \phi(\mathbf{x}))$ remains smooth. That is, the gradient of u

$$\nabla u = \nabla_{\mathbf{x}} U + \partial_z U \, \nabla\phi, \tag{6}$$

is continuous so the normal derivative jump $[\![\partial_n u]\!] = 0$ across the interface Γ. Here, $\nabla_{\mathbf{x}} U \in \mathbb{R}^d$ represents a vector with partial derivatives of U with respect to the components in $\mathbf{x}$, and $\partial_z U$ is the partial derivative of U with respect to z. We also suppress the notation of $\mathbf{x}$ in the gradients of u and ϕ since they both are functions of $\mathbf{x}$. Thus, if we want to require ∇u to be discontinuous across the interface then $\nabla\phi$ should be discontinuous too. Therefore, we need to modify the original smooth level set function accordingly.

3.1. Cusp-enforced level set function augmentation

As mentioned above, we need to modify the level set function so that its gradient is discontinuous across the interface without changing the zero level set. This can be done easily by taking the absolute value of the level set function; that is, we define $\phi_a(\mathbf{x}) = |\phi(\mathbf{x})|$. We therefore call this ϕ_a as a cusp-enforced level set function since it is non-differentiable at the interface Γ. Furthermore, one can immediately derive that this cusp-enforced level set function has the gradient jump as $[\![\nabla\phi_a]\!](\mathbf{x}_\Gamma) = 2\nabla\phi(\mathbf{x}_\Gamma), \mathbf{x}_\Gamma \in \Gamma$. Note that, the above jump condition is evaluated by the limiting values from both sides of the interface where $\nabla\phi_a$ is well-defined. With this modified level set function, we now define a new neural network solution in the form as $u(\mathbf{x}) = U(\mathbf{x}, z) = U(\mathbf{x}, \phi_a(\mathbf{x}))$. Since the neural network function U is smooth, calculating the derivatives of the network U with respect to its input variables $\mathbf{x}$ and z via automatic differentiation [8] has no problem at all. Thus, the gradient jump of u across the interface can be computed directly from Eq. (6) as

$$[\![\nabla u]\!](\mathbf{x}_\Gamma) = \partial_z U [\![\nabla\phi_a]\!](\mathbf{x}_\Gamma) = 2\partial_z U \nabla\phi(\mathbf{x}_\Gamma). \tag{7}$$

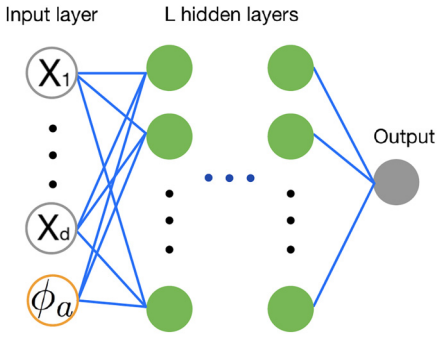

Fig. 1. Diagram of the L-hidden-layer network structure.

Notice that, in the above implementation we have used $[\![\nabla_{\mathbf{x}} U]\!](\mathbf{x}_\Gamma) = 0$ since U is smooth. By multiplying the normal vector $\mathbf{n} = \nabla\phi/\|\nabla\phi\|$ to the above equation, we obtain the following normal derivative jump of u as

$$[\![\partial_n u]\!](\mathbf{x}_\Gamma) = 2\partial_z U \|\nabla\phi(\mathbf{x}_\Gamma)\|. \tag{8}$$

Therefore, the neural network solution U is capable of capturing the cusp behavior of the solution in Eqs. (1)-(3) even if the network function $U(\mathbf{x}, z)$ is smooth across its entire $\mathbb{R}^{d+1}$ domain.

By using the relation $\nabla u = \nabla_{\mathbf{x}} U + \partial_z U \nabla\phi_a$ in $\Omega^{\pm}$, one can explicitly write the following equation after careful calculations

$$\begin{aligned}\nabla\cdot(\beta\nabla u) = \beta\left(\Delta_{\mathbf{x}} U + 2\nabla\phi_a\cdot\nabla_{\mathbf{x}}(\partial_z U) + \|\nabla\phi\|^2\partial_{zz}U + \partial_z U\Delta\phi_a\right)\\ + \quad \nabla\beta\cdot(\nabla_{\mathbf{x}} U + \partial_z U\nabla\phi_a),\end{aligned} \tag{9}$$

where $\Delta_{\mathbf{x}}$ is the Laplace operator concerning only the variable $\mathbf{x}$.

Now, Eqs. (1)-(3) can be rewritten in terms of U as follows. For succinctness, we introduce the notation $\mathcal{L}_{\beta,\phi_a} U$ to represent the right-hand side of Eq. (9) so that Eq. (1) is rewritten to the following

$$\mathcal{L}_{\beta,\phi_a} U(\mathbf{x}, \phi_a(\mathbf{x})) - \alpha(\mathbf{x}) U(\mathbf{x}, \phi_a(\mathbf{x})) = f(\mathbf{x}), \quad \mathbf{x}\in\Omega^+\cup\Omega^-. \tag{10}$$

Using the fact that $[\![\partial_n\phi_a]\!](\mathbf{x}_\Gamma) = 2\nabla\phi(\mathbf{x}_\Gamma)\cdot\mathbf{n} = 2\|\nabla\phi(\mathbf{x}_\Gamma)\|$, we can also rewrite the interface condition $[\![\beta\partial_n u]\!](\mathbf{x}_\Gamma) = \rho(\mathbf{x}_\Gamma)$ in Eq. (5) as

$$[\![\beta]\!](\mathbf{x}_\Gamma)\partial_n U + (\beta^+(\mathbf{x}_\Gamma) + \beta^-(\mathbf{x}_\Gamma))\partial_z U\,\|\nabla\phi(\mathbf{x}_\Gamma)\| = \rho(\mathbf{x}_\Gamma) \quad \mathbf{x}_\Gamma\in\Gamma, \tag{11}$$

where $\partial_n U = \nabla_{\mathbf{x}} U\cdot\mathbf{n}$. Notice that $[\![u]\!](\mathbf{x}_\Gamma) = 0$ is automatically satisfied since U is a continuous function. The associated boundary condition (3) reads

$$U(\mathbf{x}_B, \phi_a(\mathbf{x}_B)) = g(\mathbf{x}_B) \quad \mathbf{x}_B\in\partial\Omega. \tag{12}$$

The remaining task is to train the network to simultaneously satisfy Eq. (10), the jump condition (11), and the boundary condition (12) with appropriate loss function.

3.2. Physics-informed neural networks

In this subsection, we present a physics-informed neural network to approximate the solution $U(\mathbf{x}, \phi_a(\mathbf{x}))$ for Eqs. (10)-(12). The convergence of PINNs for linear elliptic PDEs was studied recently in [35]. Fig. 1 presents the structure of a L-hidden-layer feed-forward fully connected neural network where $(\mathbf{x}, \phi_a(\mathbf{x}))^T \in \mathbb{R}^{d+1}$ represents the $d+1$ feature input of the network (recall that $\phi_a(\mathbf{x})$ is the cusp-enforced level set function). We label the input layer as layer 0 and denote the feature input as $\mathbf{v}^{[0]} = (\mathbf{x}, \phi_a(\mathbf{x}))^T$. The output at the ℓ-th hidden layer with N_ℓ neurons, denoted as $\mathbf{v}^{[\ell]} \in \mathbb{R}^{N_\ell}$, presents an affine mapping of the output of layer $\ell-1$ (i.e., $\mathbf{v}^{[\ell-1]}$) followed by an action of the activation function σ in a componentwise manner as

$$\mathbf{v}^{[\ell]} = \sigma\left(W^{[\ell]}\mathbf{v}^{[\ell-1]} + \mathbf{b}^{[\ell]}\right), \quad \ell = 1, \cdots, L, \tag{13}$$

where the matrix $W^{[\ell]} \in \mathbb{R}^{N_\ell\times N_{\ell-1}}$ contains the weights connecting the structure from layer $\ell-1$ to layer ℓ, and $\mathbf{b}^{[\ell]} \in \mathbb{R}^{N_\ell}$ is the bias vector at layer ℓ. Finally, we denote the output of this multiple-hidden-layer network as

$$U_{\mathcal{N}}(\mathbf{x}, \phi_a(\mathbf{x}); \theta) = W^{[L+1]}\mathbf{v}^{[L]}, \tag{14}$$

where $W^{[L+1]} \in \mathbb{R}^{1\times N_L}$. The notation θ denotes the vector collecting all trainable parameters (including all the weights and biases) so the dimension of θ is the total number of parameters in the network that can be easily counted as $N_\theta = N_L + \Sigma_{\ell=1}^{L}(N_{\ell-1}+1)N_\ell$.

In the training process, we select M_I points in the region of $\Omega^-\cup\Omega^+$, $\{\mathbf{x}^i\}_{i=1}^{M_I}$, M_Γ points on the interface Γ, $\{\mathbf{x}_\Gamma^i\}_{i=1}^{M_\Gamma}$, and M_B points on the domain boundary $\partial\Omega$, $\{\mathbf{x}_B^i\}_{i=1}^{M_B}$, so totally $M = M_I + M_\Gamma + M_B$ training points. Under the physics-informed framework, we hereby define the loss function as the mean squared error of the residual of differential equation (10), the jump condition (11), and the boundary condition (12) as

$$\begin{aligned}\text{Loss}(\theta) = &\frac{1}{M_I}\sum_{i=1}^{M_I}\left|L_I(\mathbf{x}^i,\phi_a(\mathbf{x}^i);\theta)\right|^2 + \frac{c_\Gamma}{M_\Gamma}\sum_{i=1}^{M_\Gamma}\left|L_\Gamma(\mathbf{x}_\Gamma^i,0;\theta)\right|^2\\ &+\frac{c_B}{M_B}\sum_{i=1}^{M_B}\left|L_B(\mathbf{x}_B^i,\phi_a(\mathbf{x}_B^i);\theta)\right|^2,\end{aligned} \tag{15}$$

where the residual error L_I, interface condition error L_Γ, and boundary condition error L_B, are shown respectively as follows:

$$L_I(\mathbf{x},\phi_a(\mathbf{x});\theta) = \mathcal{L}_{\beta,\phi_a}U_{\mathcal{N}}(\mathbf{x},\phi_a(\mathbf{x});\theta) - \alpha(\mathbf{x})U_{\mathcal{N}}(\mathbf{x},\phi_a(\mathbf{x});\theta) - f(\mathbf{x}), \tag{16}$$

$$\begin{aligned}L_\Gamma(\mathbf{x}_\Gamma,0;\theta) = &[\![\beta]\!](\mathbf{x}_\Gamma)\partial_n U_{\mathcal{N}}(\mathbf{x}_\Gamma,0;\theta) + (\beta^+(\mathbf{x}_\Gamma)+\beta^-(\mathbf{x}_\Gamma))\partial_z U_{\mathcal{N}}(\mathbf{x}_\Gamma,0;\theta)\,\|\nabla\phi(\mathbf{x}_\Gamma)\|\\ &-\rho(\mathbf{x}_\Gamma),\end{aligned} \tag{17}$$

$$L_B(\mathbf{x}_B,\phi_a(\mathbf{x}_B);\theta) = U_{\mathcal{N}}(\mathbf{x}_B,\phi_a(\mathbf{x}_B);\theta) - g(\mathbf{x}_B). \tag{18}$$

The constants c_Γ and c_B appeared in the loss function (15) are chosen to balance the contribution of the terms related to the interface jump condition (11) and boundary condition (12), respectively. In latter numerical experiments, we might need to use network with smooth level set function ϕ augmentation $U_{\mathcal{N}}(\mathbf{x},\phi(\mathbf{x});\theta)$ for comparison purpose. In that case, the interface error loss in Eq. (17) should be replaced (can be easily derived) by

$$L_\Gamma(\mathbf{x}_\Gamma,0;\theta) = [\![\beta]\!](\mathbf{x}_\Gamma)\,(\partial_n U_{\mathcal{N}}(\mathbf{x}_\Gamma,0;\theta) + \partial_z U_{\mathcal{N}}(\mathbf{x}_\Gamma,0;\theta)\|\nabla\phi(\mathbf{x}_\Gamma)\|) - \rho(\mathbf{x}_\Gamma). \tag{19}$$

Meanwhile, throughout the rest of paper, we use the Levenberg-Marquardt (LM) algorithm [25] as the optimizer to train the network, and use the notation $u_{\mathcal{N}}$ to denote the network prediction solution.

Remark. The cusp-capturing PINN is designed for solving elliptic interface problems where the solution is continuous but the derivatives have jumps. The present method can be easily extended to handle problems with non-zero solution jumps. If the solution is discontinuous across the interface, we can incorporate an additional supervised learning task for solution jump approximation and the remaining part of the solution can be found by the cusp-capturing PINN. To see this, suppose we want to solve Eqs. (1)-(3) but with nonzero solution jump $[\![u]\!](\mathbf{x}_\Gamma) = \lambda(\mathbf{x}_\Gamma), \forall\mathbf{x}_\Gamma\in\Gamma$ instead. We first write the solution as $u(\mathbf{x}) = v(\mathbf{x}) + w(\mathbf{x})$ in which we assume $v(\mathbf{x})$ has the jump discontinuity $[\![v]\!](\mathbf{x}_\Gamma) = \lambda(\mathbf{x}_\Gamma)$ so $w(\mathbf{x})$ is continuous ($[\![w]\!](\mathbf{x}_\Gamma) = 0$). We further assume $v(\mathbf{x})$ has the form

$$v(\mathbf{x}) = \begin{cases} V(\mathbf{x}) & \mathbf{x}\in\Omega^-,\\ 0 & \mathbf{x}\in\Omega^+,\end{cases} \tag{20}$$

so the jump $[\![v]\!](\mathbf{x}_\Gamma) = -V(\mathbf{x}_\Gamma) = \lambda(\mathbf{x}_\Gamma)$ for $\mathbf{x}_\Gamma\in\Gamma$. The construction of $V(\mathbf{x})$ will become clear later. Substituting the expression of $u(\mathbf{x})$ into Eqs. (1)-(3), one can immediately obtain the equations for $w(\mathbf{x})$ as

$$\nabla\cdot(\beta(\mathbf{x})\nabla w(\mathbf{x})) - \alpha(\mathbf{x})w(\mathbf{x}) = \begin{cases} f(\mathbf{x}) - \nabla\cdot(\beta(\mathbf{x})\nabla V(\mathbf{x})) + \alpha(\mathbf{x})V(\mathbf{x}), & \mathbf{x}\in\Omega^-,\\ f(\mathbf{x}), & \mathbf{x}\in\Omega^+\end{cases} \tag{21}$$

$$[\![w]\!](\mathbf{x}_\Gamma) = 0, \quad [\![\beta\partial_n w]\!](\mathbf{x}_\Gamma) = \rho(\mathbf{x}_\Gamma) + \beta^-(\mathbf{x}_\Gamma)\partial_n V(\mathbf{x}_\Gamma), \quad \mathbf{x}_\Gamma\in\Gamma, \tag{22}$$

$$w(\mathbf{x}_B) = g(\mathbf{x}_B), \quad \mathbf{x}_B\in\partial\Omega. \tag{23}$$

Note that, the flux jump in Eq. (22) is obtained by the fact $[\![\beta\partial_n v]\!](\mathbf{x}_\Gamma) = -\beta^-(\mathbf{x}_\Gamma)\partial_n V(\mathbf{x}_\Gamma)$. The above equations (21)-(23) can be solved by the present cusp-capturing PINN since the solution $w(\mathbf{x})$ now is continuous.

The remaining question is how to construct the function $V(\mathbf{x})$ so that $V(\mathbf{x}_\Gamma) = -\lambda(\mathbf{x}_\Gamma)$ for $\mathbf{x}_\Gamma\in\Gamma$. Here, we simply adopt a shallow (one-hidden-layer) fully-connected feedforward neural network to approximate V by supervised learning. That is, we randomly choose M_Γ points $\{\mathbf{x}_\Gamma^i\}_{i=1}^{M_\Gamma}$ on the interface Γ, and minimize the corresponding mean squared error loss as

$$\text{Loss}(\tilde{\theta}) = \frac{1}{M_\Gamma}\sum_{i=1}^{M_\Gamma}\left(V(\mathbf{x}_\Gamma^i;\tilde{\theta}) + \lambda(\mathbf{x}_\Gamma^i)\right)^2, \tag{24}$$

where $\tilde{\theta}$ denotes the vector collecting the trainable weights and biases used in the network.

4. Numerical results

In this section, we aim to demonstrate the capability of the present neural network method for solving elliptic interface problems, Eqs. (1)-(3). We set the penalty constants in the loss function $c_B = c_\Gamma = 1$ to focus on the accuracy check of the present cusp-capturing technique. The merit of the proposed cusp-capturing PINN is to allow one to use a smooth neural network $U(\mathbf{x}, z)$ to learn the non-smooth solution, $u(\mathbf{x})$, through the relation $u(\mathbf{x}) = U(\mathbf{x}, z = \phi_a(\mathbf{x}))$. The only requirement of the choice of activation function is subject to the C^2-regularity of $u(\mathbf{x})$ in each subdomain. Thus, we simply choose the sigmoid function, $\sigma(x) = \frac{1}{1+e^{-x}}$ as our activation function. For the following numerical examples, we employ different depth networks (from 1 to L hidden layers) with equal number of neurons in each hidden layer $N_1 = N_2 = \cdots = N_L = N$. The training and test data points are generated by the Latin hypercube sampling algorithm [24], which effectively avoids the clustering of data points at some specific locations so resulting in a nearly random sampling. To measure the accuracy of the network solution, we choose M_{test} points (different from the training points) in Ω to calculate the relative L^∞ and L^2 errors defined respectively as $\|u_{\mathcal{N}} - u\|_\infty / \|u\|_\infty$ and $\|u_{\mathcal{N}} - u\|_2 / \|u\|_2$, where

$$\|u\|_\infty = \max_{1 \le i \le M_{test}} |u(\mathbf{x}^i)|, \quad \|u\|_2 = \sqrt{\frac{1}{M_{test}} \sum_{i=1}^{M_{test}} (u(\mathbf{x}^i))^2}.$$

In general, we set $M_{test} = 100M$, where M is the total number of training points. Since the predicted results will vary slightly for each experiment (it is affected by the randomness of the training and test data points, and the initialization of trainable parameters), we show the average value of the errors and losses over 5 trial runs.

In the training procedure, we use the Levenberg-Marquardt (LM) algorithm as our optimizer and update the damping parameter μ by the strategies introduced in [38]. The training is stopped when the loss value $\text{Loss}(\theta)$ is below a threshold ϵ_θ (problem dependent) or the maximum iteration (training) step $epoch = 3000$ is reached. All trials are run on a desktop equipped with one NVIDIA GeForce RTX3060 GPU. We implement the cusp-capturing PINN architecture using Pytorch (v1.13) [33] and all trainable parameters (weights and biases) are initialized using Pytorch default settings. The source codes used throughout this paper are available on GitHub at https://github.com/teshenglin/cusp_capturing_PINN.

Example 1. As the first example, we demonstrate the cusp-capturing capability for the present network by considering the following one-dimensional Poisson equation on an interval $\Omega = [0, 1]$ with an interface point at $x_\Gamma = \frac{1}{3}$:

$$\frac{d^2u}{dx^2} = 0, \quad x \in (0,1) \backslash \{x_\Gamma\}, \tag{25}$$

$$[\![u]\!](x_\Gamma) = 0, \quad [\![\frac{du}{dx}]\!](x_\Gamma) = 1, \tag{26}$$

$$u(0) = u(1) = 0. \tag{27}$$

The exact solution of the above problem can be easily derived as

$$u(x) = \begin{cases} (x_\Gamma - 1)x, & x \in [0, x_\Gamma), \\ x_\Gamma(x-1), & x \in [x_\Gamma, 1], \end{cases} \tag{28}$$

where the cusp appears exactly at the interface x_Γ. We thus choose $\phi(x) = x - x_\Gamma$ as the smooth level set function so that $\phi_a(x) = |x - x_\Gamma|$ represents the cusp-enforced level set function.

For the neural network in this test, we use a completely shallow network structure ($L = 1$) with N neurons in the hidden layer; here, the input dimension is two, one for x and the other for the augmented feature input ϕ_a. The number of overall training data points is $M = M_I + 3$, including M_I points in the interval $(0, 1)$, two points ($M_B = 2$) at the boundary, and one point ($M_\Gamma = 1$) at the interface. We use only 2 neurons in the hidden layer and 13 training points, that is, $(N, M) = (2, 13)$. After completing the training process, we use $M_{test} = 1000$ test points to examine the predicted accuracy of the network solution.

Fig. 2(a) shows the profiles of the exact solution u (denoted by the red-dashed line) and the network-predicted solution $u_{\mathcal{N}}$ with augmented input ϕ_a (solid line). One can immediately see that the ϕ_a input network solution captures the cusp sharply where the L^∞ error achieves $\|u - u_{\mathcal{N}}\|_\infty = 7.01 \times 10^{-8}$. Meanwhile, the corresponding loss drops significantly within just 40 epochs, as shown in panel (b) of the figure.

Then we test to see if the solution can be learned by using a level set function augmented input (not the cusp-enforced one); that is, we assume $u_{\mathcal{N}} = U_{\mathcal{N}}(\mathbf{x}, \phi(\mathbf{x}))$. We train the network with $(N, M) = (20, 103)$. The learned solution is shown in Fig. 2(a) (denoted by "∘") and the corresponding loss is presented in (b). It turns out that the ϕ input network learns a completely wrong solution $u_{\mathcal{N}} \approx 0$. This result is not surprising, since this network solution is inherently smooth, so all the jumps are zero, which gives $L_\Gamma(\mathbf{x}_\Gamma, 0; \theta) = -\rho(\mathbf{x}_\Gamma)$ that is independent of the trainable parameters θ. So this smooth neural network tries to minimize only the residual error and boundary error, that is, to learn a solution with zero second-order derivative and zero boundary condition. The loss for this ϕ input network shown in panel (b) is dominated by the interface loss L_Γ that gives an $O(1)$ value throughout the whole training process.

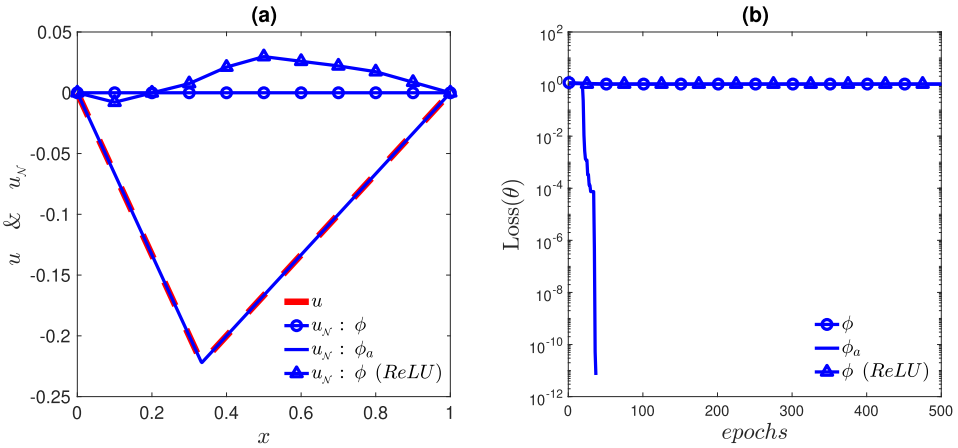

Fig. 2. (a) The profiles of the exact solution u, the network solutions $u_{\mathcal{N}}$ with augmented input ϕ and ϕ_a, and the network solution using ReLU activation function with ϕ augmented input. (b) The corresponding losses in (a). (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Meanwhile, one may wonder if a feed-forward network using the ReLU activation function with augmented smooth level set function ϕ can work due to the cusp-like profile of the ReLU function. Notice that, the ReLU function is linear so a shallow network (one hidden layer) with ReLU activation can learn the differential equation (25) with zero loss (i.e. $L_I(x, \phi(x), \theta) = 0$). However, it seems to be difficult to locate the cusp singularity for such a network which we can see from the solution profile (denote by "△") in Fig. 2(a). Again, like the sigmoid activation function with ϕ augmented input, the corresponding loss (also see in Fig. 2(b)) remains to be $O(1)$ which leads to unsuccessful training. As discussed in [40], a single network with non-differentiable activation usually does not satisfy the differential requirement in high-dimensional interface problems. As a result, the cusp singularity obtained by the network does not coincide with the given interface. This is exactly what we see from Fig. 2(a) even in a one-dimensional case.

Example 2. As the second example, we consider an elliptic equation with a piecewise-constant coefficient defined in the two-dimensional domain $\Omega = [-1, 1] \times [-1, 1]$. The embedded interface Γ is described by the zero level set of the function $\phi(x, y) = \frac{x^2}{0.5^2} + \frac{y^2}{0.5^2} - 1$, separating Ω into the inner (Ω^-) and outer (Ω^+) regions. We choose the exact solution u and the coefficient β, respectively, as

$$u(x, y) = \begin{cases} 1 - \exp\left(\frac{1}{\eta}\left(\frac{x^2}{0.5^2} + \frac{y^2}{0.5^2} - 1\right)\right), & (x, y) \in \Omega^-, \\ -\gamma \ln\left(\frac{x^2}{0.5^2} + \frac{y^2}{0.5^2}\right), & (x, y) \in \Omega^+, \end{cases} \tag{29}$$

and

$$\beta(x, y) = \begin{cases} \beta^-, & (x, y) \in \Omega^-, \\ \beta^+, & (x, y) \in \Omega^+, \end{cases}$$

where the parameter $\eta = \beta^-/\beta^+$ represents the ratio of β^- to β^+. (Here, we fix $\beta^+ = 1$ and adjust η to control the contrast of the coefficients.) One can immediately see that the solution is continuous across the interface Γ but its normal derivative has jump discontinuity as $[\![\beta \partial_n u]\!] = -4(\gamma - 1)$. The corresponding right-hand side function f can be calculated directly from Eq. (1) and the boundary condition g is given by the exact solution u on $\partial\Omega$. We introduce a number M_0 which can be regarded as the grid number used in each spatial dimension as in traditional grid-based methods so the training data set includes $M_I = M_0^2$ points in $\Omega^- \cup \Omega^+$, $M_\Gamma = 3M_0$ points on the interface Γ, and $M_B = 4M_0$ points on the boundary $\partial\Omega$, respectively. Thus, the total training points $M = M_0^2 + 3M_0 + 4M_0$.

Next, we will discuss some numerical issues about the implementation of cusp-capturing strategy, including the accuracy study of shallow neural networks with different number of neurons and training points, and the comparisons of different optimizers and different augmented inputs.

Accuracy check: shallow neural networks with different number of neurons and training points. The first experiment aims to study the number of neurons and training points needed to get satisfactory results. To test whether the proposed method works for different types of boundary condition, we impose the Dirichlet boundary condition at $x = \pm 1$ and the Neumann boundary condition at $y = \pm 1$. We choose $\alpha = 1$, $\eta = 10$, $\gamma = 2$, and fix $L = 1$ such that the neural network is completely shallow. Table 1 shows the relative L^∞ and L^2 errors between the network solution $u_{\mathcal{N}}$ and the exact solution u when using different numbers of neurons N and training points M. Also, we examine the relative L^∞ error of $\nabla u_{\mathcal{N}}$ by the

Table 1
Relative errors of u and ∇u, and training losses for the shallow network solution with different number of neurons N and training points M. Here, $\alpha = 1$, $\eta = 10$, and $\gamma = 2$ in Example 2.

(M_0, M)	(N, N_θ)	$\frac{\|u_\mathcal{N} - u\|_\infty}{\|u\|_\infty}$	$\frac{\|u_\mathcal{N} - u\|_2}{\|u\|_2}$	$\frac{\|\nabla u_\mathcal{N} - \nabla u\|_\infty}{\|\nabla u\|_\infty}$	Loss(θ)
(20, 540)	(20, 100)	1.13×10^{-3}	2.82×10^{-3}	2.48×10^{-3}	2.23×10^{-4}
	(30, 150)	4.41×10^{-5}	2.33×10^{-4}	2.08×10^{-4}	1.61×10^{-7}
	(40, 200)	1.14×10^{-5}	4.56×10^{-5}	4.10×10^{-5}	2.29×10^{-9}
	(50, 250)	5.17×10^{-6}	3.69×10^{-5}	3.09×10^{-5}	1.04×10^{-10}
(30, 1110)	(20, 100)	7.75×10^{-4}	4.47×10^{-4}	1.17×10^{-3}	7.12×10^{-5}
	(30, 150)	2.77×10^{-5}	2.21×10^{-5}	7.73×10^{-5}	8.79×10^{-8}
	(40, 200)	4.40×10^{-6}	4.30×10^{-6}	1.75×10^{-5}	2.97×10^{-9}
	(50, 250)	1.13×10^{-6}	1.07×10^{-6}	4.33×10^{-6}	1.17×10^{-10}

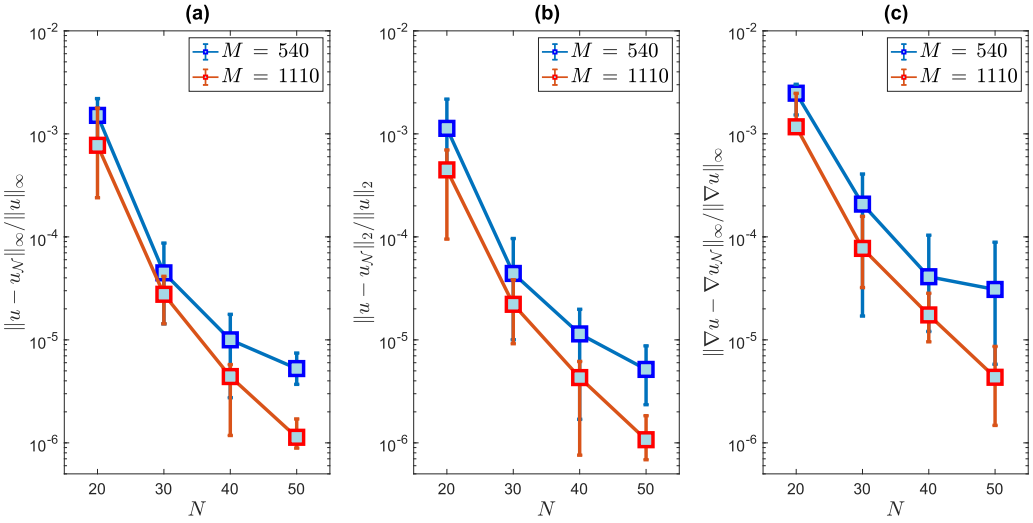

Fig. 3. Error bar plots associated with Table 1. Each bar represents the errors over 5 trial runs. (a) Relative L^∞ error of $u_\mathcal{N}$; (b) Relative L^2 error of $u_\mathcal{N}$; (c) Relative L^∞ error of $\nabla u_\mathcal{N}$.

formula $\|\nabla u_\mathcal{N} - \nabla u\|_\infty / \|\nabla u\|_\infty$ with the definition $\|\nabla u\|_\infty = \frac{1}{2}\left(\|\frac{\partial u}{\partial x}\|_\infty + \|\frac{\partial u}{\partial y}\|_\infty\right)$. Notice that since the network has only one hidden layer, the overall number of trainable parameters is $N_\theta = N(d+3) = 5N$ for this two-dimensional problem. The corresponding final loss values are also shown in the table. One can see that the present model can achieve a prediction accuracy of about 0.1% in relative L^∞ and L^2 errors even using one hidden layer with merely $N = 20$ neurons. As we increase the number of neurons from $N = 20$ to $N = 50$, the relative error decreases from the magnitude $O(10^{-3})$ to $O(10^{-6})$, and the loss drops from $O(10^{-4})$ to $O(10^{-10})$ accordingly. In addition, one can also see that all relative errors decrease by increasing the number $M_0 = 20$ to $M_0 = 30$ (same as increasing the number of total training points M). From this numerical experiment, we conclude that the solution errors can indeed be reduced by increasing the number of neurons or training points, which provides an informal evidence for the numerical convergence of the present method. The errors for the solution gradient show a similar convergence trend as the solution errors. In addition, since the derivatives are computed by automatic differentiation, the relative L^∞ errors of the gradient seem to have almost the same order of magnitude as the ones of the solution itself. We also present the error bar plots of 5 trail runs associated with Table 1 in Fig. 3.

We depict the solution profile $u_\mathcal{N}$ in Fig. 4(a), the corresponding absolute error $|u_\mathcal{N} - u|$ in Fig. 4(b), and the cross-sectional view of $u_\mathcal{N}$ and u along the line $y = 0$ in Fig. 4(c). One can clearly see that the cusps on the interface are accurately captured and the largest error occurs at the domain boundary rather than on the interface, which indicates the effectiveness of the present network model.

Comparison of different optimizers. The reasons why we choose Levenberg-Marquardt algorithm as our optimizer are two-fold. First, the LM algorithm is a combination of Gauss-Newton and gradient descent method which is suitable for nonlinear least squares problems. (The minimization of the loss function in the present paper is a nonlinear least square problem.) Meanwhile, the number of parameters to be trained in our proposed neural network is moderate (a few hundreds), so the cost per epoch for LM algorithm is acceptable. Second, the LM algorithm usually converges faster than commonly used optimizers such as Adam [15] and L-BFGS [17]. Here, we compare the training performance for three different optimizers (Adam, L-BFGS, LM) by showing the corresponding training loss evolutions in Fig. 5. We use the previous setup and fix the number of training points $M = 1110$ but vary the number of neurons from $N = 30$ to 50. One can see that, the LM optimizer

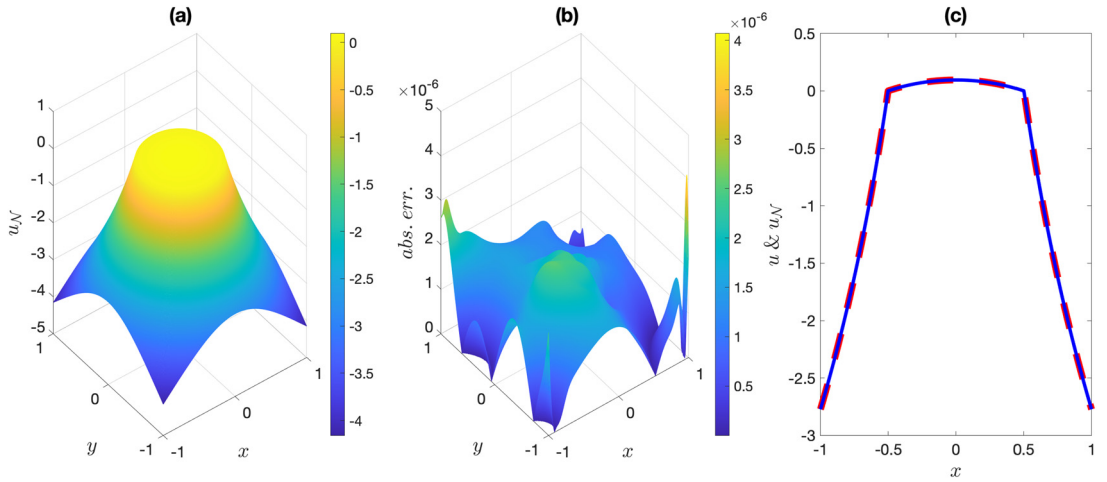

Fig. 4. (a) The solution profile of $u_{\mathcal{N}}$; (b) Absolute error $|u_{\mathcal{N}} - u|$; (c) Cross-sectional view of $u_{\mathcal{N}}$ (blue-solid line) and u (red-dashed line) along the line $y = 0$. The figure is the case when $(M_0, M) = (30, 1110)$ and $(N, N_\theta) = (50, 250)$ in Table 1.

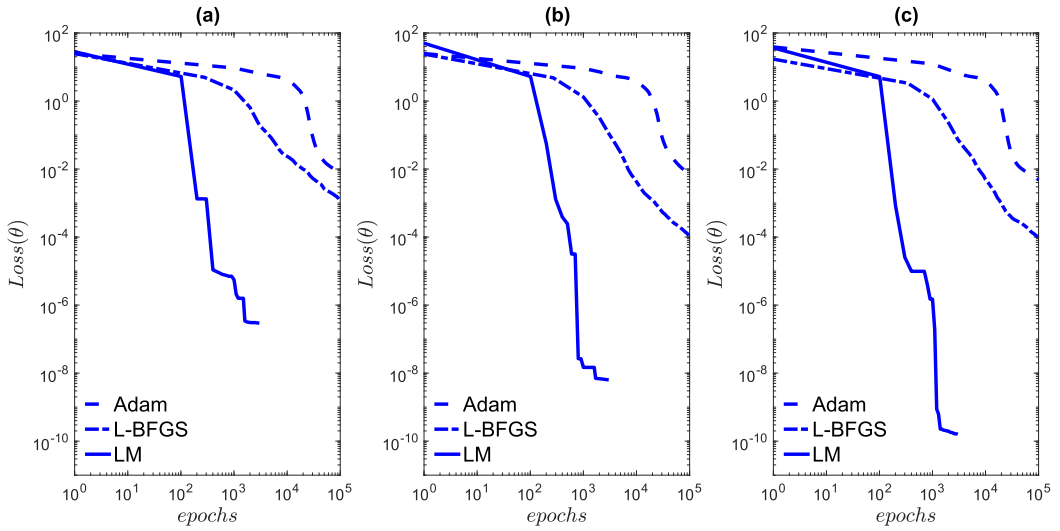

Fig. 5. Comparison of loss evolutions using different optimizers: Adam (dashed line), L-BFGS (dashed-dotted line), and LM (solid line). (a) $N = 30$; (b) $N = 40$; (c) $N = 50$. All cases use 1110 training data points.

can effectively reduce the loss to $O(10^{-10})$ within 3000 epochs when the number of neurons increases. In contrast, the Adam and L-BFGS optimizers reduce the loss values more slowly, and barely achieve the losses of the magnitude $O(10^{-2})$ and $O(10^{-4})$ even up to 10^5 epochs. Although not shown here, the final relative errors of LM algorithm show about three orders of magnitude smaller than the ones obtained by the Adam or L-BFGS.

Comparison of different augmented inputs. In the third experiment, we demonstrate the robustness of present cusp-enforced level set function augmented input $\phi_a = |\phi|$. Here, we keep $\eta = 10$ but choose $\alpha = 0$ and impose Dirichlet boundary condition on $\partial\Omega$ for simplicity. We also set $\gamma = 1$ so the flux jump $[\![\beta\partial_n u]\!]$ is zero while the solution u still has discontinuous first derivatives to focus on the expressibility of the present network. We compare the relative errors and the losses of using either ϕ or ϕ_a as the augmented input in a fixed shallow neural network with the number of neurons $N = 40$. The total number of training points used is $M = 1110$ (or $M_0 = 30$). The results are shown in Table 2 where the used augmented input is listed in the first column. One can see that the prediction accuracy for the level set function input ϕ is quite poor. The relative errors for ϕ and ϕ_a input are $O(10^{-1})$ and $O(10^{-5})$, respectively, so the latter significantly outperforms the former. Therefore, the present cusp-enforced augmented feature input is indeed more accurate and capable of tackling the interface problem with discontinuous first derivatives.

We also show the evolutionary plots of training loss for the two cases in Fig. 6(a). After a few hundreds of epochs, the training loss for the case with augmented input ϕ becomes sluggish while the one with ϕ_a input continues to go down afterwards and reaches to the order of 10^{-8} eventually.

Table 2
Relative errors and training loss for the shallow network with different augmented inputs, ϕ and ϕ_a. Here, $\alpha = 0$, $\eta = 10$, and $\gamma = 1$ in Example 2. $(M_0, M) = (30, 1110)$, $(L, N, N_\theta) = (1, 40, 200)$.

Augmented input	$\|u_{\mathcal{N}} - u\|_\infty / \|u\|_\infty$	$\|u_{\mathcal{N}} - u\|_2 / \|u\|_2$	Loss(θ)
ϕ	8.01×10^{-1}	9.13×10^{-1}	5.57×10^{-2}
ϕ_a	2.98×10^{-5}	3.32×10^{-5}	9.17×10^{-9}

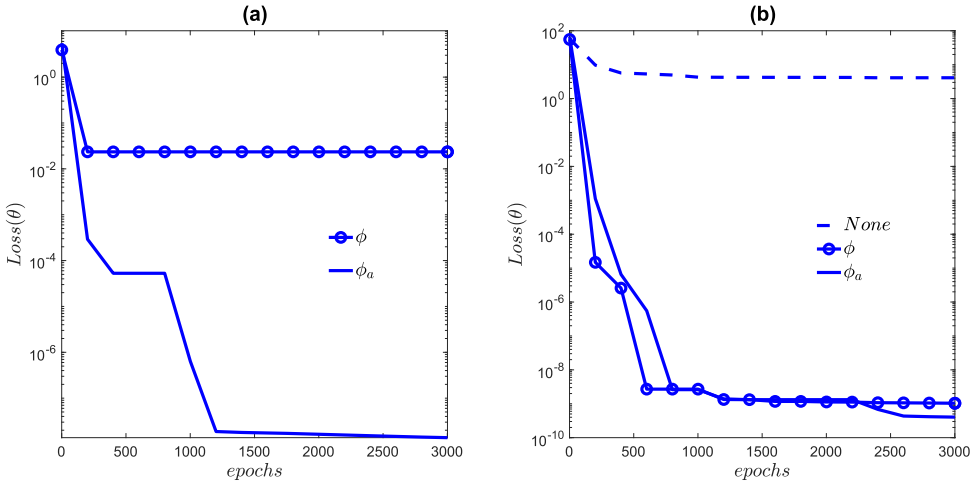

Fig. 6. (a) The evolutions of training Loss(θ) corresponding to Table 2. (b) The evolutions of training Loss(θ) corresponding to Table 3.

Table 3
Relative errors, training losses, and total training time for the shallow network with an augmented input (ϕ or ϕ_a) or without augmented input. Here, $\alpha = 0$, $\eta = 1$, and $\gamma = 1$ in Example 2. $(M_0, M) = (30, 1110)$.

Augmented input	$\|u_{\mathcal{N}} - u\|_\infty / \|u\|_\infty$	$\|u_{\mathcal{N}} - u\|_2 / \|u\|_2$	Loss(θ)	Elapsed time
None (PINN)	1.59×10^{-1}	9.15×10^{-2}	4.34×10^{0}	10.9(s)
ϕ	8.55×10^{-3}	5.43×10^{-3}	9.07×10^{-9}	17.8(s)
ϕ_a	1.65×10^{-5}	1.91×10^{-5}	4.61×10^{-10}	22.1(s)

To further investigate the power of function expressibility on the proposed cusp-enforced level set function augmentation, we consider a special case with $\eta = 1$ ($\beta^- = \beta^+ = 1$) and $\gamma = 1$ so that the jumps $[\![\beta]\!] = 0$ and $[\![\beta \partial_n u]\!] = 0$ simultaneously. One can immediately see from Eq. (5) that the normal derivative jump of u equals to zero too, i.e., $[\![\partial_n u]\!] = 0$. In this case, the solution u is continuously differentiable across the interface Γ so one might wonder if the level set function augmentation makes any differences. Table 3 shows the results for a shallow network with ϕ, ϕ_a and without augmented input (denoted by "None"). For the one without augmented variable, the input is solely the position $\mathbf{x}$. To have the same number of parameters used in the network, the one without augmented input uses $N = 50$ neurons while the ones with augmented level set function input use $N = 40$ neurons. Despite the fact that the solution is C^1, the network with solely $\mathbf{x}$ input cannot train the solution properly as the training loss remains $O(1)$ (see Fig. 6(b)) so the relative errors are greater than 5%. Again, the errors with ϕ_a augmented input are smaller than the ones with ϕ input in two orders of magnitude; that is, $O(10^{-5})$ versus $O(10^{-3})$. One can perceive that the network with cusp-enforced level set function augmentation still can predict the solution more accurately even though it is designed to capture the first-order derivatives correctly while the second-order derivatives are discontinuous across the interface in this example.

We also show the overall training time in the last column of Table 3. Under the same setting, the training time per epoch using cusp-capturing PINN is indeed more costly than the one using the PINN (without any augmented input). However, as discussed earlier, if we use merely PINN, we are unable to train the network successfully even though the solution has the zero flux jump.

Example 3. The third example illustrates that the present method is applicable for solving interface problems with high-contrast coefficients defined on irregular domains. We consider a five-fold flower region $\Omega = \{(x(r,\theta),\ y(r,\theta)) \in \mathbb{R}^2 \,|\, r(\theta) \leq 1 - 0.2\cos(5\theta)\}$ with an embedded interface, $\Gamma = \{(x,\ y) \in \mathbb{R}^2 \,|\, x^2 + y^2 = \frac{1}{4}\}$. As in Example 2, the coefficient β is defined in a piecewise-constant manner. The exact solution u is defined as

Table 4
Relative errors and training losses in Example 3.

$\eta = \beta^-/\beta^+$	(L, N, N_θ)	$\|u_\mathcal{N} - u\|_\infty / \|u\|_\infty$	$\|u_\mathcal{N} - u\|_2 / \|u\|_2$	Loss(θ)
10^4	$(1, 63, 315)$	3.29×10^{-5}	3.29×10^{-5}	1.35×10^{-9}
	$(2, 15, 315)$	3.65×10^{-5}	3.82×10^{-5}	7.29×10^{-10}
	$(3, 11, 319)$	8.73×10^{-5}	1.24×10^{-4}	3.32×10^{-9}
10^{-4}	$(1, 190, 950)$	4.25×10^{-3}	1.42×10^{-3}	6.25×10^{-9}
	$(2, 28, 952)$	3.88×10^{-4}	1.28×10^{-4}	1.82×10^{-11}
	$(3, 20, 940)$	1.50×10^{-3}	5.07×10^{-4}	3.05×10^{-10}

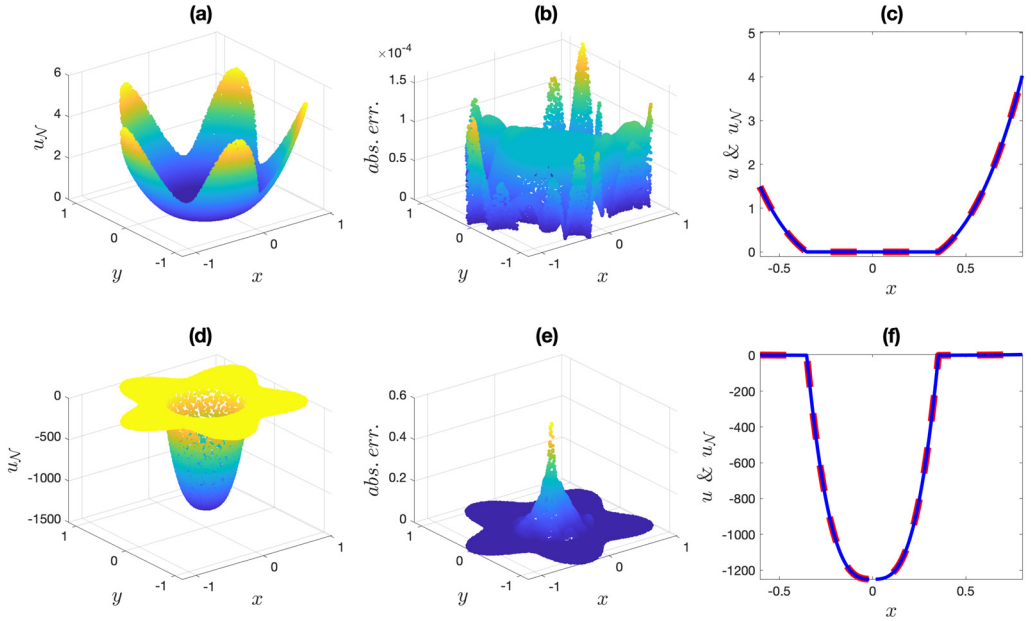

Fig. 7. (a) and (d): The profile of $u_\mathcal{N}$; (b) and (e): Absolute point-wise error $|u_\mathcal{N} - u|$; (c) and (f): Cross-sectional view of $u_\mathcal{N}$ (blue solid line) and u (red dashed line) along the line $y = x$. The upper panel is for $\eta = 10^4$ with $(L, N, N_\theta) = (2, 15, 315)$, and the lower panel is for $\eta = 10^{-4}$ with $(L, N, N_\theta) = (2, 28, 952)$ in Example 3.

$$u(x, y) = \begin{cases} \frac{1}{\beta^-}\left(\left(x^2 + y^2\right)^{\frac{3}{2}} - \frac{1}{8}\right), & (x, y) \in \Omega^-, \\ \frac{3}{\beta^+}\left(\left(x^2 + y^2\right)^{\frac{3}{2}} - \frac{1}{8}\right), & (x, y) \in \Omega^+, \end{cases} \tag{30}$$

and the Dirichlet boundary condition is imposed for simplicity. This problem was similarly studied by Wang et al. [39] using deep Ritz method on a square domain with an embedded circular interface. Again, the contrast ratio is defined by $\eta = \beta^-/\beta^+$, and we fix $\beta^+ = 1$ so $\eta = \beta^-$. Here, we consider two high-contrast ratios; namely $\eta = 10^{-4}$ and 10^4. The cusp-enforced level set function is chosen as $\phi_a(x, y) = \left|4(x^2 + y^2) - 1\right|$. We generate $M = 1498$ training data ($M_I = 1138$, $M_B = 240$, and $M_\Gamma = 120$) for the case of $\eta = 10^4$, and employ the networks comprising from single to three hidden layers. The number of neurons for each network is chosen such that the number of trainable parameters N_θ is almost the same. As shown in Table 4, for the contrast ratio $\eta = 10^4$, all network solutions can achieve accurate prediction with relative L^2 errors ranging from $O(10^{-4})$ to $O(10^{-5})$, which outperform the results obtained in [39]. However, for the contrast ratio $\eta = 10^{-4}$, the magnitude of exact solution u in Ω^- is of the order $O(10^3)$ which is much larger than the solution in Ω^+ of $O(1)$ (see also in Fig. 7(d)). So we have to use more neurons and training points ($M = 2959$ with $M_I = 2519$, $M_B = 240$, and $M_\Gamma = 200$) to train the networks. In this case, the relative errors range from $O(10^{-3})$ to $O(10^{-4})$. In addition, we depict the network solution profile, absolute point-wise error, and the cross-sectional view along $y = x$ in Fig. 7. The upper and lower panels are for the contrast ratio $\eta = 10^4$ and 10^{-4}, respectively. One can see that, without paying extra numerical efforts, the present model is able to tackle the interface problems in irregular domains thanks to the mesh-free advantage of neural network approximation. On the other hand, it could be quite tedious in implementation for traditional grid-based methods to handle such problems.

Example 4. In the fourth examples, we deal with the three-dimensional discontinuous variable-coefficient case and compare the accuracy of the present network solution with one of the immersed interface method (IIM) in [5]. The domain is set as

Table 5
Relative errors and training losses in Example 4. The results produced by IIM use $104 \times 104 \times 104$ grid points.

b	(L, N, N_θ)	$\|u_{\mathcal{N}} - u\|_\infty / \|u\|_\infty$	$\|u_{\mathcal{N}} - u\|_2 / \|u\|_2$	Loss(θ)
1	(1, 40, 240)	1.90×10^{-6}	2.17×10^{-6}	7.13×10^{-11}
	(2, 12, 228)	1.15×10^{-6}	1.77×10^{-6}	7.53×10^{-11}
	(3, 9, 234)	1.49×10^{-6}	1.52×10^{-6}	5.86×10^{-11}
	IIM	9.59×10^{-5}		
10	(1, 40, 240)	2.48×10^{-6}	1.92×10^{-6}	3.68×10^{-11}
	(2, 12, 228)	3.82×10^{-6}	1.84×10^{-6}	5.59×10^{-11}
	(3, 9, 234)	3.95×10^{-6}	3.11×10^{-6}	3.46×10^{-11}
	IIM	1.01×10^{-4}		
1000	(1, 40, 240)	5.04×10^{-6}	3.51×10^{-7}	7.83×10^{-11}
	(2, 12, 228)	5.89×10^{-6}	6.38×10^{-7}	1.43×10^{-10}
	(3, 9, 234)	4.40×10^{-6}	5.95×10^{-7}	2.21×10^{-10}
	IIM	1.61×10^{-4}		

the cube $\Omega = [-1, 1] \times [-1, 1] \times [-1, 1]$ in which the embedded interface is given by $\Gamma = \{(x, y, z) \in \mathbb{R}^3 \,|\, x^2 + y^2 + z^2 = r_0^2\}$. The exact solution u and the variable-coefficient β are chosen the same as in [5],

$$u(x, y, z) = \begin{cases} r^2, & r < r_0, \\ r_0^2 + \frac{1}{b}\left(\frac{r^4}{2} + r^2 - \frac{r_0^4}{2} - r_0^2\right), & r \geq r_0, \end{cases}$$

and

$$\beta(x, y, z) = \begin{cases} r^2 + 1, & r < r_0, \\ b, & r \geq r_0, \end{cases}$$

where $r_0 = 1/2$, $r = \sqrt{x^2 + y^2 + z^2}$, and the source term $f(x, y, z) = 10r^2 + 6$. The solution satisfies the homogeneous jump conditions $[\![u]\!] = 0$ and $[\![\beta \partial_n u]\!] = 0$. However, the variable coefficient β controlled by the parameter b implies the discontinuity of the normal derivative $[\![\partial_n u]\!]$ at the interface Γ. The cusp-enforced level set function is chosen as $\phi_a(x, y, z) = \left|4(x^2 + y^2 + z^2) - 1\right|$.

For the sampling of training data points, we generate M_I data points in the region $\Omega^+ \cup \Omega^-$, and M_B on the domain boundary ($M_B/6$ uniformly distributed training points on each face), while M_Γ data points on the surface Γ are generated by DistMesh [30]. In each of the following tests, the number of overall training points used is $M = 3360$ ($M_I = 800$, $M_B = 2400$, and $M_\Gamma = 160$).

Table 5 shows the relative errors and losses of the present method for three cases of $b = 1$, 10 and 1000. Surprisingly, no matter how large the parameter b is, the present method with single- or multiple-hidden-layer structure gives accurate network predictions with the relative L^∞ and L^2 errors of the magnitude $O(10^{-6})$. Here, we also present the results produced by IIM [5] using $104 \times 104 \times 104$ uniformly distributed grid points. It should be noted that, in 3D IIM, the total number of degree of freedom (unknowns) is 104^3 while the number of trainable parameters is just about 240 for the present method. One can clearly see that our results outperform the ones obtained by IIM in almost two orders of magnitude in the relative L^∞ error.

Example 5. In this example, we consider a problem of dimension $d = 6$ to show that the present method is able to solve high-dimensional problems. Same problem was also solved in [18] using a shallow Ritz method. Here we consider the domain Ω as a 6-sphere of radius 0.6 enclosing a smaller 6-sphere of radius 0.5 as Ω^-. The cusp-enforced level set function is chosen as $\phi_a(\mathbf{x}) = \left|(\|\mathbf{x}\|_2/0.5)^2 - 1\right|$, where $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5, x_6)$. We fix $\alpha = 0$, a constant coefficient $\beta(\mathbf{x}) = 1$, and the exact solution is defined as

$$u(\mathbf{x}) = \begin{cases} \exp(0.5^2 - \|\mathbf{x}\|_2^2) + \sum_{i=1}^5 \sin(x_i) & \mathbf{x} \in \Omega^+, \\ 1 + 2\sin(0.5^2 - \|\mathbf{x}\|_2^2) + \sum_{i=1}^5 \sin(x_i) & \mathbf{x} \in \Omega^-. \end{cases} \tag{31}$$

The right-hand side functions can be obtained using Eqs. (1)-(3).

We use a shallow network ($L = 1$) structure with $M = 2628$ points to train the network. The results are shown in Table 6. Using 40 neurons in the hidden layer (and correspondingly 360 trainable parameters), the relative L^∞ and L^2 errors are in the order of $O(10^{-6})$ and $O(10^{-7})$, respectively. This example shows that the present method is applicable to solve high-dimensional elliptic interface problems.

Example 6. Next, we take an example in [1] that has its solution being discontinuous and make an accuracy comparison with the recent mesh-free methods [1,26]. We consider a two-dimensional computational domain $\Omega = [-1, 1] \times [-1, 1]$ with

Table 6
Relative errors and losses with $M = 2628$ training data points where $(M_I, M_B, M_\Gamma) = (500, 1064, 1064)$ in Example 5.

(N, N_θ)	$\|u_\mathcal{N} - u\|_\infty / \|u\|_\infty$	$\|u_\mathcal{N} - u\|_2 / \|u\|_2$	Loss(θ)
(10, 90)	2.16×10^{-3}	1.37×10^{-3}	1.08×10^{-4}
(20, 180)	7.69×10^{-4}	2.45×10^{-4}	1.48×10^{-6}
(30, 270)	9.54×10^{-5}	3.79×10^{-5}	1.51×10^{-8}
(40, 360)	1.86×10^{-6}	6.90×10^{-7}	5.77×10^{-11}

Table 7
Comparison of L^∞ errors using the present method and two recent mesh-free methods [1,26] in Example 6. The number in the parentheses represents the number of m with the highest degree of polynomial $m-1$ used in [26].

(L, N, N_θ, M)	Present	No. nodes	Ahmad et al. [1]	No. nodes	Oruç [26]
(1, 155, 775, 3150)	4.31×10^{-3}	1600	8.75×10^{-3}	1365 (23)	1.54×10^{-3}
(2, 25, 775, 3150)	3.78×10^{-4}	6400	1.52×10^{-3}	2490 (25)	1.04×10^{-4}
(3, 20, 940, 4535)	3.72×10^{-5}	25600	-	4065 (27)	4.08×10^{-6}

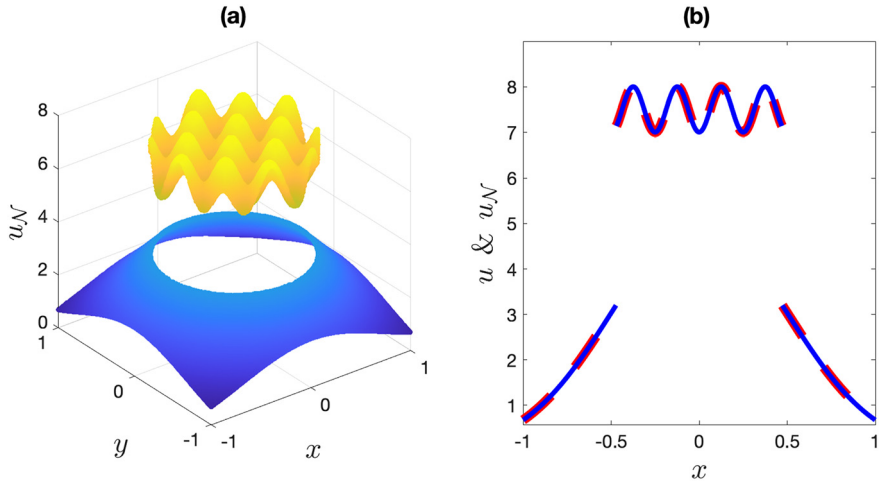

Fig. 8. The corresponding network solution plots in Example 6. $(L, N, N_\theta) = (3, 20, 940)$, $(M_I, M_B, M_\Gamma) = (3635, 600, 300)$. (a) The profile of the network solution $u_\mathcal{N}$; (b) Cross-sectional view of $u_\mathcal{N}$ (blue solid line) and u (red dashed line) along the line $y = x$.

an embedded circular interface Γ represented by the zero level set function $\phi(x, y) = x^2 + y^2 - \left(\frac{2}{3}\right)^2$. The exact solution is expressed as

$$u(x, y) = \begin{cases} \sin(4\pi x)\sin(4\pi y) + 7, & (x, y) \in \Omega^-, \\ 5\exp(-x^2 - y^2), & (x, y) \in \Omega^+, \end{cases}$$

and the coefficient β is a piecewise constant with $\beta^- = 2$ and $\beta^+ = 3$ in Ω^- and Ω^+, respectively. This example also aims to demonstrate the applicability of the proposed method presented in the Remark since the above analytic solution is discontinuous across the interface. Following the procedures in the Remark, we use a shallow network with 100 neurons and 1000 random points $\mathbf{x}_\Gamma$ on the interface to train the function $V(\mathbf{x})$ satisfying $V(\mathbf{x}_\Gamma) = -[\![u]\!](\mathbf{x}_\Gamma)$. Once V is available (thus v is obtained), we apply the present cusp-capturing PINN to solve Eqs (21)-(23) to obtain the solution w. Then we can recover the solution $u = v + w$. Table 7 presents the L^∞ errors of the proposed method and two other non-neural network mesh-free methods, including the local mesh-free method based on LMM2P in [1] and the Pascal polynomials-based multiple-scale approach in [26].

The present networks with the number of hidden layer $L = 1, 2$ use exactly same number of trainable parameters $N_\theta = 775$ and same number of total training points $M = 3150$ ($M_I = 2550$, $M_B = 400$, and $M_\Gamma = 200$) which give the L^∞ errors ranging from the magnitude $O(10^{-3})$ to $O(10^{-4})$. Here, using a deeper network seems to predict more accurate results than the shallow one under the same number of trainable parameters used. Therefore, we use a deep network $(L, N, N_\theta) = (3, 20, 940)$ with $M = 4535$ ($M_I = 3635$, $M_B = 600$, and $M_\Gamma = 300$) training points to reduce the L^∞ error to the magnitude of $O(10^{-5})$, where the solution profile $u_\mathcal{N}$ and its cross-sectional view along the line $y = x$ are shown in Fig. 8. In the Table, the number of nodes indicates the number of mesh-free points used in these methods which works like the number of training points M used in the present method. One can immediately see that our numerical results are slightly more accurate than the ones in [1] and less accurate than the ones obtained in [26].

Table 8
Relative errors and training losses in Example 7.

(N, N_θ)	$\|u_{\mathcal{N}} - u\|_\infty / \|u\|_\infty$	$\|u_{\mathcal{N}} - u\|_2 / \|u\|_2$	Loss(θ)
(25, 150)	7.55×10^{-4}	5.03×10^{-4}	1.16×10^{-7}
(50, 300)	3.04×10^{-5}	1.09×10^{-5}	1.54×10^{-9}
(100, 600)	3.07×10^{-6}	1.05×10^{-6}	1.36×10^{-11}

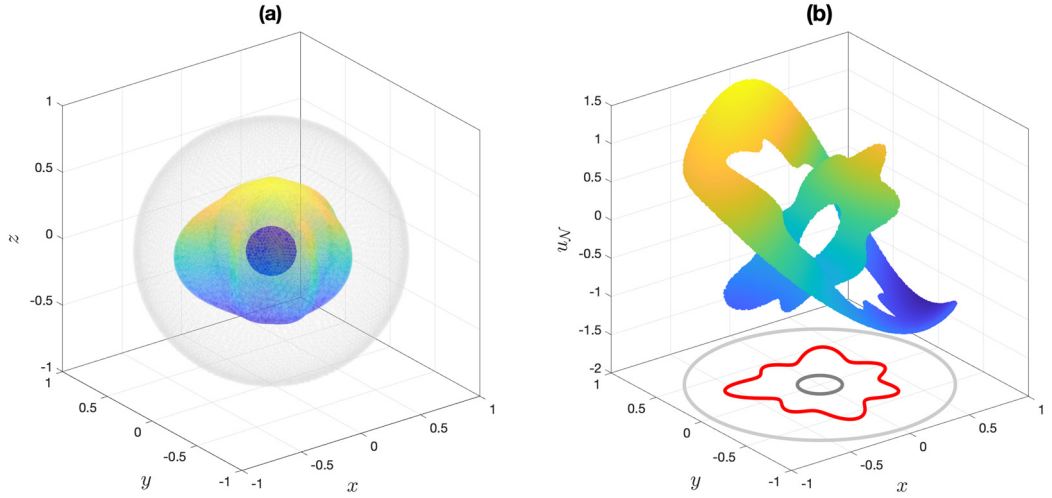

Fig. 9. (a) The illustration of domain and interface geometries in Example 7. (b) The cross-sectional view of the network solution $u_{\mathcal{N}}$ on $z = 0$. The red and grey curves indicate the corresponding cross-sectional interface and domain boundaries, respectively.

Example 7. The last example is taken from [3], in which we consider a spherical shell $\Omega = \{(x, y, z) \in \mathbb{R}^3 \mid 0.151^2 \le x^2 + y^2 + z^2 \le 0.911^2\}$ where a complex embedded interface Γ is represented by the zero level set of the level set function

$$\phi(x, y, z) = \sqrt{x^2 + y^2 + z^2} - r_0 \left(1 + \left(\frac{x^2 + y^2}{x^2 + y^2 + z^2}\right)^2 \sum_{k=1}^{3} a_k \cos\left(n_k \left(\tan^{-1}\left(\frac{y}{x}\right) - \theta_k\right)\right)\right),$$

and the setup of parameters is shown as follows:

$$r_0 = 0.483, \quad \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 0.1 \\ -0.1 \\ 0.15 \end{pmatrix}, \quad \begin{pmatrix} n_1 \\ n_2 \\ n_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 4 \\ 7 \end{pmatrix}, \text{ and } \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{pmatrix} = \begin{pmatrix} 0.5 \\ 1.8 \\ 0 \end{pmatrix}.$$

The illustration of domain and interface geometry can be found in Fig. 9(a). Note that, the dark-shading region enclosed in the interface is the inner boundary of the domain Ω. We choose the following same solution u and the coefficient β as in [3]

$$u(x, y, z) = \begin{cases} \sin(2x)\cos(2y)e^z, & (x, y, z) \in \Omega^-, \\ \left(16\left(\frac{(y-x)}{3}\right)^5 - 20\left(\frac{(y-x)}{3}\right)^3 + 5\left(\frac{(y-x)}{3}\right)\right)\log(x + y + 3)\cos(z), & (x, y, z) \in \Omega^+, \end{cases}$$

$$\beta(x, y, z) = \begin{cases} 10\left(1 + \frac{1}{5}\cos(2\pi(x + y))\sin(2\pi(x - y))\cos(z)\right), & (x, y, z) \in \Omega^-, \\ 1, & (x, y, z) \in \Omega^+. \end{cases}$$

The right-hand side functions can be obtained using Eqs. (1)-(3).

Again, the above analytic solution is obviously discontinuous across the interface so we have to follow the solution procedures discussed in the Remark to obtain the approximate network solution. First, we use a shallow network with 100 neurons and 752 training points (generated by DistMesh [30]) $\mathbf{x}_\Gamma$ on the interface to train the function $V(\mathbf{x})$ satisfying $V(\mathbf{x}_\Gamma) = -[\![u]\!](\mathbf{x}_\Gamma)$. Once V is available (thus v is obtained), we solve Eqs (21)-(23) by applying the present cusp-capturing PINN with one-hidden-layer and 2460 training data points ($M_I = 801$, $M_B = 907$, and $M_\Gamma = 752$) to train the solution w. After that, we obtain the network approximate solution $u = v + w$. Table 8 shows the relative L^∞ and L^2 errors for different number of neurons N used in the hidden layer. One can see that, using merely 25 neurons in the hidden layer (correspondingly 150 trainable parameters), the relative errors and training losses are of the magnitudes $O(10^{-4})$ and $O(10^{-7})$, respectively. The relative errors can be reduced to the magnitude $O(10^{-6})$ when the number of neurons increases to 100. Fig. 9(b) shows the cross-sectional profile of the network solution on the hyperplane $z = 0$. As a result, the present

method is indeed applicable for solving elliptic interface problems in irregular domain with complex interface subject to nonzero solution jump condition.

5. Conclusion

We propose a cusp-capturing physics-informed neural network for solving the discontinuous-coefficient elliptic interface problems. By introducing a cusp-enforced level set function as an additional feature input to the network, the predicted solution by the network can retain the inherent properties of the solution which is continuous but the normal derivative has a jump discontinuity on the interface. The training procedure uses the LM-based optimizer to minimize the loss function comprising mean squared errors of the equation residual, the interface condition, and the boundary condition in the same spirit as the physics-informed neural networks. We conduct a series of numerical tests to show the accuracy of the present network, with particular emphasis on the number of neurons and training points, and the effectiveness of the cusp-capturing technique. A high-contrast coefficient interface problem is included in our numerical experiments, and the accuracy outperforms the one obtained in previous work. The present network is efficient in terms of network structure since one hidden layer with a moderate number of neurons and sufficiently enough training data points can achieve quite accurate predictions. The results are also comparable to traditional grid-based methods, such as the immersed interface method. Besides, if the solution is discontinuous across the interface, we can simply incorporate an additional supervised learning task for solution jump approximation into the present network without much difficulty. In the future, we shall apply the present network method to practical applications where traditional grid-based methods are difficult to implement and extend to the time-dependent discontinuous-coefficient interface problems. Meanwhile, using functions other than level sets to represent interfaces for handling the C^0-interfaces and considering multiple interfaces is beyond the scope of this paper and is certainly worthy exploring in the future.

CRediT authorship contribution statement

Yu-Hau Tseng: Numerical experiments, Visualization and Writing. **Te-Sheng Lin:** Conceptualization, Methodology and Writing. **Wei-Fan Hu:** Methodology and Writing. **Ming-Chih Lai:** Conceptualization, Methodology and Writing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgement

Y.-H. Tseng, T.-S. Lin, W.-F. Hu, and M.-C. Lai acknowledge the supports by National Science and Technology Council (Taiwan), under research grants 111-2115-M-390-002, 111-2628-M-A49-008-MY4, 111-2115-M-008-009-MY3, and 110-2115-M-A49-011-MY3, respectively. T.-S. Lin and W.-F. Hu also acknowledge the supports by National Center for Theoretical Sciences, Taiwan.

References

[1] M. Ahmad, Siraj-ul-Islam, E. Larsson, Local meshless methods for second order elliptic interface problems with sharp corners, J. Comput. Phys. 416 (2020) 109500.
[2] J.W. Barrett, C.M. Elliot, Fitted and unfitted finite-element methods for elliptic equations with smooth interfaces, IMA J. Numer. Anal. 7 (3) (1987) 283–300.
[3] D. Bochkov, F. Gibou, Solving elliptic interface problems with jump conditions on Cartesian grids, J. Comput. Phys. 407 (2020) 109269.
[4] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Syst. 2 (1989) 303–314.
[5] S. Deng, K. Ito, Z. Li, Three-dimensional elliptic solvers for interface problems and applications, J. Comput. Phys. 184 (2003) 215–243.
[6] R. Egan, F. Gibou, xGFM: recovering convergence of fluxes in the ghost fluid method, J. Comput. Phys. 409 (2020) 109351.
[7] W. E, B. Yu, The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. 6 (2018) 1–12.
[8] A. Griewank, A. Walther, Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, 2nd ed., SIAM, 2008.
[9] H. Guo, X. Yang, Deep unfitted Nitsche method for elliptic interface problems, Commun. Comput. Phys. 31 (2022) 1162–1179.
[10] C. He, X. Hu, L. Mu, A mesh-free method using piecewise deep neural network for elliptic interface problems, J. Comput. Appl. Math. 412 (2022) 114358.
[11] R.J. Hill, D.A. Saville, W.B. Russel, Electrophoresis of spherical polymer-coated colloidal particles, J. Colloid Interface Sci. 258 (2002) 56–74.
[12] K. Hornik, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (1989) 359–366.
[13] W.-F. Hu, M.-C. Lai, Y.-N. Young, A hybrid immersed boundary and immersed interface method for electrohydrodynamic simulations, J. Comput. Phys. 282 (2015) 47–61.
[14] W.-F. Hu, T.-S. Lin, M.-C. Lai, A discontinuity capturing shallow neural network for elliptic interface problems, J. Comput. Phys. 469 (2022) 111576.

[15] D. Kingma, J. Ba, Adam: a method for stochastic optimization, arXiv:1412.6980, 2014.
[16] D.N. Ku, Blood flow in arteries, Annu. Rev. Fluid Mech. 29 (1997) 399–434.
[17] D.C. Liu, J. Nocedal, On the limited memory BFGS method for large scale optimization, Math. Program. 45 (1989) 503–528.
[18] M.-C. Lai, C.-C. Chang, W.-S. Lin, W.-F. Hu, T.-S. Lin, A shallow Ritz method for elliptic problems with singular sources, J. Comput. Phys. 469 (2022) 111547.
[19] M.-C. Lai, H.-C. Tseng, A simple implementation of the immersed interface methods for Stokes flows with singular forces, Comput. Fluids 37 (2008) 99–106.
[20] R.J. Leveque, Z. Li, The immersed interface method for elliptic equations with discontinuous coefficients and singular sources, SIAM J. Numer. Anal. 31 (1994) 1019–1044.
[21] Y. Liao, P. Ming, Deep Nitsche method: deep Ritz method with essential boundary conditions, Commun. Comput. Phys. 29 (2021) 1365–1384.
[22] X.-D. Liu, R.P. Fedkiw, M. Kang, A boundary condition capturing method for Poisson's equation on irregular domains, J. Comput. Phys. 160 (2000) 151–178.
[23] Y. Liu, Y. Mori, Properties of discrete delta functions and local convergence of the immersed boundary method, SIAM J. Numer. Anal. 50 (2012) 2986–3015.
[24] M.D. McKay, R.J. Beckman, W.J. Conover, A comparison of three methods for selecting values of input variables in the analysis of output from a computer code, Technometrics 21 (1979) 239–245.
[25] J.J. Moré, The Levenberg-Marquardt Algorithm: Implementation and Theory, Numerical Analysis, Springer, Berlin, Heidelberg, 1978, pp. 105–116.
[26] Ömer Oruç, An efficient meshfree method based on Pascal polynomials and multiple-scale approach for numerical solution of 2-D and 3-D second order elliptic interface problems, J. Comput. Phys. 428 (2021) 110070.
[27] R.W. O'Brien, L.R. White, Electrophoretic mobility of a spherical colloidal particle, J. Chem. Soc. Faraday Trans. 74 (1978) 1607–1626.
[28] K.-L. Pan, Y.-H. Tseng, J.-C. Chen, K.-L. Huang, C.-H. Wang, M.-C. Lai, Controlling droplet bouncing and coalescence with surfactant, J. Fluid Mech. 799 (2016) 603–636.
[29] C.S. Peskin, Numerical analysis of blood flow in the heart, J. Comput. Phys. 25 (1977) 220–252.
[30] P.O. Persson, G. Strang, A simple mesh generator in MATLAB, SIAM Rev. Soc. Ind. Appl. Math. 46 (2004) 329–345.
[31] C.S. Peskin, The immersed boundary method, Acta Numer. 11 (2002) 479–517.
[32] A. Pinkus, Approximation theory of the MLP model in neural networks, Acta Numer. 8 (1999) 143–195.
[33] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: an imperative style, high-performance deep learning library, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (Eds.), Adv. Neural. Inf. Process Syst., vol. 32, Curran Associates, Inc., 2019, pp. 8024–8035.
[34] M. Raissi, P. Perdikaris, 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) 686–707.
[35] Y. Shin, J. Darbon, G.E. Karniadakis, On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs, Commun. Comput. Phys. 28 (2020) 2042–2074.
[36] J.S. Stroud, S.A. Berger, D. Saloner, Numerical analysis of flow through a severely stenotic carotid artery bifurcation, J. Biomech. Eng. 124 (2002) 9–20.
[37] S. Tanguy, A. Berlemont, Application of a level set method for simulation of droplet collisions, Int. J. Multiph. Flow 31 (2005) 1015–1035.
[38] M.K. Transtrum, J.P. Sethna, Improvements to the Levenberg-Marquardt algorithm for nonlinear least-squares minimization, arXiv:1201.5885, 2012.
[39] Z. Wang, Z. Zhang, A mesh-free method for interface problems using the deep learning approach, J. Comput. Phys. 400 (2019) 108963.
[40] S. Wu, B. Lu, INN: interfaced neural networks as an accessible meshless approach for solving interface PDE problems, J. Comput. Phys. 470 (2022) 111588.