A Discontinuity and Cusp Capturing PINN for Stokes Interface Problems with Discontinuous Viscosity and Singular Forces

Yu-Hau Tseng[1] and Ming-Chih Lai[2,*]

[1] *Department of Applied Mathematics, National University of Kaohsiung, Kaohsiung 81148, Taiwan*
[2] *Department of Applied Mathematics, National Yang Ming Chiao Tung University, 1001 Ta Hsueh Road, Hsinchu 300, Taiwan*

Dedicated to the memory of Professor Zhongci Shi

Abstract. In this paper, we present a discontinuity and cusp capturing physics-informed neural network (PINN) to solve Stokes equations with a piecewise-constant viscosity and singular force along an interface. We first reformulate the governing equations in each fluid domain separately and replace the singular force effect with the traction balance equation between solutions in two sides along the interface. Since the pressure is discontinuous and the velocity has discontinuous derivatives across the interface, we hereby use a network consisting of two fully-connected sub-networks that approximate the pressure and velocity, respectively. The two sub-networks share the same primary coordinate input arguments but with different augmented feature inputs. These two augmented inputs provide the interface information, so we assume that a level set function is given and its zero level set indicates the position of the interface. The pressure sub-network uses an indicator function as an augmented input to capture the function discontinuity, while the velocity sub-network uses a cusp-enforced level set function to capture the derivative discontinuities via the traction balance equation. We perform a series of numerical experiments to solve two- and three-dimensional Stokes interface problems and perform an accuracy comparison with the augmented immersed interface methods in literature. Our results indicate that even a shallow network with a moderate number of neurons and

*Corresponding author.
Email:* `mclai@math.nctu.edu.tw` (M.-C. Lai)

sufficient training data points can achieve prediction accuracy comparable to that of immersed interface methods.

AMS subject classifications: 65N99, 76D07

Key words: Stokes interface problem, immersed interface method, level set function, physics-informed neural network, discontinuity capturing shallow neural network, cusp-capturing neural network.

1   Introduction

The incompressible two-phase flow problems have a wide range of applications in various scientific and engineering fields; see [7] and the references therein. Very often, this kind of problem involves solving Stokes equations with a discontinuous viscosity and singular force (such as surface tension force) along the fluid interface. For simplicity, we call this problem the Stokes interface problem hereafter. Solving Stokes interface problems numerically is known to be quite challenging in the scientific computing community, giving rise to several difficulties. One major difficulty is to handle the coupling between two adjacent fluids and the interfacial boundary conditions (arising from the singular force). Another difficulty comes from that the pressure is, in fact, discontinuous, and the partial derivatives of the velocity are also discontinuous across the interface. Thus, careful numerical treatments must be adopted for accurate discretizations near the interface, regardless of using the finite difference or finite element method.

In order to address the above issues, Peskin proposed the immersed boundary (IB) method [16], which introduces a regularized version of the discrete delta function to discretize the singular force in a Cartesian grid setting. The discontinuous viscosity can also be regularized by a smoothed version of the Heaviside function. In this framework, the problem becomes smooth so that regular finite difference discretizations such as the MAC scheme can be applied to solve the Stokes equations. However, it is known that the IB method achieves only first-order accuracy [12] for the solution variables. To improve the accuracy, an alternative approach is to solve the Stokes equations within each fluid domain separately and reformulate the singular force term into the traction balance equation on the interface. The above two formulations will be given in Section 2. In this paper, we shall introduce a neural network methodology to solve the problem based on the second formulation. Since the present goal here is to make comparisons with traditional methods such as immersed interface method (IIM, also based on the second formulation) from the

implementation and accuracy aspects, we provide a brief review of the IIM in the following.

The immersed interface method proposed by LeVeque and Li [9] was originally designed to solve elliptic interface problems with discontinuous coefficients. Their approach incorporates the derived jump conditions via local coordinates into the finite difference discretization so that the local truncation error can be reduced appropriately to achieve the solution in second-order accuracy in the L^∞ norm. Lai et al. [5,8] later proposed a simplified version of the IIM that directly utilizes jump conditions to achieve second-order accuracy in the L^∞ norm without resorting to local coordinates. Of course, there are other variants of solving elliptic interface problems using the jump conditions, such as the ghost fluid method [1,11], or meshless method [15]. However, it is not our intention to give an exhaustive review here so readers who are interested in the recent results can refer to [15], and the references therein. Comparing with the well-developed solvers for elliptic interface problems, there are relatively few works to investigate the present Stokes interface problems with discontinuous viscosity and singular forces. As we mentioned earlier, the major difficulty is that the solution and derivative discontinuities are all coupled, making it hard to develop accurate numerical methods. In [10], Li et al. developed a new IIM-based method for the two-dimensional problem that decouples the jump conditions of the fluid variables by introducing two augmented velocity variables. Although the new augmented system of equations can be respectively written as three Poisson interface problems for the pressure and the augmented velocity, they are still coupled through the augmented variable jumps defined on the interface. As a result, the GMRES iterative method is used to solve the Schur complement system for the augmented variable jumps. Recently, Wang et al. [24] used the similar idea but a simple version of IIM and extended the method to solve the three-dimensional problems.

In addition to traditional grid-based methods, the scientific computing community has shown increasing interest in utilizing deep neural networks to tackle elliptic interface problems. One advantage of the neural network approach is its mesh-free nature, enabling us to handle problems with complex interfaces or irregular domains more easily. However, with the common usage of the smooth activation function in neural network methods, the network solution is naturally smooth and cannot capture the solution behaviors (function discontinuity or derivative discontinuity) sharply. One simple way to circumvent such difficulty is to decompose the computational domain into two sub-domains and employs separate networks for each sub-domain such as the networks in [2,4,23]. Recently, the authors have developed a discontinuity capturing shallow neural network to solve the elliptic interface problems [6]. The main idea is to represent a d-dimensional discontinuous function by a

single $(d+1)$-dimensional continuous neural network function in which the extra augmented variable labels the sub-domains. Once the function representation is chosen, the rest of the method follows the physics-informed neural network (PINN) framework developed in [20]. Meanwhile, we have also developed a $(d+1)$-dimensional cusp capturing neural network [21] to represent a d-dimensional continuous function but with discontinuous partial derivatives across the interface. In such a case, the extra augmented variable uses the cusp-enforced level set function instead. As we mentioned earlier, in the present Stokes interface problem, the pressure is usually discontinuous, and the partial derivatives of velocity are discontinuous, so they are well represented by the discontinuity capturing and cusp capturing networks, respectively. Both network function representations will be given in detail in Section 3.

The remainder of the paper is organized as follows. In Section 2, we first present the Stokes equations with a discontinuity viscosity and singular force. Then we reformulate the governing equations in each sub-domain and introduce the traction balance equation to replace the singular force effect. In Section 3, we introduce the discontinuity and cusp capturing PINN for solving the model problems. Numerical experiments demonstrating the accuracy of the proposed method compared with the existing augmented immersed interface methods are shown in Section 4. Some concluding remarks are given in Section 5.

2   Stokes equations with a discontinuous viscosity and singular force

In this paper, we aim to solve a d-dimensional incompressible Stokes system with a piecewise-constant viscosity and singular force on an embedded interface. This problem arises from the simulation of the flow of two adjacent fluids with different viscosity in the presence of interfacial force. We start by writing down the governing equations in the immersed boundary (IB) formulation [16], which regards the interface as a singular force generator so that the equations can be described in the whole fluid domain Ω. To begin with, let $\Omega \subset \mathbb{R}^d$, where $d=2$ or 3, be a bounded domain, and Γ be an embedded co-dimensional one C^1-interface that separates Ω into two subdomains, Ω^- and Ω^+, such that $\Omega = \Omega^- \cup \Omega^+ \cup \Gamma$. This Stokes interface problem can be written as in [10, 24]

$$\nabla p = \nabla \cdot \left(\mu \left(\nabla \mathbf{u} + \nabla \mathbf{u}^T \right) \right) + \mathbf{f} + \mathbf{g} \qquad \text{in } \Omega, \qquad (2.1\text{a})$$

$$\nabla \cdot \mathbf{u} = 0 \qquad \text{in } \Omega, \qquad (2.1\text{b})$$

$$\mathbf{u} = \mathbf{u}_b \qquad \text{on } \partial\Omega. \qquad (2.1\text{c})$$

Here, $\mathbf{u}=(u_1,\cdots,u_d)\in\mathbb{R}^d$ is the fluid velocity, p is the pressure, and $\mathbf{g}=(g_1,\cdots,g_d)$ is an external force, which are all functions of $\mathbf{x}=(x_1,\cdots x_d)\in\Omega$. The viscosity $\mu(\mathbf{x})$ is defined piecewise-constantly such that $\mu(\mathbf{x})=\mu^\pm$ if $\mathbf{x}\in\Omega^\pm$. The force term $\mathbf{f}$ is a singular concentrated body force expressed in the IB formulation as $\mathbf{f}(\mathbf{x})=\int_\Gamma \mathbf{F}(\mathbf{X})\delta(\mathbf{x}-\mathbf{X})dS$, where the interfacial force $\mathbf{F}=(F_1,\ldots,F_d)$ and $\delta(\mathbf{x})$ is the d-dimensional Dirac delta function. Due to the delta function singularity, one can immediately see that the pressure and velocity should not be smooth mathematically. In fact, one can reformulate the governing equations (2.1a)-(2.1c) in each fluid domain $\Omega^\pm$ separately, but the solutions are linked by the traction balance equation (2.2c) on the interface Γ as follows.

$$-\nabla p+\mu\Delta\mathbf{u}+\mathbf{g}=\mathbf{0} \qquad \text{in } \Omega^\pm, \qquad (2.2a)$$

$$\nabla\cdot\mathbf{u}=0 \qquad \text{in } \Omega^\pm, \qquad (2.2b)$$

$$[\![-p\mathbf{I}+\mu(\nabla\mathbf{u}+\nabla\mathbf{u}^T)]\!]\cdot\mathbf{n}+\mathbf{F}=\mathbf{0} \qquad \text{on } \Gamma, \qquad (2.2c)$$

$$\mathbf{u}=\mathbf{u}_b \qquad \text{on } \partial\Omega. \qquad (2.2d)$$

Here, the notation $[\![\cdot]\!]$ represents the jump of a quantity across the interface; that is, for any $\mathbf{x}_\Gamma\in\Gamma$,

$$[\![f]\!](\mathbf{x}_\Gamma)=\lim_{\mathbf{x}\in\Omega^+,\mathbf{x}\to\mathbf{x}_\Gamma}f(\mathbf{x})-\lim_{\mathbf{x}\in\Omega^-,\mathbf{x}\to\mathbf{x}_\Gamma}f(\mathbf{x})=f^+(\mathbf{x}_\Gamma)-f^-(\mathbf{x}_\Gamma). \qquad (2.3)$$

The unit outward normal vector, $\mathbf{n}=(n_1,\cdots,n_d)$, points from Ω^- to Ω^+ along the interface Γ. This above traction balance equation (2.2c) can be derived in [19].

Since the fluid is viscous, the velocity is continuous across the interface. However, the traction balance equation (2.2c) shows that the pressure is discontinuous and the velocity has discontinuous partial derivatives. Indeed, we can rewrite Eq. (2.2c) component-wisely as

$$-[\![p]\!]n_k+\left[\!\!\left[\mu\frac{\partial u_k}{\partial\mathbf{n}}\right]\!\!\right]+\left[\!\!\left[\mu\frac{\partial\mathbf{u}}{\partial x_k}\right]\!\!\right]\cdot\mathbf{n}+F_k=0, \quad k=1,\ldots,d. \qquad (2.4)$$

In the next section, we shall present a neural network learning methodology based on PINN [20] to solve Eqs. (2.2a)-(2.2b) with the traction balance equation (2.4), subject to the boundary condition (2.2d). One can see from Eq. (2.4) that there are d essential interface jump conditions in the present neural network method. On the contrary, the implementation of IIM in [10] requires 6 interface jump conditions for

solving the problem with the dimension $d=2$ as follows.

$$-[\![p]\!]+2\left[\!\!\left[\mu\frac{\partial \mathbf{u}}{\partial \mathbf{n}}\right]\!\!\right]\cdot\mathbf{n}+\mathbf{F}\cdot\mathbf{n}=0, \tag{2.5a}$$

$$-\left[\!\!\left[\frac{\partial p}{\partial \mathbf{n}}\right]\!\!\right]+[\![\mathbf{g}\cdot\mathbf{n}]\!]+\frac{\partial}{\partial \boldsymbol{\tau}}(\mathbf{F}\cdot\boldsymbol{\tau})+2\left[\!\!\left[\mu\frac{\partial^2}{\partial \boldsymbol{\tau}^2}(\mathbf{u}\cdot\mathbf{n})\right]\!\!\right]=0, \tag{2.5b}$$

$$[\![\mathbf{u}]\!]=\mathbf{0}, \quad [\![\mu\nabla\cdot\mathbf{u}]\!]=0, \tag{2.5c}$$

$$\left[\!\!\left[\mu\frac{\partial \mathbf{u}}{\partial \mathbf{n}}\right]\!\!\right]\cdot\boldsymbol{\tau}+\left[\!\!\left[\mu\frac{\partial \mathbf{u}}{\partial \boldsymbol{\tau}}\right]\!\!\right]\cdot\mathbf{n}+\mathbf{F}\cdot\boldsymbol{\tau}=0. \tag{2.5d}$$

Here, $\boldsymbol{\tau}$ is the tangential vector along the interface Γ. For the three-dimensional case, one needs to use even 9 interface jump conditions of velocity and pressure in order to implement the augmented IIM in [24]. Thus, it is more advantageous to use the present neural network method to solve the Stokes interface problems than the augmented IIMs regarding the interface jump conditions needed, not to mention the latter ones need to solve the resultant linear systems that lack good symmetric properties.

3   A discontinuity and cusp capturing PINN

In this section, we present a discontinuity and cusp capturing physics-informed neural network to solve Stokes equations (2.2a)-(2.2b) with the traction balance equation (2.4), subject to the boundary condition (2.2d). To proceed, we assume there exists a smooth level set function $\phi(\mathbf{x})$ such that the interior and exterior domains are defined as $\Omega^-=\{\mathbf{x}\in\Omega\,|\,\phi(\mathbf{x})<0\}$ and $\Omega^+=\{\mathbf{x}\in\Omega\,|\,\phi(\mathbf{x})>0\}$, respectively, with the interface position given by the zero level set, $\Gamma=\{\mathbf{x}\in\Omega\,|\,\phi(\mathbf{x})=0\}$.

Since the pressure has jump discontinuity across the interface Γ, we follow the idea proposed in [6] and represent the d-dimensional discontinuous function $p(\mathbf{x})$ by a $(d+1)$-dimensional continuous neural network function $P(\mathbf{x},y)$, where $\mathbf{x}\in\Omega$ and $y\in\mathbb{R}$. We then set the augmented variable y to be the indicator function $I(\mathbf{x})$ as

$$I(\mathbf{x})=\begin{cases} -1, & \mathbf{x}\in\Omega^-, \\ 1, & \mathbf{x}\in\Omega^+. \end{cases} \tag{3.1}$$

So the pressure $p(\mathbf{x})$ in $\Omega^\pm$ can be represented by the neural network function $P(\mathbf{x},y=\pm 1)$. We point out that the neural network P is a smooth extension in $\mathbb{R}^{d+1}$ if a smooth activation function is used. The pressure $p(\mathbf{x})$ in Ω^- and Ω^+ can be regarded as the network function P restricted on two disjoint d-dimensional hyperplanes, $S^-=\{(\mathbf{x},-1)\,|\,\mathbf{x}\in\Omega^-\}\subset\mathbb{R}^{d+1}$ and $S^+=\{(\mathbf{x},1)\,|\,\mathbf{x}\in\Omega^+\}\subset\mathbb{R}^{d+1}$, respectively. Consequently,

one can rewrite the pressure jump $[\![p]\!]$ as the difference of the continuous network function P evaluated at the interface position on the two disjoint hyperplanes, $S^\pm$. That is, for any $\mathbf{x}_\Gamma \in \Gamma$,

$$[\![p]\!](\mathbf{x}_\Gamma) = P(\mathbf{x}_\Gamma, 1) - P(\mathbf{x}_\Gamma, -1) = [\![P]\!](\mathbf{x}_\Gamma, 0). \tag{3.2}$$

The gradient of p in $\Omega^\pm$ can be written by

$$\nabla p(\mathbf{x}) = \nabla_{\mathbf{x}} P(\mathbf{x}, I(\mathbf{x})) + \partial_y P(\mathbf{x}, I(\mathbf{x})) \nabla I(\mathbf{x}) = \nabla_{\mathbf{x}} P(\mathbf{x}, I(\mathbf{x})), \tag{3.3}$$

where $\nabla_{\mathbf{x}} P \in \mathbb{R}^d$ represents a vector consisting of the partial derivatives of P with respect to the components in $\mathbf{x}$, and $\partial_y P$ is the shorthand notation of partial derivative of P with respect to the augmented variable y.

Similarly, following the idea in [21], we represent the d-dimensional non-smooth velocity function $\mathbf{u}(\mathbf{x})$ by a $(d+1)$-dimensional smooth neural network function $\mathbf{U}(\mathbf{x}, z) = (U_1(\mathbf{x}, z), \cdots U_d(\mathbf{x}, z))$. We then set the augmented variable z to be $z = \phi_a(\mathbf{x}) = |\phi(\mathbf{x})|$ so the velocity $\mathbf{u}(\mathbf{x})$ in Ω can be represented by a neural network $\mathbf{U}(\mathbf{x}, z = \phi_a(\mathbf{x}))$. Here, we call $\phi_a(\mathbf{x})$ as a cusp-enforced level set function since it exhibits a gradient jump defined as $[\![\nabla \phi_a]\!](\mathbf{x}_\Gamma) = 2\nabla \phi(\mathbf{x}_\Gamma)$ for $\mathbf{x}_\Gamma \in \Gamma$. Based on the above property, one can immediately see that the gradient of velocity $\nabla u_k = \nabla_{\mathbf{x}} U_k + \partial_z U_k \nabla \phi_a$, $k = 1, \ldots, d$ in $\Omega^\pm$ has discontinuity jump across the interface which inherits the cusp-like solution behavior of the velocity. More precisely, the jump of $\mu \nabla u_k$ across the interface can be derived directly as

$$\begin{aligned}
[\![\mu \nabla u_k]\!](\mathbf{x}_\Gamma) &= [\![\mu \nabla_{\mathbf{x}} U_k]\!](\mathbf{x}_\Gamma) + [\![\mu \partial_z U_k \nabla \phi_a]\!](\mathbf{x}_\Gamma) \\
&= [\![\mu]\!] \nabla_{\mathbf{x}} U_k(\mathbf{x}_\Gamma) + (\mu^+ + \mu^-) \partial_z U_k(\mathbf{x}_\Gamma) \nabla \phi(\mathbf{x}_\Gamma),
\end{aligned} \tag{3.4}$$

for $k = 1, \ldots, d$. Similarly, the jump of $\mu \frac{\partial \mathbf{u}}{\partial x_k}$ can be calculated by

$$\begin{aligned}
\left[\!\!\left[\mu \frac{\partial \mathbf{u}}{\partial x_k} \right]\!\!\right](\mathbf{x}_\Gamma) &= \left[\!\!\left[\mu \frac{\partial \mathbf{U}}{\partial x_k} \right]\!\!\right](\mathbf{x}_\Gamma) + \left[\!\!\left[\mu \frac{\partial \mathbf{U}}{\partial z} \frac{\partial \phi_a}{\partial x_k} \right]\!\!\right](\mathbf{x}_\Gamma) \\
&= [\![\mu]\!] \frac{\partial \mathbf{U}}{\partial x_k}(\mathbf{x}_\Gamma) + (\mu^+ + \mu^-) \frac{\partial \mathbf{U}}{\partial z} \frac{\partial \phi}{\partial x_k}(\mathbf{x}_\Gamma).
\end{aligned} \tag{3.5}$$

By multiplying the normal vector $\mathbf{n} = \nabla \phi / \|\nabla \phi\|$ to the above equations (3.4) and (3.5), respectively; we obtain the following derivative jumps of $\mathbf{u}$ as follows

$$\left[\!\!\left[\mu \frac{\partial u_k}{\partial \mathbf{n}} \right]\!\!\right](\mathbf{x}_\Gamma) = [\![\mu]\!] (\nabla_{\mathbf{x}} U_k \cdot \mathbf{n})(\mathbf{x}_\Gamma) + (\mu^+ + \mu^-) \frac{\partial U_k}{\partial z} \|\nabla \phi\|(\mathbf{x}_\Gamma), \tag{3.6a}$$

$$\left[\!\!\left[\mu \frac{\partial \mathbf{u}}{\partial x_k} \right]\!\!\right] \cdot \mathbf{n}(\mathbf{x}_\Gamma) = [\![\mu]\!] \left(\frac{\partial \mathbf{U}}{\partial x_k} \cdot \mathbf{n} \right)(\mathbf{x}_\Gamma) + (\mu^+ + \mu^-) \left(\frac{\partial \mathbf{U}}{\partial z} \cdot \mathbf{n} \right) \frac{\partial \phi}{\partial x_k}(\mathbf{x}_\Gamma), \tag{3.6b}$$

for $k=1,\dots,d$. Therefore, the interfacial velocity derivative jump terms in Eq. (2.4) can be expressed by the smooth network function $\mathbf{U}(\mathbf{x},z)$ with the assistance of the level set function ϕ.

After careful calculation, the Laplacian of u_k in $\Omega^{\pm}$ can also be explicitly expressed in terms of U_k and ϕ_a as

$$\mu\Delta u_k = \mu\left(\Delta_{\mathbf{x}}U_k + 2\nabla\phi_a\cdot\nabla_{\mathbf{x}}(\partial_z U_k) + \|\nabla\phi_a\|^2\partial_{zz}U_k + \partial_z U_k\Delta\phi_a\right), \qquad (3.7)$$

where $\Delta_{\mathbf{x}}$ represents the Laplace operator applied only to the variable $\mathbf{x}$. Since the neural network function $\mathbf{U}$ is smooth, the continuity of velocity vector holds automatically, and calculating the derivatives of the network $\mathbf{U}$ with respect to its input variables $\mathbf{x}$ and z via automatic differentiation [3] is straightforward.

Next, we express the Stokes equations (2.2a)-(2.2b) in terms of P and $\mathbf{U}$ as follows. To simplify, we introduce the notation $\mathcal{L}U_k$ to represent the right-hand side of $\mu\Delta u_k$ in Eq. (3.7) so that Eqs. (2.2a) and (2.2b) are written as

$$-\frac{\partial P}{\partial x_k}(\mathbf{x},I(\mathbf{x})) + \mathcal{L}U_k(\mathbf{x},\phi_a(\mathbf{x})) + g_k(\mathbf{x}) = 0, \qquad \mathbf{x}\in\Omega^{\pm}, \quad k=1,\dots,d, \qquad (3.8a)$$

$$\nabla_{\mathbf{x}}\cdot\mathbf{U}(\mathbf{x},\phi_a(\mathbf{x})) + \frac{\partial\mathbf{U}}{\partial z}(\mathbf{x},\phi_a(\mathbf{x}))\cdot\nabla\phi_a(\mathbf{x}) = 0, \qquad \mathbf{x}\in\Omega^{\pm}. \qquad (3.8b)$$

We also denote the sum of the right-hand sides of Eqs. (3.6a)-(3.6b) by $\mathcal{D}_k\mathbf{U}$, so the traction balance equation (2.4) has the form as

$$-[\![P]\!](\mathbf{x}_{\Gamma},0)n_k(\mathbf{x}_{\Gamma}) + \mathcal{D}_k\mathbf{U}(\mathbf{x}_{\Gamma},0) + F_k(\mathbf{x}_{\Gamma}) = 0, \quad \mathbf{x}_{\Gamma}\in\Gamma, \quad k=1,\dots,d. \qquad (3.9)$$

Regarding the boundary condition Eq. (2.2d), we use $\mathbf{x}_B$ to represent the points located on $\partial\Omega$ so it becomes

$$U_k(\mathbf{x}_B,\phi_a(\mathbf{x}_B)) - u_{b,k}(\mathbf{x}_B) = 0, \quad \mathbf{x}_B\in\partial\Omega, \quad k=1,\dots,d. \qquad (3.10)$$

Next, we aim to show the network architecture for the pressure $P(\mathbf{x},I(\mathbf{x}))$ and the velocity $\mathbf{U}(\mathbf{x},\phi_a(\mathbf{x}))$. Fig. 1 illustrates the present neural network structure, where $(\mathbf{x},I(\mathbf{x}))\in\mathbb{R}^{d+1}$ represents the $d+1$ feature input for the sub-network P, while $(\mathbf{x},\phi_a(\mathbf{x}))\in\mathbb{R}^{d+1}$ represents the input for the sub-network $\mathbf{U}$. Although the two sub-networks have separate structures, their output layers are fed into the same loss function, $\text{Loss}(\boldsymbol{\theta})$, which will be given in detail later. For simplicity, we assume that both sub-networks have the same depth number, L, and we use the subscripts $\star={'}p{'}$, ${'}u{'}$ to distinguish which sub-network is present. In each sub-network, we label the input layer as layer 0 and denote the feature input as $\mathbf{v}_{\star}^{[0]}=(\mathbf{x},I(\mathbf{x}))^T$ or $(\mathbf{x},\phi_a(\mathbf{x}))^T$, a column vector in $\mathbb{R}^{d+1}$. The output at the ℓ-th hidden layer, consisting

of $N_\star^{[\ell]}$ neurons, denoted as $\mathbf{v}_\star^{[\ell]} \in \mathbb{R}^{N_\star^{[\ell]}}$, represents an affine mapping of the output of layer $\ell-1$ (i.e., $\mathbf{v}_\star^{[\ell-1]}$), followed by an element-wise activation function σ as

$$\mathbf{v}_\star^{[\ell]} = \sigma\left(W_\star^{[\ell]}\mathbf{v}_\star^{[\ell-1]} + \mathbf{b}_\star^{[\ell]}\right), \quad \ell=1,\cdots,L. \tag{3.11}$$

Here, $W_\star^{[\ell]} \in \mathbb{R}^{N_\star^{[\ell]} \times N_\star^{[\ell-1]}}$ represents the weight matrix connecting layer $\ell-1$ to layer ℓ, and $\mathbf{b}_\star^{[\ell]} \in \mathbb{R}^{N_\star^{[\ell]}}$ is the bias vector at layer ℓ. Finally, the output of this L-hidden-layer network is denoted by the following

$$P_\mathcal{N}(\mathbf{x},I(\mathbf{x});\boldsymbol{\theta}_p) = W_p^{[L+1]}\mathbf{v}_p^{[L]}, \tag{3.12a}$$

$$\mathbf{U}_\mathcal{N}(\mathbf{x},\phi_a(\mathbf{x});\boldsymbol{\theta}_u) = W_u^{[L+1]}\mathbf{v}_u^{[L]}, \tag{3.12b}$$

where $W_p^{[L+1]} \in \mathbb{R}^{1 \times N_p^{[L]}}$ and $W_u^{[L+1]} \in \mathbb{R}^{d \times N_u^{[L]}}$. The notation $\boldsymbol{\theta}_\star$ denotes the vector collecting all trainable parameters (including all the weights and biases) in each corresponding sub-network. We use $\boldsymbol{\theta} = (\boldsymbol{\theta}_p, \boldsymbol{\theta}_u)$ to represent the collection of all trainable parameters within the whole network, and the dimension of $\boldsymbol{\theta}$ is counted as

$$N_{\boldsymbol{\theta}} = \left(N_p^{[L]} + \sum_{\ell=1}^{L}(N_p^{[\ell-1]}+1)N_p^{[\ell]}\right) + \left(N_u^{[L]}d + \sum_{\ell=1}^{L}(N_u^{[\ell-1]}+1)N_u^{[\ell]}\right). \tag{3.13}$$

In the training process, we randomly choose M_I points, $\{\mathbf{x}^i\}_{i=1}^{M_I}$, within the region of $\Omega^- \cup \Omega^+$, and M_B points, $\{\mathbf{x}_B^i\}_{i=1}^{M_B}$, on the boundary of the domain $\partial\Omega$, respectively. Additionally, we select M_Γ points, $\{\mathbf{x}_\Gamma^i\}_{i=1}^{M_\Gamma}$, on the interface Γ. Altogether, we have a total number of $M = M_I + M_B + M_\Gamma$ training points. Within the PINN learning framework, the loss function consists of the mean squared errors of the residual of differential equations (3.8a)-(3.8b), the traction balance equation (3.9), and the boundary condition (3.10). That is,

$$\text{Loss}(\boldsymbol{\theta}) = \frac{1}{M_I}\sum_{i=1}^{M_I}\left(\sum_{k=1}^{d}\left|L_{I,k}(\mathbf{x}^i,I(\mathbf{x}^i),\phi_a(\mathbf{x}^i);\boldsymbol{\theta})\right|^2 + \left|L_D(\mathbf{x}^i,\phi_a(\mathbf{x}^i);\boldsymbol{\theta}_u)\right|^2\right)$$

$$+ \frac{c_\Gamma}{M_\Gamma}\sum_{i=1}^{M_\Gamma}\sum_{k=1}^{d}\left|L_{\Gamma,k}(\mathbf{x}_\Gamma^i,0;\boldsymbol{\theta})\right|^2 + \frac{c_B}{M_B}\sum_{i=1}^{M_B}\sum_{k=1}^{d}\left|L_{B,k}(\mathbf{x}_B^i,\phi_a(\mathbf{x}_B^i);\boldsymbol{\theta}_u)\right|^2, \tag{3.14}$$

where the residual error L_I, incompressibility constraint error L_D, traction balance condition error L_Γ, and boundary condition error L_B, are shown respectively as

follows:

$$L_{I,k}(\mathbf{x},I(\mathbf{x}),\phi_a(\mathbf{x});\boldsymbol{\theta})=-\frac{\partial P_{\mathcal{N}}}{\partial x_k}(\mathbf{x},I(\mathbf{x});\boldsymbol{\theta}_p)+\mathcal{L}U_{\mathcal{N},k}(\mathbf{x},\phi_a(\mathbf{x});\boldsymbol{\theta}_u)-g_k(\mathbf{x}), \quad \text{(3.15a)}$$

$$L_D(\mathbf{x},\phi_a(\mathbf{x});\boldsymbol{\theta}_u)=\nabla_{\mathbf{x}}\cdot\mathbf{U}_{\mathcal{N}}(\mathbf{x},\phi_a(\mathbf{x});\boldsymbol{\theta}_u)+\frac{\partial\mathbf{U}_{\mathcal{N}}}{\partial z}(\mathbf{x},\phi_a(\mathbf{x});\boldsymbol{\theta}_u)\cdot\nabla\phi_a(\mathbf{x}), \quad \text{(3.15b)}$$

$$L_{\Gamma,k}(\mathbf{x}_\Gamma,0;\boldsymbol{\theta})=-[\![P_{\mathcal{N}}]\!](\mathbf{x}_\Gamma,0;\boldsymbol{\theta}_p)n_k(\mathbf{x}_\Gamma)+\mathcal{D}_k\mathbf{U}_{\mathcal{N}}(\mathbf{x}_\Gamma,0;\boldsymbol{\theta}_u)+F_k(\mathbf{x}_\Gamma), \quad \text{(3.15c)}$$

$$L_{B,k}(\mathbf{x}_B,\phi_a(\mathbf{x}_B);\boldsymbol{\theta}_u)=U_{\mathcal{N},k}(\mathbf{x}_B,\phi_a(\mathbf{x}_B);\boldsymbol{\theta}_u)-u_{b,k}(\mathbf{x}_B). \quad \text{(3.15d)}$$

The constants c_B and c_Γ appeared in the loss function (3.14) are chosen to balance the contribution of the terms related to the traction balance equation (3.9), and the boundary condition (3.10), respectively.

4   Numerical results

In this section, we perform a series of accuracy tests for the proposed discontinuity and cusp capturing neural network method on solving two- and three-dimensional Stokes interface problems described by Eqs. (2.2a)-(2.2d). We set the penalty constants in the loss function, $c_\Gamma = c_B = 1$. This approach enables the use of smooth neural network functions defined in $\mathbb{R}^{d+1}$, denoted by $P(\mathbf{x},y)$ and $\mathbf{U}(\mathbf{x},z)$, to learn discontinuous and non-smooth solutions in $\mathbb{R}^d$, $p(\mathbf{x})$ and $\mathbf{u}(\mathbf{x})$, respectively. We denote the network-predicted solutions as $P_{\mathcal{N}}$ and $\mathbf{U}_{\mathcal{N}}=(U_{\mathcal{N},1},\cdots,U_{\mathcal{N},d})$. Throughout this section, we represent the interface Γ using the zero level set of a suitable level set function $\phi(\mathbf{x})$. We use the indicator function $I(\mathbf{x})$ and the cusp-enforced level set function $\phi_a(\mathbf{x})=|\phi(\mathbf{x})|$ as the augmented input for the network of $P_{\mathcal{N}}$ and $\mathbf{U}_{\mathcal{N}}$, respectively.

To ensure the C^2-regularity of $\mathbf{u}(\mathbf{x})$ in each subdomain, we employ the sigmoid function $\sigma(x)=\frac{1}{1+e^{-x}}$ as the activation function. For the numerical examples presented here (except the last example whose solution is not available, so deeper neural network approximations are used), we adopt a completely shallow network structure ($L=1$) with N_p and N_u neurons for the pressure and velocity, respectively, as shown in the network architecture Fig. 1. So the number of total hyper-parameters to be trained is $N_{\boldsymbol{\theta}}=(d+3)N_p+2(d+1)N_u$. The training and test data points are generated using the Latin hypercube sampling algorithm [14]. This algorithm effectively avoids clustering data points at specific locations, resulting in a nearly random sampling. To assess the accuracy of the network solution, we select M_{test} points (distinct from the training points) within Ω and calculate the L^∞ errors as follows:

$$E_p^\infty=\|p-P_{\mathcal{N}}\|_\infty, \quad E_{\mathbf{u}}^\infty=\frac{1}{d}\sum_{k=1}^d\|u_k-U_{\mathcal{N},k}\|_\infty.$$

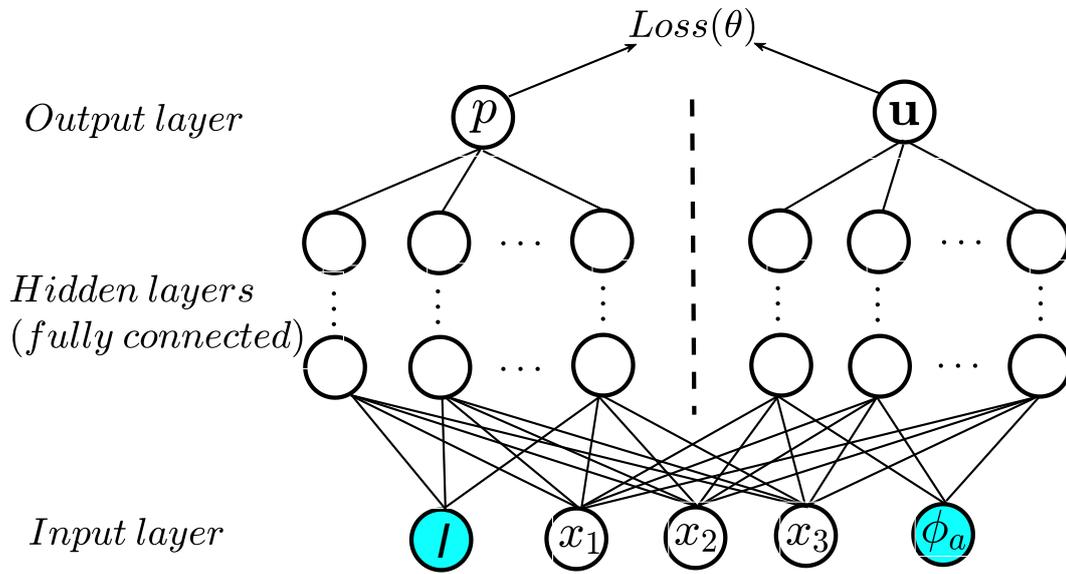Figure 1: Diagram of the L-hidden-layer network structure.

Here, the L^∞ error of a function f is defined as $\|f\|_\infty = \max_{1 \leq i \leq M_{test}} |f(\mathbf{x}^i)|$. We set the number of test points for each experiment to be $M_{test} = 100M$, where M represents the total number of training points used. Since the predicted results will vary slightly due to the randomness of training and test data points and the initialization of trainable parameters, we report the average value of errors and losses over 5 trial runs.

During the training process, we employ the Levenberg-Marquardt (LM) algorithm [13] as the optimizer to train the network and update the damping parameter ν using the strategies outlined in [22]. The training process is terminated either when the loss value Loss($\boldsymbol{\theta}$) falls below a threshold $\epsilon_\theta = 10^{-14}$ or the maximum iteration step $Epoch_{\max} = 3000$ is reached, unless otherwise stated. All experiments are performed on a desktop computer equipped with a single NVIDIA GeForce RTX3060 GPU. We implement the present network architecture using PyTorch (v1.13) [18], and all trainable parameters (i.e., weights and biases) are initialized using PyTorch's default settings. We have made the source code used in the paper available on GitHub at `https://github.com/yuhautseng/`.

Let us remark on the choice of test examples in the following. As mentioned earlier, we aim to emphasize the ease of code implementation for the problem compared with the traditional IIM; thus, we have taken the test examples in 2D and 3D directly from the literature [10] and [24], respectively. One might wonder why the interfaces Γ chosen there are in circular (2D) or spherical (3D) form. The reason

is quite apparent since it is easy to construct the analytic solution under such a circumstance (the solution must satisfy the jump conditions (2.4) for $k=1,\dots d$). Nevertheless, this kind of simple interface shape cannot be regarded as a limitation of the present neural network methodology since it is completely mesh-free. Indeed, it can be applied to more complex interface cases when the analytic solutions are not available (see Example 4.5).

Notice that, it is not completely fair to make a running time comparison between the present neural network method with IIM since the former method is mesh-free while the latter one is grid-based. The present PINN method learns the solution directly while the IIM solves the solution on uniform grids. Besides, the results in [10] and [24] were run in different computing hardware and it is unlikely for us to reproduce the results. To provide some idea of the time performance of the present PINN method, we rerun our code in a slightly faster desktop machine equipped with a single NVIDIA GeForce RTX4090 GPU. The total training time up to 3000 epochs for the two-dimensional case with $(N_p, M_0) = (20, 30)$ (as shown in second row of Table 1) cost approximately 59 seconds while the three-dimensional case with $(N_p, N_u) = (40, 100)$ (as shown in third row of Table 3) cost approximately 253 seconds, respectively.

Example 4.1. We start by solving a two-dimensional Stokes interface problem with a piecewise-constant viscosity and a singular force in a domain $\Omega = [-2, 2]^2$. This example is taken directly from Example 4.2 in [10] so that we can compare the results with the augmented immersed interface method developed there. The interface Γ is simply a unit circle defined as the zero level set of the function $\phi(\mathbf{x}) = x_1^2 + x_2^2 - 1$. We specify the viscosity $\mu(\mathbf{x})$, and the exact solutions $p(\mathbf{x})$ and $\mathbf{u}(\mathbf{x}) = (u_1(\mathbf{x}), u_2(\mathbf{x}))$ as

$$\mu(\mathbf{x}) = \begin{cases} 1 & \text{in} \quad \Omega^-, \\ 0.5 & \text{in} \quad \Omega^+, \end{cases} \qquad p(\mathbf{x}) = \begin{cases} 1 & \text{in} \quad \Omega^-, \\ 0 & \text{in} \ \Omega^+, \end{cases}$$

$$u_1(\mathbf{x}) = \begin{cases} x_2(x_1^2 + x_2^2 - 1) & \text{in} \quad \Omega^-, \\ 0 & \text{in} \quad \Omega^+, \end{cases} \qquad u_2(\mathbf{x}) = \begin{cases} -x_1(x_1^2 + x_2^2 - 1) & \text{in} \quad \Omega^-, \\ 0 & \text{in} \quad \Omega^+. \end{cases}$$

The external force $\mathbf{g}(\mathbf{x}) = (g_1(\mathbf{x}), g_2(\mathbf{x}))$ thus can be derived accordingly as

$$g_1(\mathbf{x}) = \begin{cases} -8x_2 & \text{in} \quad \Omega^-, \\ 0 & \text{in} \quad \Omega^+, \end{cases} \quad g_2(\mathbf{x}) = \begin{cases} 8x_1 & \text{in} \quad \Omega^-, \\ 0 & \text{in} \quad \Omega^+, \end{cases}$$

and the interfacial force $\mathbf{F}(\mathbf{x}_\Gamma) = (F_1(\mathbf{x}_\Gamma), F_2(\mathbf{x}_\Gamma)) = (-x_1 - 2x_2, -x_2 + 2x_1)$ can also be obtained from Eq. (2.4).

As mentioned earlier, we use a completely shallow network structure ($L = 1$) with N_p and N_u neurons for the pressure $P_\mathcal{N}$ and velocity $\mathbf{U}_\mathcal{N}$, respectively. Here,

Table 1: The L^∞ errors of $P_\mathcal{N}$ and $\mathbf{U}_\mathcal{N}$ for Example 4.1. Left panel: the present network result. Here, $N_{\boldsymbol{\theta}} = 17N_p$, $M = M_0(M_0+7)$, $\epsilon_\theta = 10^{-14}$, and $Epoch_{\max} = 3000$. Right panel: IIM results in [10] with grid resolutions 128^2, 256^2, and 512^2.

	Present			IIM [10]		
(N_p, M_0)	E_p^∞	$E_{\mathbf{u}}^\infty$	Loss($\boldsymbol{\theta}$)	Grid No.	E_p^∞	$E_{\mathbf{u}}^\infty$
$(10,20)$	8.94×10^{-5}	1.16×10^{-5}	6.55×10^{-10}	128^2	8.10×10^{-4}	2.27×10^{-4}
$(20,30)$	1.50×10^{-6}	2.73×10^{-7}	2.54×10^{-14}	256^2	2.54×10^{-4}	4.77×10^{-5}
$(30,40)$	4.05×10^{-7}	6.87×10^{-8}	9.04×10^{-15}	512^2	1.41×10^{-5}	1.41×10^{-5}

we choose $N_p = 10, 20, 30$, and $N_u = 2N_p$ so that the total training parameters is $N_{\boldsymbol{\theta}} = 17N_p$. We also vary the training points accordingly by introducing a grid parameter $M_0 = 20, 30, 40$ (can be regarded as the grid number used in traditional grid-based methods) so that the training dataset consists of $M_I = M_0^2$ points in $\Omega^- \cup \Omega^+$, $M_\Gamma = 3M_0$ points on the interface Γ, and $M_B = 4M_0$ points on the boundary $\partial\Omega$. The total number of training points is $M = M_0(M_0+7)$.

Table 1 shows the L^∞ errors of $P_\mathcal{N}$ and $\mathbf{U}_\mathcal{N}$, as well as the corresponding training losses. We also show the results obtained by IIM in [10] using various grid resolutions in the right panel for comparison. It can be observed that the proposed network method achieves prediction accuracy on the order of $\mathcal{O}(10^{-5})$ to $\mathcal{O}(10^{-8})$ as (N_p, M_0) varies from $(10,20)$ to $(30,40)$, with the loss dropping from $\mathcal{O}(10^{-10})$ to $\mathcal{O}(10^{-15})$ accordingly. Meanwhile, Table 1 can be considered as an informal convergence analysis associated with the number of learnable parameters and training points used in the network. As seen, using a resolution of 512^2 uniform mesh, the IIM yields the accuracy E_p^∞ and $E_{\mathbf{u}}^\infty$ of the magnitude $\mathcal{O}(10^{-5})$ that is at least two orders of magnitude greater than the one obtained by the present network using relatively fewer training points $M = 1880$.

Example 4.2. In the second example (also referred to as Example 4.3 in [10]), we keep the domain Ω and interface Γ same as shown in Example 4.1. However, we choose the pressure jump no longer as a constant and it is defined by

$$p(\mathbf{x}) = \begin{cases} \left(\dfrac{3}{8} - \dfrac{3}{4}x_1^2 \right) x_1 x_2 & \text{in} \quad \Omega^-, \\ 0 & \text{in} \quad \Omega^+. \end{cases}$$

The exact solution of velocity $\mathbf{u}(\mathbf{x}) = (u_1(\mathbf{x}), u_2(\mathbf{x}))$ is chosen as

$$u_1(\mathbf{x}) = \begin{cases} \dfrac{1}{4}x_2 & \text{in} \quad \Omega^-, \\ \dfrac{1}{4}x_2(x_1^2 + x_2^2) & \text{in} \quad \Omega^+, \end{cases} \qquad u_2(\mathbf{x}) = \begin{cases} -\dfrac{1}{4}x_1(1 - x_1^2) & \text{in} \quad \Omega^-, \\ -\dfrac{1}{4}x_1 x_2^2 & \text{in} \quad \Omega^+, \end{cases}$$

and the external force $\mathbf{g}(\mathbf{x}) = (g_1(\mathbf{x}), g_2(\mathbf{x}))$ is derived as

$$g_1(\mathbf{x}) = \begin{cases} \left(\dfrac{3}{8} - \dfrac{9}{4}x_1^2\right)x_2 & \text{in} \quad \Omega^-, \\ -2\mu^+ x_2 & \text{in} \quad \Omega^+, \end{cases} \qquad g_2(\mathbf{x}) = \begin{cases} \left(\dfrac{3}{8} - \dfrac{3}{2}\mu^- - \dfrac{3}{4}x_1^2\right)x_1 & \text{in} \quad \Omega^-, \\ \dfrac{1}{2}\mu^+ x_1 & \text{in} \quad \Omega^+, \end{cases}$$

The interfacial force, $\mathbf{F}(\mathbf{x}_\Gamma) = (F_1(\mathbf{x}_\Gamma), F_2(\mathbf{x}_\Gamma))$, can be derived from the jump condition (2.4) as

$$F_1(\mathbf{x}_\Gamma) = \left(\frac{3}{4}x_1^2 - \frac{3}{8}\right)x_1^2 x_2 - \frac{3}{2}[\![\mu]\!]x_1^4 x_2 + \frac{1}{2}\mu^+ x_2 + \frac{3}{4}[\![\mu]\!]x_1^2(1 - 2x_1^2)x_2,$$

$$F_2(\mathbf{x}_\Gamma) = \left(\frac{3}{4}x_2^2 - \frac{3}{8}\right)x_1 x_2^2 - \frac{3}{2}[\![\mu]\!]x_1^3 x_2^2 - \frac{1}{2}\mu^+ x_1 - \frac{3}{4}[\![\mu]\!]x_1^3(1 - 2x_1^2).$$

For the network structure, we choose the exactly same setup as in Example 4.1. That is, we use the same values of N_p and M_0 but vary the viscosity contrast as $(\mu^-, \mu^+) = (1, 0.1), (10^{-3}, 1), (1, 10^{-3})$. Table 2 shows the L^∞ norm errors of $P_\mathcal{N}$ and $\mathbf{U}_\mathcal{N}$, and the corresponding training losses in the left panel. The corresponding results from IIM [10] are shown in the right panel for comparison. In the first case $(\mu^-, \mu^+) = (1, 0.1)$, by varying (N_p, M_0) from $(10, 20)$ to $(30, 40)$, we again observe that the present network predictions $P_\mathcal{N}$ and $\mathbf{U}_\mathcal{N}$ can achieve the accuracy with L^∞ errors ranging from the order $\mathcal{O}(10^{-5})$ to $\mathcal{O}(10^{-7})$ and the loss drops from $\mathcal{O}(10^{-10})$ to $\mathcal{O}(10^{-14})$ accordingly. As we increase the neurons N_p and the training points M_0, we again see an informal evidence for the numerical convergence of the present method. The results in [10] have $E_p^\infty = 1.54 \times 10^{-4}$ and $E_\mathbf{u}^\infty = 6.49 \times 10^{-5}$ under the resolution using a 512×512 uniform mesh. Meanwhile, we also show the capability of the present method for interface problems with high viscosity contrast

Table 2: The L^∞ errors of $P_\mathcal{N}$ and $\mathbf{U}_\mathcal{N}$ for Example 4.2. Left panel: the present network result. Here, $N_{\boldsymbol{\theta}} = 17N_p$, $M = M_0(M_0 + 7)$, $\epsilon_\theta = 10^{-14}$, and $Epoch_{\max} = 3000$. Right panel: IIM results in [10] with grid resolutions 128^2, 256^2 and 512^2.

		Present			IIM [10]		
(μ^-, μ^+)	(N_p, M_0)	E_p^∞	$E_\mathbf{u}^\infty$	Loss($\boldsymbol{\theta}$)	N_{IIM}	E_p^∞	$E_\mathbf{u}^\infty$
	(10,20)	4.43×10^{-5}	7.43×10^{-6}	2.83×10^{-10}	128^2	2.30×10^{-3}	1.21×10^{-3}
(1,0.1)	(20,30)	3.14×10^{-6}	5.49×10^{-7}	1.34×10^{-12}	256^2	5.47×10^{-4}	2.69×10^{-4}
	(30,40)	1.08×10^{-6}	1.21×10^{-7}	5.09×10^{-14}	512^2	1.54×10^{-4}	6.49×10^{-5}
	(10,20)	5.64×10^{-4}	8.74×10^{-5}	3.00×10^{-9}	128^2	1.04×10^{-3}	6.23×10^{-2}
$(10^{-3},1)$	(20,30)	5.84×10^{-5}	2.59×10^{-6}	6.84×10^{-12}	256^2	3.59×10^{-4}	1.40×10^{-2}
	(30,40)	2.65×10^{-6}	2.23×10^{-7}	4.62×10^{-14}	512^2	7.09×10^{-5}	2.82×10^{-3}
	(10,20)	6.62×10^{-4}	1.15×10^{-4}	2.42×10^{-8}	128^2	6.53×10^{-3}	3.15×10^{-1}
$(1,10^{-3})$	(20,30)	5.78×10^{-5}	6.28×10^{-6}	1.98×10^{-10}	256^2	1.18×10^{-3}	4.64×10^{-2}
	(30,40)	1.10×10^{-6}	1.44×10^{-7}	6.72×10^{-14}	512^2	3.02×10^{-4}	1.17×10^{-3}

as $(\mu^-,\mu^+)=(10^{-3},1)$ and $(1,10^{-3})$. The present method achieves accuracy with L^∞ errors ranging from $E_p^\infty=\mathcal{O}(10^{-4})$ to $E_\mathbf{u}^\infty=\mathcal{O}(10^{-7})$ which outperforms the results obtained from IIM significantly.

Example 4.3. We now consider solving a three-dimensional Stokes interface problem with a piecewise-constant viscosity ($\mu^-=1$ and $\mu^+=0.1$) and singular force in a domain $\Omega=[-2,2]^3$. The interface Γ is a simple unit sphere so it can be described by the zero level set of the function $\phi(\mathbf{x})=\|\mathbf{x}\|_2^2-1$. This example is taken directly from Example 6.2 in [24] so that we can compare the results with the augmented immersed interface method developed there. We list the exact solutions of pressure $p(\mathbf{x})$ and velocity $\mathbf{u}(\mathbf{x})$ as follows:

$$p(\mathbf{x})=\left\{\begin{array}{ll} 1, & \Omega^-, \\ 0, & \Omega^+, \end{array}\right. \quad \mathbf{u}(\mathbf{x})=\left(\begin{array}{c} u_1(\mathbf{x}) \\ u_2(\mathbf{x}) \\ u_3(\mathbf{x}) \end{array}\right)=\left(\begin{array}{c} x_2x_3(\|\mathbf{x}\|_2^2-1) \\ x_3x_1(\|\mathbf{x}\|_2^2-1) \\ -2x_1x_2(\|\mathbf{x}\|_2^2-1) \end{array}\right).$$

Here, the velocity components $u_1(\mathbf{x})$, $u_2(\mathbf{x})$, and $u_3(\mathbf{x})$ are all smooth functions over the entire Ω, and $p(\mathbf{x})$ is a piecewise constant. Accordingly, the external source $\mathbf{g}(\mathbf{x})=(g_1(\mathbf{x}),g_2(\mathbf{x}),g_3(\mathbf{x}))$ can be derived as

$$g_1(\mathbf{x})=\left\{\begin{array}{ll} -14\mu^-x_2x_3, & \Omega^-, \\ -14\mu^+x_2x_3, & \Omega^+, \end{array}\right.$$

$$g_2(\mathbf{x})=\left\{\begin{array}{ll} -14\mu^-x_3x_1, & \Omega^-, \\ -14\mu^+x_3x_1, & \Omega^+, \end{array}\right.$$

$$g_3(\mathbf{x})=\left\{\begin{array}{ll} 28\mu^-x_1x_2, & \Omega^-, \\ 28\mu^+x_1x_2, & \Omega^+. \end{array}\right.$$

Again, the interfacial force can be derived from the jump condition (2.4) as

$$\mathbf{F}(\mathbf{x})=\left(\begin{array}{c} F_1(\mathbf{x}) \\ F_2(\mathbf{x}) \\ F_3(\mathbf{x}) \end{array}\right)=\left(\begin{array}{c} -x_1-2[\![\mu]\!]x_2x_3 \\ -x_2-2[\![\mu]\!]x_3x_1 \\ -x_3+4[\![\mu]\!]x_1x_2 \end{array}\right).$$

Unlike the two-dimensional cases in previous examples, we now fix the number of training points but vary the number of neurons N_p and N_u to ease our computations. For the sampling of training data points, we generate M_I data points in the region $\Omega^-\cup\Omega^+$, M_B on the domain boundary ($M_B/6$ on each face of $\partial\Omega$), and M_Γ data points on the surface Γ generated using DistMesh [17]. The total number of training points used in each of the following numerical runs is $M=6216$ ($M_I=3000$, $M_B=2400$, and $M_\Gamma=816$). In the left panel of Table 3, we present the L^∞ errors of $P_\mathcal{N}$ and

Table 3: The L^∞ errors of $P_\mathcal{N}$ and $U_\mathcal{N}$ for Example 4.3. Left panel: the present network results. Here, $\mu^- = 1$ and $\mu^+ = 0.1$, $M = 6216$ training data points. Right panel: IIM results in [24] with grid resolutions 64^3, 128^3, and 256^3.

	Present				IIM [24]	
(N_p, N_u)	E_p^∞	$E_\mathbf{u}^\infty$	Loss($\boldsymbol{\theta}$)	Grid no.	E_p^∞	$E_\mathbf{u}^\infty$
$(25, 60)$	5.04×10^{-3}	2.61×10^{-3}	8.73×10^{-7}	64^3	3.06×10^{-3}	2.54×10^{-3}
$(30, 75)$	5.02×10^{-4}	1.91×10^{-4}	6.80×10^{-9}	128^3	6.44×10^{-4}	5.40×10^{-4}
$(40, 100)$	5.06×10^{-5}	3.92×10^{-5}	7.01×10^{-11}	256^3	1.68×10^{-4}	1.22×10^{-4}

$U_\mathcal{N}$ obtained using the proposed method under three different (N_p, N_u) pairs. Note that the total number of parameters can be computed as $N_{\boldsymbol{\theta}} = 6N_p + 8N_u$. As the network size increases, the proposed method achieves accurate predictions with L^∞ errors ranging from the order of $\mathcal{O}(10^{-3})$ to $\mathcal{O}(10^{-5})$. The right panel of Table 3 lists the results generated from the augmented IIM method [24]. One can see that, the present method can achieve comparably accurate results with the augmented IIM even far less the grid resolutions used.

Example 4.4. As the fourth example (also referred to as Example 6.3 in [24]), we keep the domain Ω and interface Γ the same as shown in Example 4.3. The piecewise constant viscosity is also chosen as $\mu^- = 1$ and $\mu^+ = 0.1$. However, we choose the pressure jump no longer as a constant and the exact solutions of the pressure $p(\mathbf{x})$ and velocity $\mathbf{u}(\mathbf{x}) = (u_1(\mathbf{x}), u_2(\mathbf{x}), u_3(\mathbf{x}))$ are given as follows.

$$p(\mathbf{x}) = \begin{cases} \left(\dfrac{3}{8} - \dfrac{3}{4}x_1^2\right)x_1 x_2 x_3, & \Omega^-, \\ 0, & \Omega^+, \end{cases} \qquad u_1(\mathbf{x}) = \begin{cases} \dfrac{1}{4}x_2 x_3, & \Omega^-, \\ \dfrac{1}{4}x_2 x_3(x_1^2 + x_2^2 + x_3^2), & \Omega^+, \end{cases}$$

$$u_2(\mathbf{x}) = \begin{cases} \dfrac{1}{4}x_3 x_1, & \Omega^-, \\ \dfrac{1}{4}x_3 x_1(x_1^2 + x_2^2 + x_3^2), & \Omega^+, \end{cases} \qquad u_3(\mathbf{x}) = \begin{cases} -\dfrac{1}{2}x_1 x_2(1 - x_1^2 - x_2^2), & \Omega^-, \\ -\dfrac{1}{2}x_1 x_2 x_3^2, & \Omega^+, \end{cases}$$

and the external force $\mathbf{g}$ is given

$$g_1(\mathbf{x}) = \begin{cases} \left(\dfrac{3}{8} - \dfrac{9}{4}x_1^2\right)x_2 x_3, & \Omega^-, \\ -\dfrac{7}{2}\mu^+ x_2 x_3, & \Omega^+, \end{cases}$$

$$g_2(\mathbf{x}) = \begin{cases} \left(\dfrac{3}{8} - \dfrac{3}{4}x_1^2\right)x_3 x_1, & \Omega^-, \\ -\dfrac{7}{2}\mu^+ x_3 x_1, & \Omega^+, \end{cases}$$

$$g_3(\mathbf{x}) = \begin{cases} \left(\dfrac{3}{8} - \dfrac{3}{4}x_1^2 - 6\mu^-\right)x_1 x_2, & \Omega^-, \\ \mu^+ x_1 x_2, & \Omega^+. \end{cases}$$

Therefore, the interfacial force has the form $\mathbf{F} = (F_1, F_2, F_3)$ as

$$F_1 = -\left(\frac{3}{8} - \frac{3}{4}x_1^2\right)x_1^2 x_2 x_3 - \frac{3}{4}[\![\mu]\!]x_2 x_3 - \mu^+ x_1^2 x_2 x_3 - \frac{1}{2}x_2 x_3\left(\mu^+\left(1 - x_3^2\right) + \mu^-\left(x_3^2 - 2x_1^2\right)\right),$$

$$F_2 = -\left(\frac{3}{8} - \frac{3}{4}x_1^2\right)x_1 x_2^2 x_3 - \frac{3}{4}[\![\mu]\!]x_1 x_3 - \mu^+ x_1 x_2^2 x_3 - \frac{1}{2}x_1 x_3\left(\mu^+\left(1 - x_3^2\right) + \mu^-\left(x_3^2 - 2x_2^2\right)\right),$$

$$F_3 = -\left(\frac{3}{8} - \frac{3}{4}x_1^2\right)x_1 x_2 x_3^2 - \frac{1}{2}[\![\mu]\!]x_1 x_2 + 2\mu^+ x_1 x_2 x_3^2 - \mu^- x_1 x_2\left(x_3^2 - x_1^2 - x_2^2\right).$$

Since the domain and the interface are exactly the same as in Example 4.3, we use the same number of training points M. However, we use slightly different neuron pairs $(N_p, N_u) = (30, 75), (40, 100), (40, 120)$ to generate the solution predictions. Table 4 shows the results for different (N_p, N_u) values. One can see that using far less learnable parameters (maximal $N_{\boldsymbol{\theta}} = 1200$ parameters to be learned) and training points ($M = 6216$), we are able to produce accurate results comparable to the augmented IIM proposed in [24].

Table 4: The L^∞ errors of $P_\mathcal{N}$ and $\mathbf{U}_\mathcal{N}$ for Example 4.4. Left panel: the present network results. Here, $\mu^- = 1$ and $\mu^+ = 0.1$, $M = 6216$ training data points. Right panel: IIM results in [24] with grid resolutions 64^3, 128^3, and 256^3.

	Present			IIM [24]		
(N_p, N_u)	E_p^∞	$E_{\mathbf{u}}^\infty$	Loss($\boldsymbol{\theta}$)	Grid no.	E_p^∞	$E_{\mathbf{u}}^\infty$
$(30, 75)$	4.44×10^{-4}	1.24×10^{-4}	9.26×10^{-9}	64^3	6.93×10^{-4}	4.07×10^{-4}
$(40, 100)$	2.32×10^{-4}	4.63×10^{-5}	4.61×10^{-10}	128^3	1.88×10^{-4}	8.67×10^{-5}
$(40, 120)$	4.14×10^{-5}	1.56×10^{-5}	1.42×10^{-10}	256^3	3.72×10^{-5}	2.06×10^{-5}

Example 4.5. Unlike the previous examples with analytic solutions in regular domains, here, we consider the Stokes interface problem with a piecewise-constant viscosity ($\mu^- = 10, \mu^+ = 1$) in the two-dimensional irregular circular domain $\Omega = \{\mathbf{x} \in \mathbb{R}^2 \,|\, \|\mathbf{x}\|_2 \leq 2\}$ whose solution is not available. The embedded interface Γ is parametrized as the polar function $\mathbf{x}_\Gamma(\alpha) = r(\alpha)(\cos\alpha, \sin\alpha)$, where $r(\alpha) = 1 - 0.2\cos(3\alpha)$ with $\alpha \in [0, 2\pi)$, representing a flower shape with three petals. We use $\phi(\mathbf{x}) = x_1^2 + x_2^2 - r(\alpha_{\mathbf{x}})$ as the level set function, where $\alpha_{\mathbf{x}} = \tan^{-1}\left(\frac{x_2}{x_1}\right)$ so the augmented feature inputs, $I(\mathbf{x})$ and $\phi_a(\mathbf{x})$, can be obtained as before accordingly. Along the interface, we now define the surface tension force $\mathbf{F}(\alpha) = -\gamma\kappa(\alpha)\mathbf{n}(\alpha)$ to mimic the stationary two-phase flow at some time instant. Here, γ is the surface tension and κ
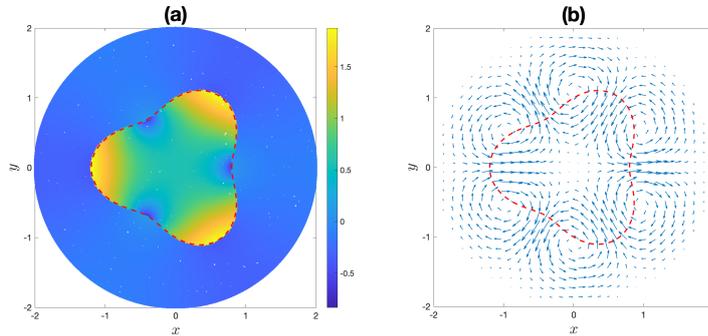
Figure 2: (a) Pressure prediction $P_{\mathcal{N}}$; (b) Velocity quiver plot of $\mathbf{U}_{\mathcal{N}}$.

is the signed curvature of the curve. For simplicity, we set the surface tension $\gamma = 1$, the external force $\mathbf{g} = \mathbf{0}$, and the boundary condition $\mathbf{u}_b = \mathbf{0}$.

Since the interface is more complex and the solution is more complicated, we hereby employ a four-hidden-layer deep neural network with $(N_p, N_u) = (15, 20)$ in each hidden layer, resulting in a total number of $N_{\boldsymbol{\theta}} = 2175$ trainable parameters used. (A comparison with a shallow network will be provided later.) The overall number of training points is chosen as $M = 7816$, consisting of $M_I = 4816$ points in $\Omega^- \cup \Omega^+$, $M_\Gamma = 1000$ points on the interface Γ, and $M_B = 2000$ points on the boundary $\partial\Omega$. We finish the training by either the loss falling below a threshold $\epsilon_{\theta} = 10^{-8}$ or reaching the maximum $Epoch_{\max} = 5000$ epochs. Since the analytic solution is unavailable, we investigate the flow behavior by plotting the pressure value and velocity quiver over the whole domain Ω in Fig. 2.

Fig. 2(a) shows the pressure prediction $P_{\mathcal{N}}$ in Ω with the interface labeled by the red dashed line. Since the surface tension force acts in the normal direction along the interface, one can clearly see the pressure jump across the interface. The pressure jump is significantly larger, especially near the tip of each petal, where the curvature is relatively large. Meanwhile, near the tip part of each petal (positive curvature region), the pressure inside is higher than the outside one. On the contrary, the pressure inside is lower than the outside one near the neck part (negative curvature region). Fig. 2(b) displays the corresponding velocity quiver $\mathbf{U}_{\mathcal{N}}$ in the domain Ω. It is observed that the flow tends to reduce the absolute magnitudes of curvature along the interface and relaxes to a circular shape. This instant flow tendency matches well with the simulation when the interface dynamics is taken into account. Meanwhile, since the fluid is incompressible, we can see three vortex dipoles (or six counter-rotating vortices) appearing at the petals.

We conclude this example by showing that the deeper neural network used here
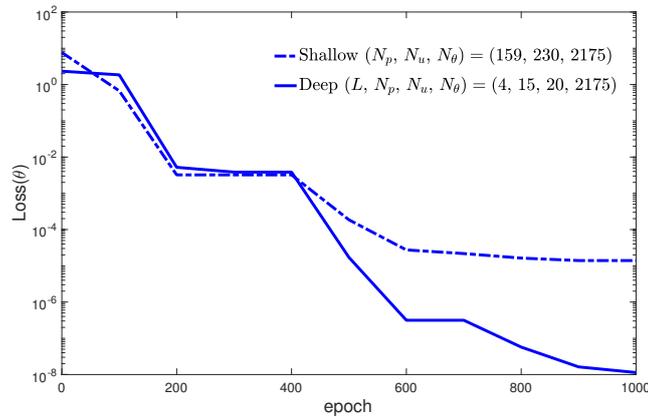
Figure 3: Evolution of training losses using shallow (dash-dotted line) and deep (solid line) networks with the same numbers of trainable parameters ($N_\theta = 2175$) and training points ($M = 7816$).

indeed performs better than a shallow one. To demonstrate that, we use a shallow network structure with neurons $(N_p, N_u) = (159, 230)$ so the total number of trainable parameters $N_\theta = 2175$ is the same as the one used in present four-hidden-layer network with $(N_p, N_u) = (15, 20)$ in each hidden layer. Also we fix the training points $M = 7816$ and train both networks up to 1000 epochs. The training time for both networks are almost the same since we use the same numbers of trainable parameters and training points. Fig. 3 shows both evolution of training losses. One can see that, using the four-hidden-layer network, the training loss can drop down to the order of 10^{-8} within 1000 epochs, while the training loss of the shallow one only drops to the order of 10^{-5}. This illustration shows an appropriate network architecture can indeed impact the efficiency of the training process.

5   Conclusions

Solving Stokes equations with a piecewise-constant viscosity and singular force on an interface using traditional grid-based methods encounter the major challenge of computing the non-smooth solutions numerically. One can reformulate the governing equations by solving the Stokes equations in two sub-domains separately; however, the pressure and velocity from both sub-domains are coupled together by the traction balance equation on the interface. This makes the grid-based methods hard to discretize accurately near the interface and the resultant linear system hard to solve. In this paper, we provide a neural network approach to solve the equations. Our approach is based on adopting a discontinuity capturing sub-network to approximate the pressure and a cusp-capturing sub-network to approximate the

velocity so that the network predictions can retain the inherent properties of the solutions. Since the present neural network method is completely mesh-free, the implementation is much easier than the traditional grid-based methods, such as immersed interface method, which needs careful and laborious efforts to discretize the equations accurately near the interface. A series of numerical experiments are performed to test the accuracy against the results obtained from the immersed interface methods in the existing literature. A completely shallow (one-hidden-layer) network with a moderate number of neurons and sufficient training data points can obtain comparably accurate results obtained from the traditional methods. We shall apply the present network to solve realistic two-phase flow applications in the future.

Acknowledgements

References

[1] R. Egan and F. Gibou, xGFM: Recovering convergence of fluxes in the ghost fluid method, J. Comput. Phys., 409 (2020), 109351.

[2] H. Guo and X. Yang, Deep unfitted Nitsche method for elliptic interface problems, Commun. Comput. Phys., 31 (2022), 1162–1179.

[3] A. Griewank and A. Walther, Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, 2nd ed., SIAM, 2008.

[4] C. Hu, 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.

[5] W.-F. Hu, M.-C. Lai, and Y.-N. Young, A hybrid immersed boundary and immersed interface method for electrohydrodynamic simulations, J. Comput. Phys., 282 (2015), 47–61.

[6] 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.

[7] D. D. Joseph, and Y. Y. Renardy, Fundamentals of Two-fluid Dynamics, Springer-Verlag, 1993.

[8] M.-C. Lai and H.-C. Tseng, A simple implementation of the immersed interface methods for stokes flows with singular forces, Comput. Fluids, 37 (2008), 99–106.

[9] R. J. LeVeque, and Z. Li, The immersed interface method for elliptic equations with discontinuous coefficients and singular sources, SIAM J. Numer. Anal., 31(4) (1994), 1019–1044.

[10] Z. Li, K. Ito, and M.-C. Lai, An augmented approach for Stokes equations with a discontinuous viscosity and singular forces, Comput. Fluid, 36 (2007), 622–635.

[11] X.-D. Liu, R. P. Fedkiw, and M. Kang, A boundary condition capturing method for poisson's equation on irregular domains, J. Comput. Phys., 160 (2000), 151–178.

[12] Y. Liu, and Y. Mori, Properties of discrete delta functions and local convergence of the immersed boundary method, SIAM J. Numer. Anal., 50 (2012), 2986–3015.

[13] J. J, Moré, The Levenberg-Marquardt Algorithm: Implementation and Theory, Numerical analysis, Springer, Berlin, Heidelberg, (1978), 105–116.

[14] 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, 21 (1979) 239–245.

[15] Ö. 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.

[16] C. S. Peskin, The immersed boundary method, Acta Numer., 11 (2002), 479–517.

[17] P. O. Persson, and G. Strang, A simple mesh generator in MATLAB, SIAM Rev. Soc. Ind. Appl. Math., 46 (2004), 329–345.

[18] 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, and 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., 32, Curran Associates, Inc. (2019), 8024–8035.

[19] C. Pozrikidis, Introduction to Theoretical and Computational Fluid Dynamics, Oxford, 1997.

[20] 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), 686–707.

[21] Y.-H. Tseng, T.-S. Lin, W.-F. Hu, and M.-C. Lai, A cusp-capturing PINN for elliptic interface problems, J. Comput. Phys., in press (2023), arXiv: 2210.08424.

[22] M. K. Transtrum, and J. P. Sethna, Improvements to the Levenberg-Marquardt algorithm for nonlinear least-squares minimization, arXiv: 1201.5885 (2012).

[23] 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.

[24] W. Wang, and Z. Tan, A simple augmented IIM for 3D incompressible two-phase Stokes flows with interfaces and singular forces, Comput. Phys. Commun., 270 (2022), 108154.