Journal of Computational Physics 469 (2022) 111547

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

A shallow Ritz method for elliptic problems with singular sources

Ming-Chih Lai [a], Che-Chia Chang [a], Wei-Syuan Lin [a], Wei-Fan Hu [b,c], Te-Sheng Lin [a,c,*]

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

ARTICLE INFO

Article history:
Received 26 July 2021
Received in revised form 1 July 2022
Accepted 14 August 2022
Available online 22 August 2022

Keywords:
Shallow neural network
Deep Ritz method
Elliptic problems
Singular source
Immersed boundary method
Level set function

ABSTRACT

In this paper, a shallow Ritz-type neural network for solving elliptic equations with delta function singular sources on an interface is developed. There are three novel features in the present work; namely, (i) the delta function singularity is naturally removed, (ii) level set function is introduced as a feature input, (iii) it is completely shallow, comprising only one hidden layer. We first introduce the energy functional of the problem and then transform the contribution of singular sources to a regular surface integral along the interface. In such a way, the delta function singularity can be naturally removed without introducing a discrete one that is commonly used in traditional regularization methods, such as the well-known immersed boundary method. The original problem is then reformulated as a minimization problem. We propose a shallow Ritz-type neural network with one hidden layer to approximate the global minimizer of the energy functional. As a result, the network is trained by minimizing the loss function that is a discrete version of the energy. In addition, we include the level set function of the interface as a feature input of the network and find that it significantly improves the training efficiency and accuracy. We perform a series of numerical tests to show the accuracy of the present method and its capability for problems in irregular domains and higher dimensions.

1. Introduction

Fluid-structure interaction problems have many applications in science and engineering, one example of which is blood flow (fluid) simulation in heart valve leaflets (embedded structure) [28]. The numerical simulation challenge for such problems is mainly that the shapes of embedded structures are often irregular and such structures change with time. To address these issues, Peskin [28] proposed the so-called Immersed Boundary (IB) method, which is widely used because of its simplicity of implementation. This method does not require body-fitted discretization of the structure, which can save significant computational efforts.

The immersed boundary method is, in fact, both a mathematical formulation and numerical method for fluid-structure interaction problems. In IB formulation, we represent the fluid variables in an Eulerian manner and the embedded structure in Lagrangian one. The embedded structure is usually one-dimensional lower than the fluid dimensional space and regards

* Corresponding author.
E-mail addresses: mclai@math.nctu.edu.tw (M.-C. Lai), wfhu@math.ncu.edu.tw (W.-F. Hu), tslin@math.nctu.edu.tw (T.-S. Lin).

https://doi.org/10.1016/j.jcp.2022.111547
0021-9991/

as a singular force generator. Thus, the governing equations comprise Navier-Stokes (or Stokes) equations with singular forces as the Dirac delta function. To solve the problem numerically, a projection-type of Navier-Stokes solver (for instance, see an overview in [12]) often involves solving elliptic equations with singular sources for the intermediate velocity. The IB numerical method then solves the equations by using finite difference discretization and a smooth version of the discrete delta function to regularize the singular sources. This method is easy to implement but leads to first-order accuracy. Another example is solving heat equations with singular sources. If time discretization is applied (for instance implicit Euler method), then at each time step, an elliptic equation with singular sources must be solved. So motivated by the above applications, we aim to solve elliptic problems with singular sources on an interface in this paper.

As mentioned above, the IB method is first-order accurate for elliptic interface problems due to the discrete regularization of the delta function sources. There are several other grid-based methods that achieve better accuracy in literature. For instance, the immersed interface method (IIM) proposed by LeVeque and Li [22] incorporates the jump conditions via local coordinates into the finite difference scheme so that the local truncation error near the interface can be first-order resulting in the overall second-order accuracy in maximum norm. A simple implementation version of IIM that directly uses the jump conditions developed by the first author and his coworkers in [21,19] has the second-order accuracy in maximum norm as well. A boundary condition capturing method (also named as ghost fluid method (GFM)) proposed by Liu et al. [25] is able to solve the elliptic interface problems in a dimension-by-dimension manner, and captures the solution and its normal derivative jumps sharply while smoothing the tangential derivative. The method is first-order accurate in maximum norm in general. Recently, Egan and Gibou [9] have developed a novel idea to extend the original GFM by recovering the convergence of the gradient so that second-order accuracy can be achieved without modifying the resultant linear system. Along with this GFM approach, many other numerical methods for solving elliptic problems with interfaces or in irregular domains [11,13,8,2,10,3] have been successfully developed to improve the overall accuracy in maximum norm. One should mention that the linear systems resulting from those above methods are often symmetric positive definite, which can be solved efficiently by using iterative methods. Nevertheless, like IIM, special numerical treatments are always needed in the finite difference discretization of GFM near the interface or the irregular domain boundary. Thus, a mesh-free neural network method for solving the above problems provides an alternative to circumvent the difficulty arising from discretization near the interface or domain boundary for grid-based methods.

Solving partial differential equations (PDEs) with deep neural networks (DNNs) has drawn much attention in the scientific computing community recently. Part of the theoretical reason can be attributed to the various kinds of expressive power for function approximations using DNN such as those described in [6,17,27,16], just to name a few. In terms of implementation, there are mainly two different approaches; namely, the physics-informed neural networks [31], and the deep Ritz method [7]. The major difference between the two approaches is how the loss is defined. One trains the physics-informed neural networks by minimizing the mean squared error loss of the equation residual, along with the initial and boundary condition errors. The deep Ritz method, however, begins with formulating the variational problem equivalent to the original PDE, so the natural loss function in this framework is simply the energy. Both approaches share the same major mesh-free advantage and therefore can practically solve problems in complex geometry [32] and in high-dimensional space [15,33].

Regarding deep learning approaches to solve PDEs with solutions that are piecewise smooth, Wang et al. [34] proposed a deep Ritz-type approach to solve elliptic interface problems with high-contrast discontinuous coefficients, and Cai et al. [5] introduced the deep least squares method to solve elliptic interface problems where the solutions are continuous but the derivatives have jumps across the interface. In [14], a deep unfitted Nitsche method is developed where two deep neural networks are formulated to represent two components of the solution. In [4], the authors developed an immersed boundary neural network for solving elliptic equations with singular sources in which the delta function singularity still appears in the formulation and is discretized by a smooth discrete delta function proposed by Peskin [28].

Recently, the authors of the present paper proposed an efficient and accurate Discontinuity Capturing Shallow Neural Network (DCSNN) [18] to solve elliptic interface problems where the solution is only piecewise-continuous. By augmenting one coordinate variable, the proposed shallow neural network is trained with PINNs-type loss.

In this paper, we propose a new shallow Ritz method for solving elliptic problems with singular sources. The novelties of the proposed network are three-fold. First, we remove the delta function singularity appearing in the original PDE by formulating the variational problem. Second, we include the level set function, which is commonly used as an interface indicator, as an additional feature input of the network that effectively improves the model's efficiency and accuracy. Third, we approximate the solution using a shallow neural network with only one hidden layer that significantly reduces the training cost in contrast to DNN.

The rest of the paper is organized as follows. In Section 2, we show how to transform a d-dimensional elliptic equation with singular sources into an energy functional minimization problem. The shallow Ritz method to solve the problem is presented in Section 3, followed by a series of numerical accuracy tests and comparisons in Section 4. We give some concluding remarks and future work in Section 5.

2. Elliptic equations with singular sources on the interface

We consider a d-dimensional elliptic equation in a bounded domain $\Omega \subset \mathbb{R}^d$, divided by an embedded interface $\Gamma \subset \mathbb{R}^{d-1}$ into two regions; namely, inside (Ω^-) and outside (Ω^+) of the interface, so that $\Omega = \Omega^- \cup \Omega^+ \cup \Gamma$. The interface Γ is

represented by its parametric form $\mathbf{X}(\mathbf{s})$ with surface parametrization $\mathbf{s} \in \mathbb{R}^{d-1}$. The elliptic equation with singular sources on the interface is written as

$$\Delta u(\mathbf{x}) - \alpha u(\mathbf{x}) = f(\mathbf{x}) + \int_\Gamma c(\mathbf{s})\delta(\mathbf{x} - \mathbf{X}(\mathbf{s}))\, d\mathbf{s}, \quad \text{in } \Omega, \tag{1}$$

$$u(\mathbf{x}) = g(\mathbf{x}), \quad \text{on } \partial\Omega, \tag{2}$$

where α is a non-negative constant ($\alpha = 0$ corresponds to the Poisson equation), f is a given function, $c(\mathbf{s})$ is the source density defined only on the interface, and δ is the d-dimensional Dirac delta function. Throughout this paper, we only focus on the case of the Dirichlet boundary condition, but other boundary conditions can be easily applied without changing the major ingredients of the proposed method.

As mentioned before, Eq. (1) appears in many realistic applications and, in particular, the fluid velocity equations in the immersed boundary formulation for fluid-structure interaction problems [29]. The fundamental difficulty in numerically solving Eq. (1) is that the second term of the right-hand side has the integral over the $d-1$ dimensional interface, while the delta function is d-dimensional, this leaves the term has one-dimensional delta function singularity. The IB method uses finite difference discretization and a grid-based discrete delta function to approximate the integral term in Eq. (1). Because of the singularity of the delta function, this regularization technique is only first-order accurate, no matter what discrete delta functions are used. It was rigorously proved by Li [23] that the IB method for Eqs. (1)-(2) in 2D is indeed first-order accurate.

It is known that the solution to Eq. (1) is piecewise smooth across the interface; more precisely, the solution is continuous over the domain Ω but has a discontinuity in its normal derivative on the interface Γ. In fact, Eq. (1) is equivalent to the following elliptic interface problem with the same boundary condition (2) as

$$\Delta u(\mathbf{x}) - \alpha u(\mathbf{x}) = f(\mathbf{x}) \quad \text{in } \Omega \setminus \Gamma, \quad [\![u]\!](\mathbf{s}) = 0, \quad [\![\partial_n u]\!](\mathbf{s}) = c(\mathbf{s}) \quad \text{on } \Gamma, \tag{3}$$

where the notation $[\![\cdot]\!]$ represents the difference of the quantity (from the outside value to the inside value) across the interface. The above derivation can be found for example in [24]. One can immediately see that the delta function singularity no longer exists in Eq. (3), but in terms of normal derivative jump instead. It is also worth mentioning that using equation (3), one can easily construct different analytic solutions for the purpose of numerical tests later.

As mentioned in the Introduction, we have developed a neural network solver called DCSNN [18] to approximate piecewise continuous functions and to solve more general elliptic interface problems than the one in Eq. (3). The DCSNN augments an additional coordinate variable to label the pieces of a function so it inherently represents a discontinuous function. However, as you can see from Eq. (3), the solution is continuous across the interface. So in this paper, we propose a shallow Ritz-type neural network to solve Eqs. (1)-(2) by augmenting a level set function (continuous) as a feature input so that the solution is continuous and the energy is the natural loss function of the neural network. To proceed, we first reformulate Eq. (1) into a variational problem as follows.

2.1. Variational problem

As shown above, the solution to the elliptic equation with singular sources on the interface, Eq. (1), is continuous but has discontinuous derivatives across the interface. Since the solution is not classically smooth, we are going to use the usual Sobolev spaces $H_0^1(\Omega)$ and $H^1(\Omega)$ to define the solution space. We assume the right-hand side function $f \in L^2(\Omega)$ and the source strength $c \in L^2(\Gamma)$. In order to take the boundary condition (2) into account, followed the work in [26], we assume that the boundary data g has a smooth extension $\tilde{g}$ to the bounded domain Ω so that $u - \tilde{g} \in H_0^1(\Omega)$ and $\tilde{g}|_{\partial\Omega} = g$. So the solution space can be defined as $H_g^1 = \{u \in H^1(\Omega) : u - \tilde{g} \in H_0^1(\Omega)\}$. We now formulate equation (1) into its weak form by simply multiplying the test function $v \in H_0^1(\Omega)$ in Eq. (1). Using the Green's first identity and the fact that test function v vanishes at the boundary $\partial\Omega$, we obtain the following weak formulation: find $u \in H_g^1$ such that

$$-\int_\Omega \nabla u(\mathbf{x}) \cdot \nabla v(\mathbf{x})\, d\mathbf{x} - \alpha \int_\Omega u(\mathbf{x}) v(\mathbf{x})\, d\mathbf{x} = \int_\Omega f(\mathbf{x}) v(\mathbf{x})\, d\mathbf{x} + \int_\Gamma c(\mathbf{s}) v(\mathbf{X}(\mathbf{s}))\, d\mathbf{s}, \tag{4}$$

for all $v \in H_0^1(\Omega)$. Note that the second term on the right-hand side comes from the definition of the Dirac delta function as

$$\begin{aligned} &\int_\Omega v(\mathbf{x}) \int_\Gamma c(\mathbf{s})\delta(\mathbf{x} - \mathbf{X}(\mathbf{s}))\, d\mathbf{s}\; d\mathbf{x} \\ &= \int_\Gamma c(\mathbf{s}) \int_\Omega v(\mathbf{x})\delta(\mathbf{x} - \mathbf{X}(\mathbf{s}))\, d\mathbf{x}\; d\mathbf{s} = \int_\Gamma c(\mathbf{s}) v(\mathbf{X}(\mathbf{s}))\, d\mathbf{s}. \end{aligned} \tag{5}$$

One can also derive the weak formulation by using the equivalent equation (3) with jump conditions in which the Green's first identity is applied separately in Ω^+ and Ω^-, and then summing up together to get Eq. (4). The existence and uniqueness of the above weak solution can be found in [26].

To adopt a Ritz-type neural network to solve the problem, we now rewrite the above weak formulation (4) to its equivalent minimization problem as follows. Find $u \in H_g^1$ such that $L[u] \le L[v]$ for all $v \in H_g^1$, where the energy functional reads

$$L[v] = \frac{1}{2}\int_\Omega |\nabla v(\mathbf{x})|^2 \,\mathrm{d}\mathbf{x} + \frac{\alpha}{2}\int_\Omega v(\mathbf{x})^2 \,\mathrm{d}\mathbf{x} + \int_\Omega v(\mathbf{x}) f(\mathbf{x}) \,\mathrm{d}\mathbf{x} + \int_\Gamma c(\mathbf{s}) v(\mathbf{X}(\mathbf{s})) \,\mathrm{d}\mathbf{s}. \tag{6}$$

As a result, just like the weak formulation, the delta function singularity disappears completely in the energy and its contribution becomes a regular integral over the interface Γ. Thus, we do not need to handle the singularity problem arising from the original equation (1).

2.2. Boundary condition enforcement

We aim to solve the problem by using neural networks as the solution representation. However, while neural networks are expressive in function approximation, and one hidden layer neural network can be a universal approximator (see a review in [30]), the boundary condition requirement in H_g^1 still seems to be infeasible. Thus, we simply relax the boundary condition (2) by adding a penalty term to the energy (6) that reflects the penalty effect if the boundary condition is not satisfied exactly. The energy functional is therefore modified as

$$\begin{aligned}\tilde{L}[v] = {} & \frac{1}{2}\int_\Omega |\nabla v(\mathbf{x})|^2 \,\mathrm{d}\mathbf{x} + \frac{\alpha}{2}\int_\Omega v(\mathbf{x})^2 \,\mathrm{d}\mathbf{x} + \int_\Omega v(\mathbf{x}) f(\mathbf{x}) \,\mathrm{d}\mathbf{x} \\ & + \int_\Gamma c(\mathbf{s}) v(\mathbf{X}(\mathbf{s})) \,\mathrm{d}\mathbf{s} + \beta \int_{\partial\Omega} (v(\mathbf{x}) - g(\mathbf{x}))^2 \,\mathrm{d}\mathbf{x},\end{aligned} \tag{7}$$

where β is some positive penalty constant.

Now, it is interesting to see what the global minimizer will be for such a modified energy functional. To proceed, let us decompose a function $v \in H^1(\Omega)$ as a sum of two functions by writing $v = u + h$ (the choice of u and h will be clear later). We have its energy

$$\begin{aligned}\tilde{L}[v] = {} & \frac{1}{2}\int_\Omega |\nabla u + \nabla h|^2 \,\mathrm{d}\mathbf{x} + \frac{\alpha}{2}\int_\Omega (u+h)^2 \,\mathrm{d}\mathbf{x} + \int_\Omega (u+h) f \,\mathrm{d}\mathbf{x} \\ & + \int_\Gamma c(u+h) \,\mathrm{d}\mathbf{s} + \beta \int_{\partial\Omega} (u + h - g)^2 \,\mathrm{d}\mathbf{x} \\ = {} & \tilde{L}[u] + \frac{1}{2}\int_\Omega |\nabla h|^2 \,\mathrm{d}\mathbf{x} + \frac{\alpha}{2}\int_\Omega h^2 \,\mathrm{d}\mathbf{x} + \beta \int_{\partial\Omega} h^2 \,\mathrm{d}\mathbf{x} \\ & + \int_\Omega (\nabla u \cdot \nabla h + \alpha u h + h f) \,\mathrm{d}\mathbf{x} + \int_\Gamma c h \,\mathrm{d}\mathbf{s} + \beta \int_{\partial\Omega} 2h(u-g) \,\mathrm{d}\mathbf{x},\end{aligned}$$

where the term $\int_\Omega \nabla u \cdot \nabla h \,\mathrm{d}\mathbf{x}$ can be rewritten, using the Green's first identity under the smoothness assumption of u in Ω^+ and Ω^- (i.e. $u \in C^2(\Omega^\pm)$), as

$$\begin{aligned}\int_\Omega \nabla u \cdot \nabla h \,\mathrm{d}\mathbf{x} &= \int_{\Omega^-} \nabla u \cdot \nabla h \,\mathrm{d}\mathbf{x} + \int_{\Omega^+} \nabla u \cdot \nabla h \,\mathrm{d}\mathbf{x} \\ &= -\int_{\Omega\setminus\Gamma} h \Delta u \,\mathrm{d}\mathbf{x} - \int_\Gamma h [\![\partial_n u]\!] \,\mathrm{d}\mathbf{s} + \int_{\partial\Omega} h \partial_n u \,\mathrm{d}\mathbf{x}.\end{aligned}$$

Therefore we obtain

$$\begin{aligned}\tilde{L}[v] = {} & \tilde{L}[u] + \frac{1}{2}\int_\Omega |\nabla h|^2 \,\mathrm{d}\mathbf{x} + \frac{\alpha}{2}\int_\Omega h^2 \,\mathrm{d}\mathbf{x} + \beta \int_{\partial\Omega} h^2 \,\mathrm{d}\mathbf{x} \\ & + \int_{\Omega\setminus\Gamma} h(-\Delta u + \alpha u + f) \,\mathrm{d}\mathbf{x} + \int_\Gamma \left(c - [\![\partial_n u]\!]\right) h \,\mathrm{d}\mathbf{s} + \int_{\partial\Omega} h\left(\partial_n u + 2\beta(u-g)\right) \,\mathrm{d}\mathbf{x}.\end{aligned}$$

That is, if we choose u as the solution of Eq. (3) but with a modified Robin-type boundary condition

$$u(\mathbf{x}) + \frac{1}{2\beta}\partial_n u(\mathbf{x}) = g(\mathbf{x}), \quad \mathbf{x} \in \partial\Omega, \tag{8}$$

we should have

$$\tilde{L}[v] = \tilde{L}[u] + \frac{1}{2}\int_\Omega |\nabla(v-u)|^2 \, d\mathbf{x} + \frac{\alpha}{2}\int_\Omega (v-u)^2 \, d\mathbf{x} + \beta \int_{\partial\Omega} (v-u)^2 \, d\mathbf{x}. \tag{9}$$

So it is clear that the solution of Eq. (3) (or equivalently Eq. (1)) with boundary condition (8) is the global minimizer of the energy functional $\tilde{L}$. We thus formally write the solution to the variational problem as

$$u = \arg\min_{v \in H^1(\Omega)} \tilde{L}[v], \tag{10}$$

where $H^1(\Omega)$ is the set of trial functions that does not require any constraint at the domain boundary. It is important to mention that as the penalty constant β becomes larger, the above minimizer should approach to the solution of Eq. (3) with exact boundary condition (2) (in fact, as $\beta \to \infty$, Eq. (8) tends to be Eq. (2)).

2.3. Level set function augmentation

As shown above, the minimizer of the energy functional (or the solution of Eq. (3)) is a continuous function, but has a jump discontinuity at its normal derivative on the interface. Therefore, the trial functions must carry the same feature. We note that the construction of such a set of functions using neural net approximation requires additional efforts. Here, inspired by DCSNN [18], where an augmented variable is introduced to categorize precisely the spatial coordinates into each sub-domain, we also introduce an augmented variable and require the function to be continuous throughout the whole domain. More precisely, consider a level set function $\phi(\mathbf{x})$ such that the zero level set gives the position of the interface Γ, i.e., $\Gamma = \{\mathbf{x} \in \mathbb{R}^d \,|\, \phi(\mathbf{x}) = 0\}$. We define a function $U(\mathbf{x}, z) : \mathbb{R}^{d+1} \to \mathbb{R}$ that satisfies

$$u(\mathbf{x}) = U(\mathbf{x}, \phi(\mathbf{x})), \quad \mathbf{x} \in \Omega. \tag{11}$$

Here the level set function $\phi(\mathbf{x})$ is considered as a feature input in the augmented variable z. We require both the level set function ϕ and the extension function U being continuous, so that $u(\mathbf{x})$ is a continuous function. Although it looks like the derivative discontinuity on the interface is not considered, the augmented variable $\phi(\mathbf{x})$ somehow gives additional information to the function U related to the interface. As we will see later in the numerical experiments, indeed the introduction of the augmented variable effectively improves the capability of neural networks in function approximation.

It is worth mentioning that we assume the solution to the problem takes the form: $u(\mathbf{x}) = U(\mathbf{x}, \phi(\mathbf{x}))$. Therefore, if one instead uses an indicator function for the interface as the additional feature input, e.g., $\phi(\mathbf{x}) = 1$ if $\mathbf{x} \in \Omega^+$ and $\phi(\mathbf{x}) = -1$ if $\mathbf{x} \in \Omega^-$, the resulting function will be discontinuous, which essentially violates the assumption of a continuous solution that we consider here.

2.4. Summary

To summarize, we solve elliptic equations with singular sources on the interface by looking for a $d+1$ dimensional continuous function of the form $U(\mathbf{x}, z)$. The target function is found by minimizing the energy functional $\tilde{L}[u]$, where $u(\mathbf{x}) = U(\mathbf{x}, \phi(\mathbf{x}))$. In the following we shall develop a neural network architecture to represent U and a loss function to be used for model training.

3. A shallow Ritz method

We propose a shallow neural network to approximate the $(d+1)$-dimensional continuous function of the form $U(\mathbf{x}, \phi(\mathbf{x}))$ and serve as an ansatz for solving the minimization problem Eq. (10). Based on the universal approximation theory [6], we hereby design a *shallow*, *feedforward*, *fully-connected* neural network architecture, in which only one hidden layer is employed. Let N be the number of neurons used in the hidden layer, the approximation function (or output layer) under this network structure is explicitly expressed by

$$u(\mathbf{x}) = U(\mathbf{x}, \phi(\mathbf{x})) = W^{[2]}\sigma(W^{[1]}(\mathbf{x}, \phi(\mathbf{x}))^T + b^{[1]}) + b^{[2]}, \tag{12}$$

where $W^{[1]} \in \mathbb{R}^{N\times(d+1)}$ and $W^{[2]} \in \mathbb{R}^{1\times N}$ are the weight matrices, $b^{[1]} \in \mathbb{R}^N$ and $b^{[2]} \in \mathbb{R}$ are the bias vectors, and σ is the activation function. One can easily see that the function U is a linear combination of the activation functions and hence shares exactly the same smoothness as σ. By collecting all the training parameters (including all the weights and biases) in a vector $\mathbf{p}$, the total number of parameters in the network (i.e., dimension of $\mathbf{p}$) is counted by $N_p = (d+3)N + 1$.

As the solution we are looking for is the global minimizer of the energy functional, it is therefore natural to consider the loss function in the training procedure using that energy. In this stage, we aim to learn the training parameters $\mathbf{p}$ via minimizing the modified energy (7). The loss function in the training procedure is thus defined in the framework of deep Ritz method [7] by replacing the integrals in Eq. (7) using discrete quadrature rules. That is, given training points in Ω, along the embedding interface Γ, and on the domain boundary $\partial\Omega$, denoted by $\{\mathbf{x}^i\}_{i=1}^{M}$, $\{\mathbf{x}_\Gamma^j\}_{j=1}^{M_\Gamma}$ and $\{\mathbf{x}_{\partial\Omega}^k\}_{k=1}^{M_b}$, respectively, we define the loss function as

$$\begin{aligned}\text{Loss}(\mathbf{p}) = &\frac{\text{Vol}(\Omega)}{M}\sum_{i=1}^{M}\left(\frac{1}{2}|\nabla u(\mathbf{x}^i)|^2 + \frac{\alpha}{2}u(\mathbf{x}^i)^2 + u(\mathbf{x}^i)f(\mathbf{x}^i)\right)\\ &+ \frac{\text{Vol}(\Gamma)}{M_\Gamma}\sum_{j=1}^{M_\Gamma} c(\mathbf{x}_\Gamma^j)u(\mathbf{x}_\Gamma^j) + \beta\,\frac{\text{Vol}(\partial\Omega)}{M_b}\sum_{k=1}^{M_b}\left(u(\mathbf{x}_{\partial\Omega}^k) - g(\mathbf{x}_{\partial\Omega}^k)\right)^2,\end{aligned} \tag{13}$$

where $\text{Vol}(\Omega)$ is the volume of Ω in $\mathbb{R}^d$, while $\text{Vol}(\Gamma)$ and $\text{Vol}(\partial\Omega)$ are volumes of Γ and $\partial\Omega$, respectively, in $\mathbb{R}^{d-1}$. Here, for brevity, we suppress the notation of the parameters $\mathbf{p}$ in the solution u as one can see their dependence through the equation (12). Also note that, the evaluation of u at the domain training point $\mathbf{x}^i$ is realized through the relation $u(\mathbf{x}^i) = U(\mathbf{x}^i, \phi(\mathbf{x}^i))$ so the feature input of the network is $(\mathbf{x}^i, \phi(\mathbf{x}^i))$. Similarly, the evaluations of u at the training points $\mathbf{x}_\Gamma^j$ in Γ and $\mathbf{x}_{\partial\Omega}^k$ in $\partial\Omega$ are defined in the same manner.

3.1. Selection of training points

As one can see, the quadrature rule used to derive the loss function (13) is of the Monte Carlo type, where the integrals in modified energy Eq. (7) are approximated by the mean of the samples. However, since the energy is presented as integrals, one may expect a better performance by evaluating these integrals using more accurate numerical quadrature rules. Thus, if the problem under consideration is defined in regular domains such as a rectangle or a circle, the first intuitive way to select the training points might be the quadrature nodes based on, e.g., midpoint or Gaussian quadrature rules, just to name a few. (Note, if Gaussian nodes are chosen as the training points, one should change the weights in approximating the energy functional.) But, we found that the training of neural networks heavily depends on "accurate" evaluations of these integrals. The question is not which quadrature rule is chosen, but how to ensure accurate evaluation of the integral value.

In traditional scientific computing approach, a convergence test must be performed if we want to evaluate an integral to the desired accuracy but without a prior information about the number of quadrature nodes required. Therefore, if we fix the number of nodes to compute the integral, the result might not be reliable. In neural network approach, the parameters change at each iteration of the training process meaning that the function presented by the network differs from iteration to iteration. For the Ritz method, if the training points are fixed throughout the entire optimization process to minimize the energy functional, we can not guarantee the numerical integration accuracy.

To ensure a reliable estimation of the energy, ideally, we should check the convergence of the numerical quadrature at each training step which is not practical in reality. In this paper, we adopt the strategy proposed in [7]. That is, we use Monte Carlo integration to evaluate the loss function Eq. (13) but re-select training points at each iteration of the optimization process. The spirit of this idea is like the mini-batch gradient descent method. Imagine that our training dataset contains all the points in the domain Ω, and at each iteration step, we train the model based on only one mini-batch of the entire dataset. The training process is ended if certain stopping criteria are satisfied. We find that such a procedure, even though it seems to be less accurate in evaluating the energy at first glance, is more stable in the sense that the loss will remain close to the true energy.

3.2. Selection of optimizer

Even though the energy admits a global minimizer, as shown in the previous section, there may be local minimizers in the space of network parameters. On the other hand, since we re-select the training points in each iteration, the loss landscape will be different for each iteration. So it is not suitable to use traditional descent method such as the gradient descend method for minimization. In this paper, we exploit the Adaptive Moment Estimation (Adam) optimizer [20], that can deal with sparse gradients and non-stationary objectives, which is well-suited to the current aim. It has been empirically found that Adam has never underperformed SGD in general [35].

4. Numerical results

In this section, we perform several numerical tests in two, three, and six dimensions for elliptic problems with singular sources using the developed shallow Ritz method. In the following examples, we choose sigmoid as the activation function. After the training process is complete, we check the accuracy by computing the error between the neural network solution and the exact solution of the problem. To do this, we randomly choose N_{test} testing points lying in Ω to compute the relative L_∞ and L_2 errors as

$$\frac{\|u_{\mathcal{S}} - u\|_\infty}{\|u\|_\infty}, \quad \frac{\|u_{\mathcal{S}} - u\|_2}{\|u\|_2},$$

respectively, where

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

Here u is the exact solution of Eqs. (1)-(2) while $u_{\mathcal{S}}$ is the solution obtained from the present model. In general, we set $N_{test} = 100M$, where M is the number of training points in Ω. We always terminate the training procedure after 50000 iterations with a fixed learning rate 5×10^{-3}.

Throughout all numerical experiments conducted here, the training time of the present network typically takes a few minutes on a desktop with 8-core Intel i7-10700k CPU and a Nvidia 2080Ti graphic card. Particularly, we found that the training time usually scales linearly with the number of training points but only weakly with the number of neurons. Regarding the choice of the number of neurons, we usually increase the number of neurons until the loss value (also the approximation of the energy) tends to converge. As you can see in our numerical results, just a moderate number of neurons (up to 40) is generally sufficient to give predictive accuracy less than 1% relative error in L_2 norm.

In the following numerical experiments, we will just list the analytic form of the exact solution in Ω and the level set function to represent the interface Γ. The resultant function $f(\mathbf{x})$ and the source density $c(\mathbf{s})$ in Eq. (1) can be easily computed via the equation itself and the derivative normal jump condition $[\![\partial_n u]\!] = c(\mathbf{s})$, respectively.

Example 1. In the first example, we choose a square domain $\Omega = [-1, 1] \times [-1, 1]$ with a circular interface that can be labeled by the zero-level set of the function $\phi(x, y) = x^2 + y^2 - 0.5^2$. The exact solution is chosen as

$$u(x, y) = \begin{cases} -\ln\left(x^2 + y^2\right) & (x, y) \in \Omega^+, \\ -\ln(0.5^2) & (x, y) \in \Omega^-. \end{cases} \tag{14}$$

We choose $\alpha = 0$ in this example.

Fixed training points. At first, we would like to show the over-fitting problem of using training points that are fixed throughout the entire optimization process. Instead of using the Monte Carlo method to approximate the energy, we use the midpoint rule with fixed uniform grid points on domain Ω, interface Γ, and domain boundary $\partial\Omega$, which is supposed to be more accurate in approximating the integrals. Notice that, the midpoint quadrature rule leads to exactly the same formula for the loss function as in Eq. (13). More precisely, in the selection of training points, we use 40×40 uniform quadrature nodes in the domain Ω as

$$(x_i, y_i) = (-1 + (i - 1/2)\,\Delta x, -1 + (j - 1/2)\,\Delta y), \quad i = 1, \cdots, 40, \quad j = 1, \cdots, 40, \tag{15}$$

where $\Delta x = \Delta y = \frac{1}{20}$ is the grid size. Both the interface and the domain boundary have 160 grids points, which are also uniformly distributed. So overall we have $M = 1600$, $M_\Gamma = M_b = 160$ which gives 1920 training points in total. The shallow neural network consists of one hidden layer with 30 neurons, for a total number of $N_p = 151$ parameters to be trained. The penalty constant is set by $\beta = 200$. We note that the number of training points is much larger than the number of parameters used in this case.

The training loss obtained during the optimization process is shown in Fig. 1(a) as a solid (blue) line. One can see that the loss function indeed decreases throughout the entire training process. Meanwhile, the losses corresponding to the domain (Ω), domain boundary ($\partial\Omega$) and the interface (Γ) are also shown in panels (b), (c) and (d) as solid (blue) lines, respectively.

In addition, in Fig. 1(a), we plot a dotted (black) line indicating the energy value that corresponds to the exact solution; that is, the theoretical minimum energy. As the loss function is a discrete representation of the energy, one might expect to have the same global energy minimizer so the loss should not go below the theoretical minimum value. However, it is surprising to see that the training loss becomes significantly lower than the dotted line at about 25000 iterations so consequently the solution obtained afterwards is completely different from the exact one, although is not shown here. To find out what went wrong, we carefully re-evaluated all terms in the loss function (13) using 10^6 testing points throughout training to ensure accurate evaluations of its continuous counterparts in the energy. The results are shown as dashed (red) lines in Fig. 1. We infer these values as the actual corresponding energy values in training process, as they are more accurately representing the true energy values. It can also be seen in panel (a) that the actual energy during training is indeed always larger than the theoretical minimum value which is consistent with our previous analysis. Besides, at about 25000 iterations, the actual energy is increasing, unlike the training loss that continues to decrease. We also observe that the contributions of loss from the domain boundary and interface are accurately predicted while the one from the domain (shown in panel (b)) is far from the correct value at later stage of training. Consequently the training loss is totally different from its actual energy during training (see Fig. 1(a) at later stage of iterations) and predicts the values that are much less than the correct ones.

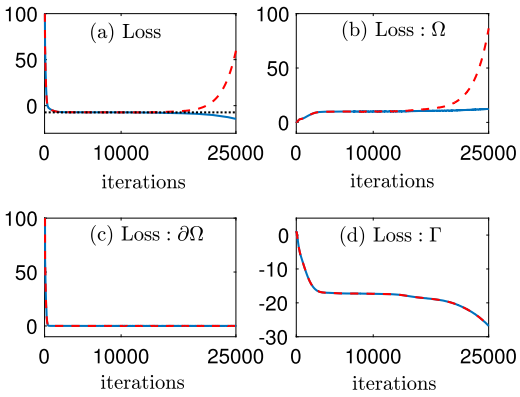

Fig. 1. Evaluation of the loss functions. The solid (blue) line shows the loss obtained during training process while the dashed (red) line shows a re-evaluation of the loss using testing points. The dotted (black) line in (a) indicates the theoretical energy value corresponding to the exact solution. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

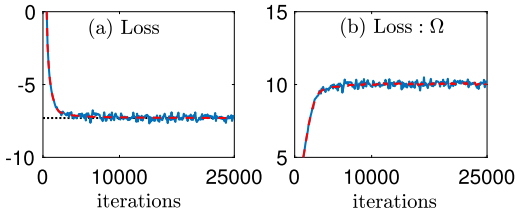

Fig. 2. Evaluation of the loss functions. The solid (blue) line shows the loss obtained during training process while the dashed (red) line shows a re-evaluation of the loss using 10^6 testing points. The dotted (black) line in (a) indicates the theoretical energy value corresponding to the exact solution.

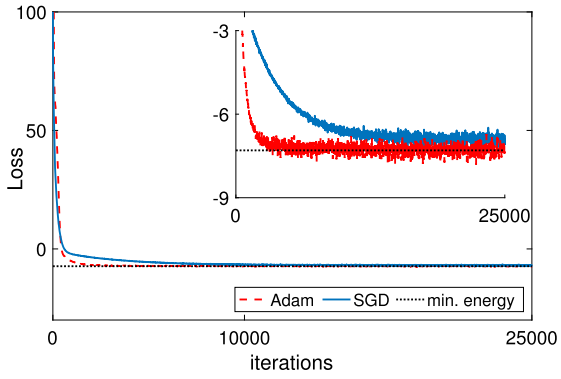

Fig. 3. Comparison between the performance of optimizers. The dotted (black) line indicates the theoretical minimum energy given by the exact solution. The inset shows a zoom-in to the region close to the minimum energy.

Such a scenario is known as over-fitting: the model learns too much detail of the training data and it deteriorates the performance of the model on new data. We therefore conclude that using fixed training points causes the problem of over-fitting. We should also point out that it happens even when the number of training points is much greater than the number of parameters (here in this experiment $M + M_\Gamma + M_b = 1920$ vs. $N_p = 151$). To overcome such a difficulty, in the following we use a strategy similar to the mini-batch gradient descent method; that is, we re-select the training points randomly in each iteration during the optimization process. As the training set is not fixed throughout the entire process, the loss evaluation in each iteration is made with a fresh set of points and therefore does not inherently overfit. The above confirmation is shown in Fig. 2(a) where we solve Example 1 again, but with 1600 randomly selected training points in each iteration. This time, the training and testing loss, shown as solid (blue) and dashed (red) lines respectively, align nicely with the theoretical energy minimum, which shows clearly the training is not over-fitted. Meanwhile, the contribution of loss from the domain also predicts correctly, see Fig. 2(b).

Comparison of optimizers. With the above re-selection strategy, we then compare the training performance with two different optimizers: Adam and stochastic gradient descent (SGD). The results are shown in Fig. 3 where the dashed (red) and solid (blue) lines show the training losses corresponding to Adam and SGD, respectively. The dotted (black) line indicates

Table 1
Comparison between immersed boundary method and shallow Ritz method. u_{IB}: Solution obtained by the IB method. $u_{\mathcal{S}}$: Solution obtained by the present model. u: Exact solution. $(M, M_\Gamma, M_b) = (200, 80, 80)$, $\beta = 200$.

N_{deg}	$\|u_{IB} - u\|_\infty / \|u\|_\infty$	(N, N_p)	$\|u_{\mathcal{S}} - u\|_\infty / \|u\|_\infty$	$\|u_{\mathcal{S}} - u\|_2 / \|u\|_2$
6400	1.9333e−02	(10, 51)	1.8172e−02	1.1883e−02
25600	9.7151e−03	(20, 101)	9.5521e−03	6.7409e−03
102400	4.8355e−03	(30, 151)	7.5025e−03	6.8292e−03

Table 2
Comparison between the networks with or without the third level set function input. $u_{\mathcal{T}}$: Solution obtained by a shallow network with 2 neurons in the input layer. $u_{\mathcal{S}}$: Solution obtained by the present model. u: Exact solution. $(M, M_\Gamma, M_b) = (1600, 160, 160)$, $\beta = 200$.

(N, N_p)	$\|u_{\mathcal{T}} - u\|_\infty / \|u\|_\infty$	$\|u_{\mathcal{T}} - u\|_2 / \|u\|_2$	(N, N_p)	$\|u_{\mathcal{S}} - u\|_\infty / \|u\|_\infty$	$\|u_{\mathcal{S}} - u\|_2 / \|u\|_2$
(10, 41)	4.0140e−01	3.9491e−01	(10, 51)	1.7232e−02	8.3483e−03
(30, 121)	3.6079e−01	2.4959e−01	(20, 101)	7.1492e−03	3.7503e−03
(500, 2001)	3.2370e−01	2.1671e−01	(30, 151)	5.5734e−03	3.6137e−03

the theoretical minimum energy attained by the exact solution. It can be seen that the losses of the two optimizers decrease rapidly, while the Adam optimizer converges slightly faster.

We also show an enlarged view of the region near the minimum energy in the inset of the figure. The loss oscillates because of the stochastic nature of the training points selection, as expected. But on average, Adam converges faster and is closer (more accurate) to the minimum energy than SGD. Therefore, in all the following examples, we use Adam optimizer.

Comparison with the immersed boundary method. Here we compare the solution of the present shallow Ritz method with the numerical solution obtained by the IB method, which is known to be a first-order accurate finite difference method for equations (1)-(2) on Cartesian grids [1] due to the employment of the discrete (or regularized) delta function. Note that in the IB method, the total number of degrees of freedom (unknowns), denoted by N_{deg}, is equal to the sum of the number of the Cartesian grid points m^2. We also denote m_Γ as the number of Lagrangian markers on the interface.

Table 1 shows the comparison results. The IB method uses the grid resolutions $m = m_\Gamma = 80$, 160, and 320 corresponding to the number $N_{deg} = 6400$, 25600, and 102400, respectively, while the present shallow Ritz method uses $N = 10$, 20, and 30 neurons in the hidden layer of the network, corresponding to just $N_p = 51$, 101, and 151 parameters. One can see how significantly different those numbers of unknowns are. Using just a few number of neurons with training points $(M, M_\Gamma, M_b) = (200, 80, 80)$, and the penalty constant $\beta = 200$, the results obtained by the present network are comparable with the ones obtained by the IB method even in the relative L_∞ errors.

The accuracy of the proposed shallow Ritz method possibly depends on several factors. One is the penalty constant β. With a finite penalty constant, the minimizer of the energy functional is changed to satisfy a Robin-type boundary condition that introduces an $O(\beta^{-1})$ error. The second one is the quadrature rule that approximates the energy functional. Here, we choose the Monte Carlo type quadrature rule that has convergence rate $O(M^{-1/2})$, where M is the number of sampling points. The third one is the precision of the computation. Neural networks can be easily implemented on GPUs to take full advantage of parallel computing but at the expense of accuracy. However, we also implement the code running in double precision by fixing β and M but does not seem to improve the overall accuracy significantly. So once the number of neurons is sufficient (here $N = 20$), the magnitude of the error becomes steady. This explains why an increase in the number of neurons to $N = 30$ does not help improve the accuracy.

Example 2. In the second example, we aim to demonstrate the effectiveness of the present level set function augmentation described in Subsection 2.3 by solving the problems with or without the level set function input. We choose a setup similar to the previous example, a square domain $\Omega = [-1, 1] \times [-1, 1]$ with a circular interface of radius 0.5 centered at the origin. We choose $\alpha = 1$ and the exact solution is given as

$$u(x, y) = \begin{cases} -\ln\left(x^2 + y^2\right) + \sin(x) + \sin(y) & (x, y) \in \Omega^+, \\ -\ln(0.5^2) + \sin(x) + \sin(y) & (x, y) \in \Omega^-. \end{cases} \tag{16}$$

If one does not require an augmented variable, i.e., not taking the third input of a level set function, a shallow network can be developed with just 2 neurons in the input layer (recall for the present shallow Ritz method, there are 3 neurons in the input layer for two-dimensional problems). For this network, the total number of parameters becomes $(d + 2)N + 1$ where N is the number of neurons used in the hidden layer. The network can be trained with exactly the same loss function defined in Eq. (13), and we denote the solution without augmentation as $u_{\mathcal{T}}$.

Table 2 shows a comparison between the accuracy performance of $u_{\mathcal{T}}$ (without level set function input) and $u_{\mathcal{S}}$ (with level set function input) where the number of training points are given as $(M, M_\Gamma, M_b) = (1600, 160, 160)$. In fact, it is not

Table 3
Comparison between the networks with different penalty constant β. $u_{\mathcal{S}}$: Solution obtained by the present model in Example 2. u: Exact solution. $(M, M_\Gamma, M_b) = (1600, 160, 160)$. The number of neurons and number of parameters $(N, N_p) = (30, 151)$.

β	$\|u_{\mathcal{S}} - u\|_\infty / \|u\|_\infty$	$\|u_{\mathcal{S}} - u\|_2 / \|u\|_2$
1	3.6104e−01	5.3255e−01
10	4.9901e−02	6.4870e−02
100	6.9001e−03	6.7623e−03

Table 4
Comparison between the networks with different depths. $u_{\mathcal{S}}$: Solution obtained by the present model in Example 2. u: Exact solution. $(M, M_\Gamma, M_b) = (1600, 160, 160)$, $\beta = 200$.

(k, N_k, N_p)	$\|u_{\mathcal{S}} - u\|_\infty / \|u\|_\infty$	$\|u_{\mathcal{S}} - u\|_2 / \|u\|_2$
(1, 30, 151)	5.5734e−03	3.6137e−03
(2, 10, 161)	9.0884e−03	5.8180e−03
(3, 7, 148)	7.4770e−03	6.2458e−03
(3, 30, 2011)	9.2629e−03	7.7047e−03

Table 5
$u_{\mathcal{S}}$: Solution obtained by the present model in Example 3. u: Exact solution. $(M, M_\Gamma, M_b) = (400, 80, 80)$, $\beta = 200$.

(N, N_p)	$\|u_{\mathcal{S}} - u\|_\infty / \|u\|_\infty$	$\|u_{\mathcal{S}} - u\|_2 / \|u\|_2$
(30, 151)	1.7095e−02	8.8148e−03
(40, 201)	1.2277e−02	8.5489e−03

required using such many training points in the domain, but we just want to avoid insufficient training points in all the following runs.

Indeed, the augmented input carrying level set function information can effectively improve training accuracy. For $u_{\mathcal{T}}$, the network output fails to capture the exact solution as one can see the relative L_∞ error is of the order 10^{-1} even with 500 neurons in the hidden layer (2001 parameters). However, for the present network with level set function augmentation, using 20 neurons in the hidden layer is already capable to approximate the solution to the order 10^{-3} both in relative L_∞ and L_2 errors. This example precisely shows the effectiveness of the augmented variable for carrying additional level set information to the network.

To quantify further the sources of error in the proposed method, in Table 3 we show the experiments to see the effect of the penalty constant β. It reveals that the error is linearly proportional to $1/\beta$. Such a result is also consistent with the analysis in Sec. 2.2, where we have shown the global minimizer satisfies a boundary condition that is changed by $O(1/\beta)$ to the true one.

We also conduct experiments to test the effect of network depth structure on the error. Here, we consider k-hidden layer network, where $k = 1, 2, 3$. To have a fair comparison, we choose the number of neurons in each hidden layer N_k accordingly so that the total parameters N_p to be trained are similar. Table 4 shows the relative L_∞ and L_2 errors for those k-hidden layer networks. One can immediately see that for networks with roughly the same number of parameters, the magnitudes of the error are quite similar. Also, an increase in the number of parameters does not really improve the accuracy for the present problem, as clearly shown by the result of the three-hidden layer network $(k, N_k, N_p) = (3, 30, 2011)$ where the error remains in the order of 10^{-3}.

Example 3. In the third example, we highlight the mesh-free nature of neural network by considering an irregular domain Ω that is given by a polar curve $r(\theta) = 1 - 0.2\cos(5\theta)$. The interface Γ is chosen as an ellipse that can be labeled by the zero level set of the function $\phi(x, y) = \frac{x^2}{0.7^2} + \frac{y^2}{0.5^2} - 1$. We fix $\alpha = 0$ and the exact solution is chosen as

$$u(x, y) = \begin{cases} \ln\left(\frac{x^2}{0.7^2} + \frac{y^2}{0.5^2}\right) & (x, y) \in \Omega^+, \\ \sin(x)\cos(y)\left[\left(\frac{x^2}{0.7^2} + \frac{y^2}{0.5^2}\right)^2 - 1\right] & (x, y) \in \Omega^-. \end{cases} \tag{17}$$

We randomly sample training points in the domain with $(M, M_\Gamma, M_b) = (400, 80, 80)$. The results are shown in Table 5 and the solutions are accurate to the order of 10^{-3} in relative L_2 error with just 30 neurons in the hidden layer. We should also point out that it can be quite tedious to implement for traditional finite difference methods to solve the problem on

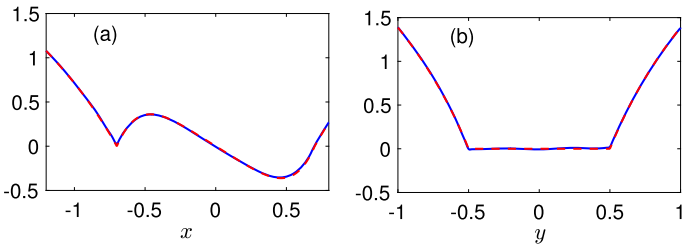

Fig. 4. Cross sections at (a) $y = 0$ and (b) $x = 0$ for the solution in Example 3 with $N = 40$. The model solution $u_{\mathcal{S}}$ is shown as solid (blue) line and the exact solution is shown as dashed (red) line.

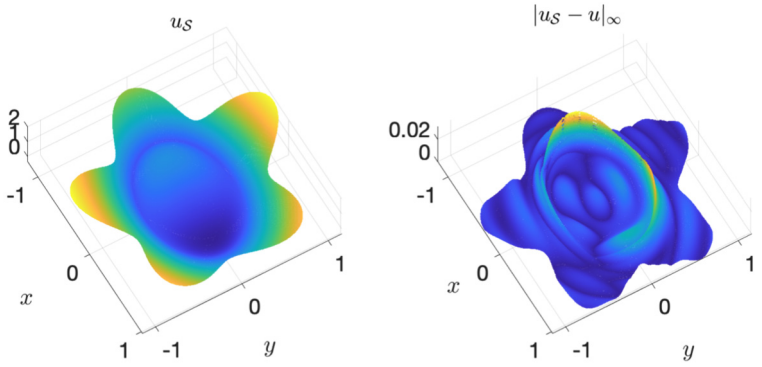

Fig. 5. Left: The solution profile obtained by the present shallow Ritz method with $N = 40$. Right: The profile of absolute error.

Table 6
$u_{\mathcal{S}}$: Solution obtained by the present model in Example 4 (three-dimensional case). u: Exact solution. $(M, M_\Gamma, M_b) = (216, 216, 216)$, $\beta = 100$.

(N, N_p)	$\|u_{\mathcal{S}} - u\|_\infty / \|u\|_\infty$	$\|u_{\mathcal{S}} - u\|_2 / \|u\|_2$
(20, 121)	1.9960e−02	1.4343e−02
(30, 181)	1.6274e−02	9.9769e−03
(40, 241)	1.2727e−02	8.7665e−03

such irregular domain. Thus, we emphasize that the irregular domain case can be properly handled with no substantial difficulty.

We also show the cross-sections of the solution at $y = 0$ and $x = 0$ in Fig. 4(a) and (b), respectively. One can see that, the present network indeed accurately approximates the solution even though there are cusps at the interface. Furthermore, in Fig. 5 showing the solution profile and absolute error correspondingly, we observe that the largest error occurs in the region close to the interface, but is not significantly larger than the values in other regions.

Example 4. In this example, we show the ability of the present method to solve three-dimensional problems. We choose the domain as a cube $\Omega = [-1, 1]^3$ with a spherical interface that is labeled by a zero-level set of the function $\phi(x, y, z) = \frac{x^2}{0.4^2} + \frac{y^2}{0.4^2} + \frac{z^2}{0.4^2} - 1$. The exact solution is chosen as

$$u(x, y, z) = \begin{cases} x(-1 + \exp(0.4^2 - (x^2 + y^2 + z^2))) & (x, y, z) \in \Omega^+, \\ -1 + \cos(0.4^2 - (x^2 + y^2 + z^2)) & (x, y, z) \in \Omega^-. \end{cases} \tag{18}$$

In this test, we choose $\alpha = 1$ and $(M, M_\Gamma, M_b) = (216, 216, 216)$. The results are shown in Table 6. Again, the present network shows a good performance. The relative L_2 error is less than 1% with just 30 neurons in the hidden layer.

Example 5. As the last example, we demonstrate the capability of the present method to solve high-dimensional problem by taking the dimension size $d = 6$. For the problem setup, we consider a 6-sphere of radius 0.6 as the domain Ω enclosing another smaller 6-sphere of radius 0.5 as the interface Γ. The interface Γ can be labeled by the zero level set of the function $\phi(\mathbf{x}) = \left(\frac{\|\mathbf{x}\|_2}{0.5}\right)^2 - 1$. We fix $\alpha = 0$ and the exact solution is chosen as

Table 7
u_S: Solution obtained by the present model in Example 5 (six-dimensional case). u: Exact solution. $(M, M_\Gamma, M_b) = (500, 1065, 1065)$, $\beta = 100$.

(N, N_p)	$\|u_S - u\|_\infty / \|u\|_\infty$	$\|u_S - u\|_2 / \|u\|_2$
(10, 91)	2.8309e−02	6.4073e−03
(20, 181)	2.6612e−02	7.0114e−03
(30, 271)	2.0292e−02	6.9735e−03

Table 8
Comparison between the number of training points in Example 5. u_S: Solution obtained by the present model. u: Exact solution.

(M, M_Γ, M_b)	(N, N_p)	$\|u_S - u\|_\infty / \|u\|_\infty$	$\|u_S - u\|_2 / \|u\|_2$
(100, 278, 278)	(10, 91)	2.4877e−02	7.4379e−03
(200, 496, 496)	(10, 91)	2.5073e−02	7.2977e−03
(500, 1065, 1065)	(10, 91)	2.8309e−02	6.4073e−03

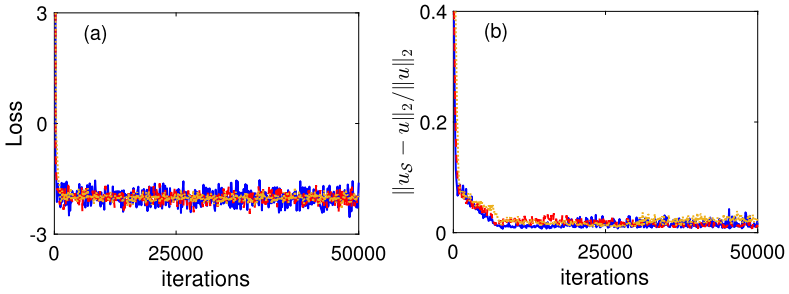

Fig. 6. The evolution plots of (a) training loss, and (b) relative L_2 error. $(M, M_\Gamma, M_b) = (100, 278, 278)$: solid (blue) line; $(200, 496, 496)$: dashed (red) line; $(500, 1065, 1065)$: dotted (yellow) line.

$$u(\mathbf{x}) = \begin{cases} \exp(0.5^2 - \|\mathbf{x}\|_2^2) + \sum_{i=1}^5 \sin(x_i) & \mathbf{x} \in \Omega^+, \\ 1 + 2\sin(0.5^2 - \|\mathbf{x}\|_2^2) + \sum_{i=1}^5 \sin(x_i) & \mathbf{x} \in \Omega^-, \end{cases} \tag{19}$$

where $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5, x_6)$.

Here, we choose the number of training points based on the following strategy. Given a radius R, the volume and surface area of the 6-sphere are $(1/6)\pi^3 R^6$ and $\pi^3 R^5$, respectively, so the ratio between these two numbers is $R^6 : 6R^5$. We then choose the number of training points in the sphere and on the surface based on this ratio; that is, if we have 500 points in the domain that corresponds to the effective radius $R = (500)^{1/6}$, then we select $6R^5 \approx 1065$ points on the boundary and interface.

The results are shown in Table 7. With just 10 neurons in the hidden layer, the relative L_2 error is in the order of 10^{-3}. In addition, we compare the performance between different number of training points, shown in Table 8. It is surprising to see that accuracy of the solution is already good enough by choosing $(M, M_\Gamma, M_b) = (100, 278, 278)$ in six dimensions, while doubling or even quintupling the training points does not improve the accuracy. More precisely, we also show the evolution of the training loss and the relative L_2 error during the training process in Fig. 6. One can see that the results are almost indistinguishable from three different training point choices. Here, even in 6 dimensions, we only need as few as 656 training points to achieve the desired accuracy. A possible explanation is that the error of the Monte Carlo integration is independent of the problem dimension. Although we have conducted other possible sources of error such as the choice of penalty number β and the network depth in Example 2, a detailed error analysis is still required for further investigation.

5. Conclusion

In this paper, a novel shallow Ritz method is developed to solve elliptic problems with delta function singular sources on the interface. By introducing an energy functional, we reformulate the governing equation as a variational problem. The crucial observation is that the contribution of the singular source in the equation becomes a regular surface integral term in the energy. Therefore, we do not need to introduce the discrete delta function used in traditional regularization methods, such as the immersed boundary method. To enforce the boundary condition, we add a penalty term to the energy and find that this treatment changes the global minimizer to one that satisfies Robin-type boundary condition.

We propose a shallow neural network to approximate the global minimizer of the energy functional, and train the network by minimizing a loss function that presents a discrete version of the energy. In addition, we include the level set

function as an additional feature input to the network and find that it significantly improves the training accuracy. We perform a series of numerical tests to show the accuracy and efficiency of the present network and its capability to handle problems in irregular domains or high dimensions. As shown in numerical experiments in this paper, most testing problems can be solved with acceptable accuracy by the present network with moderate number of neurons (no larger than 40). Although the present network is similar in spirit to the deep Ritz method [7], here we consider a completely shallow one (only one hidden layer) so it significantly reduces the computational complexity and learning workload without sacrificing the accuracy.

We have to emphasize that it is not our intention to compete with the traditional methods such as immersed boundary (IB) method, immersed interface method (IIM), immersed finite element (IFE) method or other grid-based methods listed in the reference. Instead, we just provide an alternative using neural network method and follow the recent pioneering deep Ritz method by E and Yu in [7] to solve the elliptic interface problems with singular delta function sources. Since the underlying solution of the problem is usually not smooth (the function is continuous but not for its derivatives) across the interface, the traditional accurate methods (for instance IIM, IFE method, see the book of Li and Ito [24]) need special treatments near the interface. However, due to the mesh-free advantage, the present neural network uses the level set function of the interface as a feature input and learns the solution directly. Once the network is trained successfully, the solution at any point in the domain can be obtained by a feedforward one hidden layer neural network approximation. So from this point of view, it is efficient. In fact, the training time of the present network takes only a few minutes on a desktop, even for a 6-dimensional problem listed in Example 5. Meanwhile, neural networks can be easily implemented on GPUs to take full advantage of parallel computation and are easy to deal with problems defined in irregular domains as well. Another advantage is that the present approach can handle high dimensional problems with exactly same framework and the number of hyperparameters (unknowns in neural network method) to be trained scales only linearly with the dimension.

As mentioned before, the accuracy of the proposed shallow Ritz method possibly depends on three factors, namely the penalty constant β, the quadrature rule for estimating the energy functional, and the precision of the computation. To improve accuracy, we must consider three factors simultaneously. We believe that the present one hidden layer network is not the reason caused the accuracy no better than 10^{-3} in relative L_∞ or L_2 error throughout all numerical experiments here. The one hidden layer network with sufficiently smooth activation function has been proved to be capable to represent an arbitrary function and its derivatives to arbitrary accuracy. From the numerical results shown in Section 4, the accuracy obtained by the present method is comparable with the one existing in related literatures despite the fact that the present network has simplest structure (one hidden layer). For example, the pioneering work of E and Yu in [7] where the authors used the deep learning network ResNet (say a stack of 4 blocks with 2 fully-connected layers in each block), the relative L_2 errors for all considered problems are also up to 10^{-3}. In the recent work of Guo and Yang in [14], a similar ResNet was applied to solve elliptic interface problems where the jump conditions are put into their unfitted Nitsche's energy functional. Again, most of their numerical results in relative L_2 errors are no better than 10^{-3}. Certainly, it will be interesting to investigate further to improve the accuracy of the Ritz-type neural network method for solving PDEs which we shall leave as our future work. Nevertheless, from the authors' point of view, the Ritz-type neural network is still valuable and useful for computational physics applications due to following two reasons. First, it is completely mesh-free for high-dimensional problems when traditional numerical methods are hard to tackle. Second, in many interesting physical problems, finding a solution to the problem is often equivalent to finding the minimum of its energy law, which falls into the Ritz-type neural network methodology naturally.

The present goal is to solve some elliptic interface problems using a Ritz-type neural network. Indeed, for given Ω, Γ, $f(\mathbf{x})$, $c(\mathbf{s})$ and the boundary condition, we need to train a new network that satisfies the equation and boundary condition simultaneously. These given information are all from the original PDE formulation so there is no difficulty in generating the training data. (That is, generating training data from a well-posed PDE can be done without knowing any numerical techniques for solving that PDE.) However, when using some traditional numerical techniques such as IB method for the present problem, if the domain Ω and mesh size are fixed, the matrix of resultant linear system remains the same even if the interface Γ and the right-hand side functions $f(\mathbf{x})$, $c(\mathbf{s})$ are changed. So from this point of view, the IB method is more generalizable. Therefore, to improve the generalizability of the present network, we might consider to design a neural network as an operator representation that has the solution as the output. But this is beyond the scope of the paper and we leave it as our future work.

In the present work, we only consider stationary elliptic problems with constant coefficients. There should be no difficulty in considering problems with contrast coefficients or even variable coefficients. The same framework can be applied straightforwardly by writing the corresponding energy functionals. As a forthcoming extension, we shall consider time-dependent problems, and particularly the moving interface problems.

CRediT authorship contribution statement

Ming-Chih Lai: Conceptualization, Methodology, Supervision, Writing – original draft, Writing – review & editing. **Che-Chia Chang:** Numerical experiments and Visualization. **Wei-Syuan Lin:** Numerical experiments and Visualization. **Wei-Fan Hu:** Supervision, Visualization and Writing – original draft. **Te-Sheng Lin:** Conceptualization, Methodology, Supervision, Writing – original draft, Writing – review & editing.

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

T.-S. Lin and W.-F. Hu acknowledge supports by Ministry of Science and Technology (MOST), Taiwan, under research grant 109-2115-M-009-006-MY2 and 109-2115-M-008-014-MY2, respectively. T.-S. Lin and W.-F. Hu also acknowledge support by NCTS of Taiwan. M.-C. Lai acknowledges the support by MOST, Taiwan, under research grants 108-2119-M-009-012-MY2 and 110-2115-M-A49-011-MY3.

References

[1] R.P. Beyer, R.J. LeVeque, Analysis of a one-dimensional model for the immersed boundary method, SIAM J. Numer. Anal. 29 (2) (1992) 332–364.
[2] D. Bochkov, F. Gibou, Solving Poisson-type equations with Robin boundary conditions on piecewise smooth interfaces, J. Comput. Phys. 376 (2019) 1156–1198.
[3] D. Bochkov, F. Gibou, Solving elliptic interface problems with jump conditions on Cartesian grids, J. Comput. Phys. 407 (2020) 109269.
[4] R.B. Balam, F. Hernandez-Lopez, J. Trejo-Sanchez, M.U. Zapata, An immersed boundary neural network for solving elliptic equations with singular forces on arbitrary domains, Math. Biosci. Eng. 18 (1) (2020) 22–56.
[5] Z. Cai, J. Chen, M. Liu, X. Liu, Deep least-squares methods: an unsupervised learning-based numerical method for solving elliptic PDEs, J. Comput. Phys. 420 (2020) 109707.
[6] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Syst. 2 (1989) 303–314.
[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] R. Egan, F. Gibou, Geometric discretization of the multidimensional Dirac delta distribution - application to the Poisson equation with singular source terms, J. Comput. Phys. 346 (2017) 71–90.
[9] R. Egan, F. Gibou, xGFM: recovering convergence of fluxes in the ghost fluid method, J. Comput. Phys. 409 (2020) 109351.
[10] F. Gibou, D. Hyde, R. Fedkiw, Sharp interface approaches and deep learning techniques for multiphase flows, J. Comput. Phys. 380 (2019) 442–463.
[11] F. Gibou, C. Min, Efficient symmetric positive definite second-order accurate monolithic solver for fluid/solid interactions, J. Comput. Phys. 231 (2012) 3246–3263.
[12] J.L. Guermond, P. Minev, J. Shen, An overview of projection methods for incompressible flows, Comput. Methods Appl. Mech. Eng. 195 (44) (2006) 6011–6045.
[13] A. Guittet, M. Lepilliez, S. Tanguy, F. Gibou, Solving elliptic problems with discontinuities on irregular domains - the Voronoi interface method, J. Comput. Phys. 298 (2015) 747–765.
[14] H. Guo, X. Yang, Deep unfitted Nitsche method for elliptic interface problems, Commun. Comput. Phys. 31 (4) (2022) 1162–1179.
[15] J. Han, A. Jentzen, W. E, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci. 115 (2018) 8505–8510.
[16] B. Hanin, M. Sellke, Approximating continuous functions by ReLU nets of minimal width, arXiv:1710.11278, 2018.
[17] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (1989) 359–366.
[18] W.-F. Hu, T.-S. Lin, M.-C. Lai, A discontinuity capturing shallow neural network for elliptic interface problems, arXiv:2106.05587, 2021.
[19] 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.
[20] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, in: International Conference on Learning Representations (ICLR), 2015.
[21] M.-C. Lai, H.-C. Tseng, A simple implementation of the immersed interface methods for Stokes flows with singular forces, Comput. Fluids 37 (2) (2008) 99–106.
[22] R.J. LeVeque, Z. Li, The immersed interface method for elliptic equations with discontinuous coefficients and singular sources, SIAM J. Numer. Anal. 31 (4) (1994) 1019–1044.
[23] Z. Li, On convergence of the immersed boundary method for elliptic interface problems, Math. Comput. 84 (2015) 1169–1188.
[24] Z. Li, K. Ito, The Immersed Interface Method, SIAM, 2006.
[25] 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.
[26] X.-D. Liu, T. Sideris, Convergence of the ghost fluid method for elliptic equations with interfaces, Math. Comput. 72 (2003) 1731–1746.
[27] Z. Lu, H. Pu, F. Wang, Z. Hu, L. Wang, The expressive power of neural networks: a view from the width, in: NIPS'17, 2017, pp. 6232–6240.
[28] C.S. Peskin, Numerical analysis of blood flow in the heart, J. Comput. Phys. 25 (1977) 220–252.
[29] C.S. Peskin, The immersed boundary method, Acta Numer. 11 (2002) 479–517.
[30] A. Pinkus, Approximation theory of the MLP model in neural networks, Acta Numer. 8 (1999) 143–195.
[31] 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.
[32] H. Sheng, C. Yang, PFNN: a penalty-free neural network method for solving a class of second-order boundary-value problems on complex geometries, J. Comput. Phys. 428 (2021) 110085.
[33] J. Sirignano, K. Spiliopoulos, DGM: a deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364.
[34] Z. Wang, Z. Zhang, A mesh-free method for interface problems using the deep learning approach, J. Comput. Phys. 400 (2020) 108963.
[35] D. Choi, C.J. Shallue, Z. Nado, J. Lee, C.J. Maddison, G.E. Dahl, On empirical comparisons of optimizers for deep learning, arXiv:1910.05446, 2019.