


Article

# Deep Learning Nonhomogeneous Elliptic Interface Problems by Soft Constraint Physics-Informed Neural Networks

Fujun Cao <sup>1,2,\*</sup> , Xiaobin Guo <sup>3</sup>, Fei Gao <sup>3</sup> and Dongfang Yuan <sup>1,2</sup>

<sup>1</sup> School of Science, Inner Mongolia University of Science and Technology, Baotou 014010, China; yuandf@imust.edu.cn

<sup>2</sup> School of Mathematics and Science, Inner Mongolia Normal University, Hohhot 010028, China

<sup>3</sup> School of Information Engineering, Inner Mongolia University of Science and Technology, Baotou 014010, China; guoxb0710@163.com (X.G.); 970114532@163.com (F.G.)

\* Correspondence: caofujun@imust.edu.cn

**Abstract:** It is a great challenge to solve nonhomogeneous elliptic interface problems, because the interface divides the computational domain into two disjoint parts, and the solution may change dramatically across the interface. A soft constraint physics-informed neural network with dual neural networks is proposed, which is composed of two separate neural networks for each subdomain, which are coupled by the connecting conditions on the interface. It is beneficial to capture the singularity of the solution across the interface. We formulate the PDEs, boundary conditions, and jump conditions on the interface into the loss function by means of the physics-informed neural network (PINN), and the different terms in the loss function are balanced by optimized penalty weights. To enhance computing efficiency for increasingly difficult issues, adaptive activation functions and the adaptive sampled method are used, which may be improved to produce the optimal network performance, as the topology of the loss function involved in the optimization process changes dynamically. Lastly, we present many numerical experiments, in both 2D and 3D, to demonstrate the proposed method's flexibility, efficacy, and accuracy in tackling nonhomogeneous interface issues.

**Keywords:** partial differential equations; physics-informed neural networks; nonhomogeneous elliptic interface problems; dual neural networks

**MSC:** 35J05; 35J25; 35Q79



**Citation:** Cao, F.; Guo, X.; Gao, F.; Yuan, D. Deep Learning Nonhomogeneous Elliptic Interface Problems by Soft Constraint Physics-Informed Neural Networks. *Mathematics* **2023**, *11*, 1843. <https://doi.org/10.3390/math11081843>

Academic Editors: Nicholas Christakis, George Kossioris and Mayur Patel

Received: 18 February 2023

Revised: 18 March 2023

Accepted: 28 March 2023

Published: 13 April 2023



**Copyright:** © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

Elliptic interface problems with discontinuous coefficients and singular sources are found in many applications, such as incompressible two-phase flow [1,2], computational electro-magnetics [3], heat conduction between materials of different heat capacity and conductivity [4–6], thermal convection in a Forchheimer–Brinkman model [7,8], etc. The solution of elliptic interface problems with discontinuous coefficients has low global regularity and usually belongs to  $H^1(\Omega)$  if the jump conditions are homogeneous. When the jump conditions are nonhomogeneous, the solutions to the elliptic interface problems are often discontinuous [9]. It is a difficult job to construct an efficient numerical method for such problems. In the past decades, various numerical methods have been provided to solve these kinds of problems. According to the geometric relationship between the computational grid and the material interface, the numerical method can be generally divided into two categories: (1) The interface-fitted methods [10–14]. In this kind of method the computational mesh fits the interface, which means that an element of the underlying mesh is required to intersect with the interface only through its edges. This approach is beneficial for the numerical scheme to reach optimal convergence. Material interactions in real-world applications can be geometrically complex and very heterogeneous. Geometric singularities, such as sharp edges, cusps, and tips, may be encountered in some extreme

circumstances with nonsmooth surfaces or interfaces with Lipschitz continuity. Yet, creating high-quality meshes for some exotically complicated geometries may be challenging and time-consuming. Such a challenge drives the development of numerical algorithms that employ structured meshes and allow the interface to cut through elements. (2) The interface-unfitted method [15–20]. By allowing the interface to cut computational elements, the immersed interface approach lessens the effort of creating meshes for complicated domains and interface geometry. Specific interface techniques are required to manage interface jump conditions, which are essential for the well-posedness of the interface problem. The convergence order in the  $L_\infty$  norm oscillates while using the interface immersed numerical approach, and the point-by-point error across the interface elements is often large.

To secure accuracy near the interface, the jump conditions on the interface have to be incorporated into the numerical discretization in a certain manner, and some special treatment needs to be introduced on the elements through which the interface passes. Although both sorts of solutions have had some success in tackling interface difficulties, implementing such numerical schemes is not an easy undertaking, due to the inhomogeneous jump conditions on the interface. High dimensional interface difficulties, nonlinear interface problems, and other generic interface problems continue to provide significant hurdles to traditional numerical methods.

Deep neural networks (DNNs) have recently received a lot of interest in the field of scientific machine learning (SciML) and have been used to build new ways of solving partial differential equations, e.g., the deep Ritz method (DRM) [21], deep Galerkin method [22] and physics-informed neural networks (PINNs) [23]. DNNs provide nonlinear approximation through the composition of hidden layers because of their universal approximation capabilities, which do not limit the approximation to linear spaces. In SciML, PINN has become one of the most prominent deep learning approaches. While the differential operators in the governing PDEs are approximated by automated differentiation, PINNs provide a mesh-free approach. Such techniques have also been applied in solving different types of partial differential equations (PDEs), including integro-differential equations [24], fractional PDEs [25], and stochastic PDEs [26,27]. Furthermore, PINNs have been effectively used to tackle a variety of issues in other domains, such as optics [28,29], fluid mechanics [30], systems biology [31], etc.

Several initiatives have been launched in recent years to employ neural networks to tackle interface problems, since neural network approaches are meshless and may benefit from deep learning techniques, such as automated differentiation and GPU acceleration. In particular, the use of multiple neural networks based on the domain decomposition method (DDM) have attracted increasing attention as they are more accurate and flexible in dealing with the interface and have shown remarkable success in various interface problems [32–38].

In [34], a deep-learning-based domain decomposition approach (DeepDDM) was introduced, which uses deep neural networks to discretize the subproblems split by domain decomposition methods (DDMs) to solve PDEs with complicated interfaces in the computational domain. Wang [35] proposed a mesh-free method based on DRM to solve interface problems with high-contrast discontinuous coefficients. He et al. [37] proposed a mesh-free method using piecewise DNN for elliptic interface problems with discontinuous solution and derivatives across the interface. In order to ensure that the solution is smooth in each subdomain, they approximate the solution using two neural networks that correspond to two distinct subdomains. In [33], it was suggested to use a conservative physics-informed neural network (cPINN) on discrete subdomains for nonlinear conservation laws. The computing domain is subdivided into discrete subdomains, with a different PINN applied in each subdomain. The conservation property of the cPINN is then attained by enforcing flux continuity in the strong form along the subdomain interfaces. A generalized space–time domain decomposition framework, named eXtended PINN (xPINN) was proposed in [32] to solve nonlinear PDEs in arbitrary complex-geometry domains. Wu et al. [38] performed a convergence analysis of neural network combined with domain decomposition technolo-

gies and gradient-enhanced strategies for solving second-order elliptic interface problems. It was demonstrated that, as the number of samples increased, the neural network sequence generated by minimizing a Lipschitz regularized loss function converged to the unique solution to the interface problem in  $H_2$ .

In this paper, we propose a soft constraint physics-informed neural network to solve the nonhomogeneous elliptic interface problems with discontinuous solutions and derivatives across the interface. Since the interface divides the domain into two disjoint parts, the solution may change dramatically across the interface. Instead of representing the approximate solutions on the whole domain with a single DNN structure, we employ two DNN structures to approximate the solution when the interface splits the domain into two subdomains. The solution is approximated by the PINNs, which formulate the PDEs and jump condition on the interface into the loss of the neural network. To improve the computational efficiency for more challenging problems, the adaptive activation function, as well as the adaptive sampling strategy, are employed to achieve the best performance as the network is optimized in the learning process. Lastly, we present many numerical experiments, in both 2D and 3D, to demonstrate the flexibility, efficacy, and accuracy of the proposed technique for handling interface issues.

The remainder of the paper is structured as follows. Section 2 describes the second-order elliptic interface issue and its three types of boundary conditions. The topology of neural networks with adaptive activation functions is discussed in Section 3. We present physical-informed neural networks with soft constraints in Section 4. Section 5 shows numerical data to demonstrate the efficacy of the suggested technique. Lastly, in Section 6, we draw some conclusions.

### 2. Problem Formulation

Let  $\Omega$  be a bounded domain in  $\mathcal{R}^d, d = 2, 3$ , with a border  $\partial\Omega$  and an interface  $\Gamma$  that separates  $\Omega$  into two disjoint subdomains,  $\Omega^+$  and  $\Omega^-$ . The border  $\partial\Omega$  and the interface  $\Gamma$  are assumed to be Lipschitz continuous. The interface is defined by a piecewise smooth level-set function  $\phi \in \Omega$ , such that  $\Gamma = \{\vec{x} | \phi(\vec{x}) = 0, \forall \vec{x} \in \Omega\}$ . As such, two subdomains can be given by  $\Omega^+ = \{\vec{x} | \phi(\vec{x}) \geq 0, \forall \vec{x} \in \Omega\}$  and  $\Omega^- = \{\vec{x} | \phi(\vec{x}) < 0, \forall \vec{x} \in \Omega\}$ . In some practical applications, the interface may be given as an iso value of a scalar field.  $\vec{n} = \frac{\nabla\phi}{|\nabla\phi|}$  is the outward unit normal vector of the interface. The following second-order semi-linear elliptic interface issue is taken into consideration.

$$\begin{cases} -\nabla \cdot (\kappa \nabla u) = f(\vec{x}), & \vec{x} \in \Omega^+ \cup \Omega^-, \\ \llbracket u \rrbracket = u^+ - u^- = \phi, & \vec{x} \text{ on } \Gamma, \\ \llbracket \vec{\kappa} \nabla u \cdot \vec{n} \rrbracket = \kappa^+ u_n^+ - \kappa^- u_n^- = \psi, & \vec{x} \text{ on } \Gamma, \\ u = \begin{cases} g_d, & \vec{x} \in \partial\Omega_d, \\ \kappa \nabla u \cdot \vec{n} = q, & \vec{x} \in \partial\Omega_q, \\ \kappa \nabla u \cdot \vec{n} = h(u_\infty - u), & \vec{x} \in \partial\Omega_h. \end{cases} \end{cases} \tag{1}$$

where the function  $f(\vec{x})$  describes acting source, the physical meaning of which is determined by the nature of the considered processes. The boundary  $\partial\Omega$  has been split into partitions  $\partial\Omega_d, \partial\Omega_q$ , and  $\partial\Omega_h$ , each of which is mutually exclusive. Here,  $\partial\Omega_d$  corresponds to the Dirichlet boundary condition for the given temperature  $g_d$  and  $\partial\Omega_q$  corresponds to the Neumann boundary condition with the heat flux  $q : \partial\Omega_q \rightarrow \mathcal{R}$ , and  $\partial\Omega_h$  denotes the region with Robin (convective) boundary conditions with the heat transfer coefficient  $h : \partial\Omega_h \rightarrow \mathcal{R}$ , and surrounding temperature  $u_\infty : \partial\Omega_h \rightarrow \mathcal{R}$ .  $u^+, u_n^+, \kappa^+$  indicate their limiting value from the  $\Omega^+$  side of the interface, and  $u^-, u_n^-, \kappa^-$  indicate their limiting value from the  $\Omega^-$  side of the interface  $\Gamma$ . On the interface, the derivatives  $u_n^+$  and  $u_n^-$  are assessed along the normal direction.  $\phi(\vec{x})$  and  $\psi(\vec{x})$  are at least  $C^1$  continuous.

### 3. Mathematical Setup for Dual Neural Networks Structure

A feed-forward neural network of  $L$  layers and  $N_k$  neurons in the  $k$ th layer ( $N_0 = D_i$ , and  $N_L = D_o$ ) is denoted by  $\mathcal{N} : \mathbb{R}^{D_i} \rightarrow \mathbb{R}^{D_o}$ .  $\mathbf{W}^k \in \mathbb{R}^{N_k \times N_{k-1}}$  and  $\mathbf{b}^k \in \mathbb{R}^{N_k}$  are the weight matrix and bias vector in the  $k$ th layer ( $1 \leq k \leq L$ ), respectively. The input vector is denoted by  $\vec{x} \in \mathbb{R}^{D_i}$  and the output vector at the  $k$ th layer is denoted by  $\mathcal{N}^k(\vec{x})$  and  $\mathcal{N}^0(\vec{x}) = \vec{x}$ . The activation function is denoted by  $\Phi$  and is applied layer-wise together with the scalable parameters  $na^k$ , where  $n$  is the scaling factor. The extra parameters  $a^k$  modify the slope of the activation function in each hidden layer, resulting in faster training speed. By means of the slope recovery term, the activation slopes can also influence the loss function, see [39,40] for more details. Such locally adaptable activation functions, in particular during the initial training phase, improve the network’s potential for learning. This document uses a scaling factor  $n = 5$  for all hidden layers and an initialization  $na^k = 1, \dots, k$ .

The  $(L - 1)$ -hidden layer feed-forward neural network is defined by

$$\mathcal{N}^k = \mathbf{W}^k \Phi(a^{k-1} \mathcal{N}^{k-1}(\vec{x})) + \mathbf{b}^k, \quad 2 \leq k \leq L \tag{2}$$

and  $\mathcal{N}^1(\vec{x}) = \mathbf{W}^1 \vec{x} + \mathbf{b}^1$ , where, on the first layer, the activation function is identity.  $\Theta = \{\mathbf{W}^k, \mathbf{b}^k, a^k\} \in V$  is the collection of all weights, biases, and slopes and takes  $V$  as the parameter space.

However, unlike conventional approaches that use only a single DNN to approximate the  $u(\vec{x})$  solution over the entire  $\Omega$  domain, we use two DNN structures independently to estimate  $u^+(\vec{x})$  and  $u^-(\vec{x})$  on  $\Omega^+$  and  $\Omega^-$ , respectively. For  $\vec{x} \in \Omega_i, i = 1, 2$ , a dual neural network is used to approximate  $u_i(\vec{x}), i = 1, 2$ , as follows,

$$u_i(\vec{x}) \approx \mathcal{U}_{i,\mathcal{N}}(\vec{x}, \Theta_i) = \mathcal{N}_i^L \circ \mathcal{N}_i^{L-1} \circ \dots \circ \mathcal{N}_i^2 \circ \mathcal{N}_i^1(\vec{x}), \quad i = 1, 2, \tag{3}$$

where  $\mathcal{U}_{i,\mathcal{N}}(\vec{x}, \Theta_i)$  highlights the dependence of the neural network output  $\mathcal{U}_{i,\mathcal{N}}(\vec{x})$  on  $\Theta_i$ . Then, the global approximation of  $u(\vec{x})$  can be defined as follows,

$$u(\vec{x}) \approx \mathcal{U}_{\mathcal{N}}(\vec{x}, \Theta) = \begin{cases} \mathcal{U}_{1,\mathcal{N}}(\vec{x}, \Theta), & \text{if } \vec{x} \in \Omega_1, \\ \mathcal{U}_{2,\mathcal{N}}(\vec{x}, \Theta), & \text{if } \vec{x} \in \Omega_2. \end{cases} \tag{4}$$

where  $\Theta = \Theta_i$ , if  $\vec{x} \in \Omega_i, i = 1, 2$ .  $\mathcal{U}_{\mathcal{N}}(\vec{x}, \Theta)$  emphasizes the dependence of the neural network output  $\mathcal{U}_{\mathcal{N}}(\vec{x})$  on  $\Theta$ .

### 4. Physics-Informed Neural Networks with Soft Constraints

A key feature of a PINN is that it can easily turn a PDE problem into an optimization problem by combining all available information, including control equations, empirical data, and initial/boundary conditions into a loss function. The advantage of this approach is that it provides a meshless algorithm, because the differential operators in the managed PDE are approximated by an automatic discriminant. To reduce the difficulty of neural network learning, we hope that the network constructed can, as far as possible, meet the general solution of PDE, and then achieve the goal of accelerating convergence speed and improving solution accuracy. The dual neural networks structure of the approximation  $\mathcal{U}_{\mathcal{N}}(\vec{x}, \Theta)$  is shown in Figure 1. As we can see, these two networks were used to approximate the solution in different domains. The sampling points were classified, according to the location of the sample pickup points, and then it was determined to which separate DNN network they belonged. This method can effectively approximate the singular discontinuous solution along the material interface.

For a forward problem, we define the loss function as

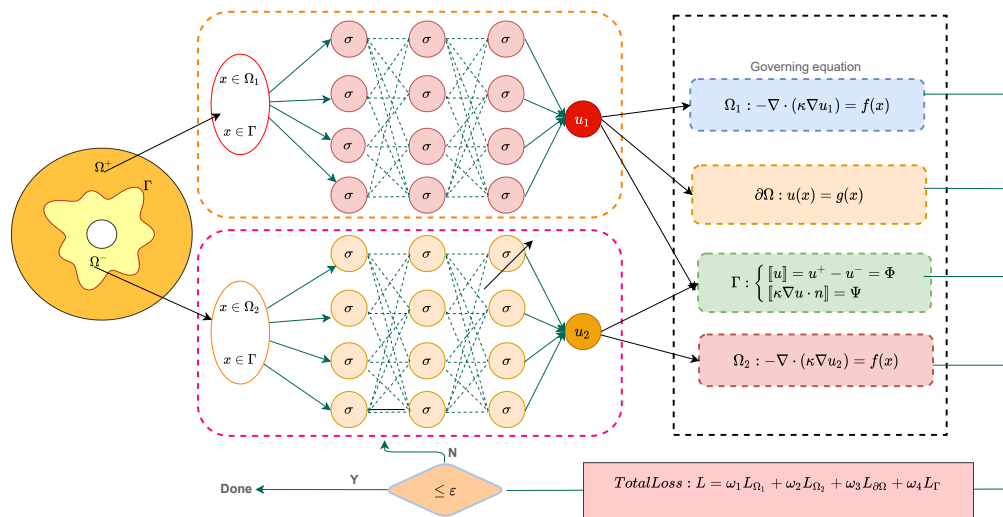
$$\mathcal{L}(\Theta) = \omega_1 L_{\Omega_1} + \omega_2 L_{\Omega_2} + \omega_3 L_{\Gamma} + \omega_4 L_{\partial\Omega} + \omega_5 L_d, \tag{5}$$

where  $L_{\Omega_1}$  and  $L_{\Omega_2}$  are the losses corresponding to the residuals of governing equations in subdomains  $\Omega_1$  and  $\Omega_2$ , respectively.  $L_{\partial\Omega}$  is the loss due to the boundary condition,  $L_{\Gamma}$

is the loss due to the interface conditions, and  $L_d$  is the loss corresponding to label data (if any).

$$\begin{aligned}
 L_{\Omega_1} &= \frac{1}{N_{r1}} \sum_{i=1}^{N_{r1}} |-\nabla \cdot (\kappa \nabla u_1(x_{r1}^i, \theta)) - f_1|^2, \\
 L_{\Omega_2} &= \frac{1}{N_{r2}} \sum_{i=1}^{N_{r2}} |-\nabla \cdot (\kappa \nabla u_2(x_{r2}^i, \theta)) - f_2|^2, \\
 L_{\Gamma} &= \frac{1}{N_I} \sum_{i=1}^{N_I} (|u^i - \phi|^2 + |\kappa \nabla u(x_{\gamma}^i, \theta) \cdot n - \psi|^2), \\
 L_{\partial\Omega} &= \frac{1}{N_b} \sum_{i=1}^{N_b} |u(x_b^i; \theta) - g_b^i|^2 \\
 L_d &= \frac{1}{N_d} \sum_{i=1}^{N_d} |\hat{u}(x_d^i, \theta) - y_d^i|^2,
 \end{aligned}$$

where  $\{\bar{x}_{r1}^i\}_{i=1}^{N_{r1}}$  and  $\{\bar{x}_{r2}^i\}_{i=1}^{N_{r2}}$  are the collocation points randomly distributed in the domain  $\Omega_1$  and  $\Omega_2$ ,  $\{\bar{x}_b^i, g_b^i = g(\bar{x}_b^i)\}_{i=1}^{N_b}$  are the boundary condition points,  $\{\bar{x}_I^i\}_{i=1}^{N_I}$  are the interface condition points, and  $\{\bar{x}_d^i, t_d^i\}_{i=1}^{N_d}$  are sample data (if any).  $N_{\Omega_1}, N_{\Omega_2}, N_{\Gamma}, N_{\partial\Omega}, N_d$  denote the total number of collocation points in two different regions, interface points, boundary points and labeled data, respectively. The weights  $\omega_1, \omega_2, \omega_3, \omega_4$  and  $\omega_5$  are the weights of residuals for two governing equations, interface conditions, boundary conditions, and labeled data. Weights in PINNs are crucial for learnability and can be set manually or auto-configured [41–43].



**Figure 1.** Schematic dual PINNs for interface problems. Two independent neural networks are constructed to approximate solutions to two subdomains, respectively.

The basic idea is to train a neural network to approximate the solution of PDE by minimizing the physically-informed loss function, given the residual of the PDEs together with the interface and boundary conditions, as follows:

$$\Theta^* = \underset{\Theta}{\operatorname{argmin}} \mathcal{L}(\Theta).$$

where  $\Theta^*$  is the minimizer and  $\mathcal{U}_{\mathcal{N}}(\bar{x}; \Theta^*)$  is the related DNN approximation.

### 5. Numerical Experiments

Several numerical tests for the elliptic interface problems in two and three dimensions were carried out, using the current soft constraint PINNs. Due to these intricate contact geometries, traditional numerical methods for handling such problems provide a significant challenge to the robustness of numerical schemes. Apart from numerical scheme concerns, the great computational complexity of typical numerical approaches for three-dimensional problems is a significant challenge in dealing with complex interface challenges. As a result, this section highlights the benefits and potential utility of deep learning approaches in dealing with such difficult numerical instances. The setup we use in the example below aims to demonstrate the robustness and efficacy of the suggested algorithm, even in the case of a small network architecture. By choosing the sigmoid as the activation function, we terminate the learning process when the stopping standardization  $Loss(\Theta) < \epsilon$  is satisfied (the  $\epsilon$  tolerance was set to at least  $10^{-5}$ ). Cross-validation was used in machine learning practice for validation. That is, we used the test error instead of the training error to measure the accuracy of the solution. The  $L_\infty$  and  $L_2$  norm errors are calculated by randomly selecting test points  $N_{test}$  located in  $\Omega$  as

$$\|\hat{u} - u\|_\infty = \max_{1 \leq i \leq N_{test}} \frac{|\hat{u}(\bar{x}^i) - u(\bar{x}^i)|}{|\hat{u}(\bar{x}^i)|}, \quad \|\hat{u} - u\|_2 = \sqrt{\frac{\sum_{i=1}^{N_{test}} |\hat{u}(\bar{x}^i) - u(\bar{x}^i)|^2}{\sum_{i=1}^{N_{test}} |\hat{u}(\bar{x}^i)|^2}}, \quad (6)$$

where  $\hat{u}$  was the function obtained by the present PINN.

**Example 1.** This example employs a sophisticated interface in the form of parameters

$$x(\theta) = (a + b \cos(m\theta)) \sin(n\theta) \cos(\theta), \quad y(\theta) = (a + b \cos(m\theta)) \sin(n\theta) \sin(\theta), \quad (7)$$

where  $\theta \in [0, 2\pi]$ ,  $a = b = 0.40178$ ,  $m = 2$  and  $n = 6$ . The coefficients  $\kappa^\pm$  and the solutions  $u^\pm$  are given as follows

$$\kappa = \begin{cases} \frac{2+xy}{5}, & (x, y) \in \Omega^+, \\ \frac{x^2-y^2+3}{7}, & (x, y) \in \Omega^-, \end{cases} \quad u = \begin{cases} x + y + 1, & (x, y) \in \Omega^+, \\ \sin(x + y) + \cos(x + y) + 1, & (x, y) \in \Omega^-. \end{cases} \quad (8)$$

A dual neural network with 3 layers and 20 neurons in each layer was used. The geometry of interface  $\Gamma$  and subdomains is illustrated in Figure 2a. The uniform sampled points with  $N_{r1} = 500$  on the domain  $\Omega_1$ ,  $N_{r2} = 300$  on the domain  $\Omega_2$  and  $N_\Gamma = 300$  on the interface  $\Gamma$ , and  $N_{\partial\Omega} = 800$  on the boundary  $\partial\Omega$ , were used to calculate the numerical solution, see Figure 2b. The balance weights in the loss function were chosen as  $\omega_1 = 0.00001$ ,  $\omega_2 = 0.00001$ ,  $\omega_3 = 0.99999$ ,  $\omega_4 = 0.99999$ . The PINN solution is shown in Figure 3a. The error between PINN and exact solutions is shown in Figure 3b. It can be seen that the presented PINN was able to approximate the solution of the interface problem, and the relative error between the numerical and the analytical solution was between  $-4.0 \times 10^{-3}$  and  $1.0 \times 10^{-3}$ . Figure 4 presents the evolution process of the loss function with adaptive and fixed activation functions for Sub-Net1 and Sub-Net2, respectively. It is shown that the adaptive activation function was beneficial in improving the convergence speed of the loss function in the process of deep learning.



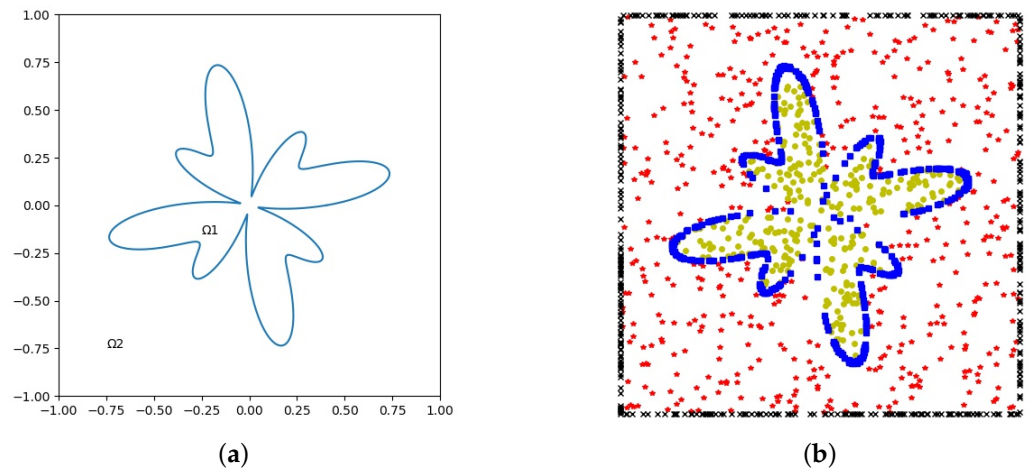


Figure 2. (a) Computational domain and interface geometry and (b) training points.

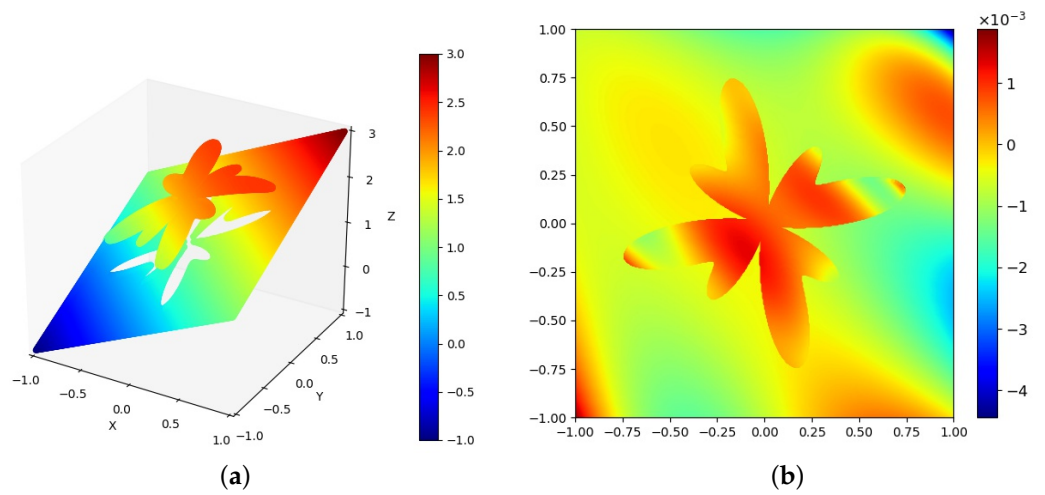


Figure 3. (a) The PINN solution and (b) the error between PINN and exact solutions for Example 1.

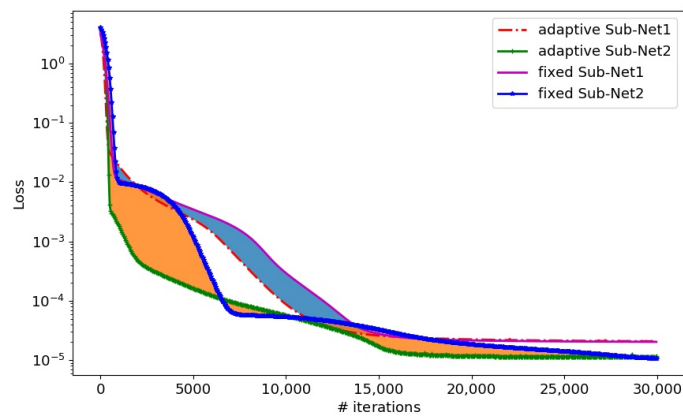


Figure 4. The evolution of loss error for Example 1.

Table 1 shows the relative  $L_2$  errors with the different number of training points under the adaptive activation function and fixed activation function.  $e_{\Omega_1}$ ,  $e_{\Omega_2}$ ,  $e_{\Gamma}$  and  $e_{\partial\Omega}$  are the loss errors at  $\Omega^+$ ,  $\Omega^-$ , interface  $\Gamma$  and boundary  $\partial\Omega$ , respectively. It can be seen that the error decreased with increase in the number of training points on the boundary and interface, and the error by the adaptive activation function method was better than that of the fixed activation function. Table 2 presents the relative  $L_2$  errors of the same training points with different numbers of neurons. In this table, we can see that the calculation accuracy increased with increase in the number of neurons, but the calculation time cost increased accordingly. Therefore, the appropriate network depth and width should be selected in order to acquire good accuracy and efficiency.

**Table 1.** Comparison of the errors of the adaptive and the fixed activating function strategies for different numbers of training points for Example 1.

Activation Function	$N_{\partial\Omega}/N_{r1}/N_{r2}/N_{\Gamma}$	$e_{\Omega_1}$	$e_{\Omega_2}$	$e_{\Gamma}$	$e_{\partial\Omega}$
Adaptive	200/500/300/75	$9.997 \times 10^{-4}$	$6.553 \times 10^{-4}$	$6.621 \times 10^{-4}$	$7.786 \times 10^{-4}$
	400/500/300/150	$7.223 \times 10^{-4}$	$4.552 \times 10^{-4}$	$2.800 \times 10^{-4}$	$5.483 \times 10^{-4}$
	800/500/300/300	$4.539 \times 10^{-4}$	$3.675 \times 10^{-4}$	$3.319 \times 10^{-4}$	$5.771 \times 10^{-4}$
Fixed	200/500/300/75	$5.137 \times 10^{-4}$	$7.497 \times 10^{-4}$	$6.797 \times 10^{-4}$	$5.013 \times 10^{-4}$
	400/500/300/150	$4.997 \times 10^{-4}$	$6.513 \times 10^{-4}$	$6.628 \times 10^{-4}$	$5.186 \times 10^{-4}$
	800/500/300/300	$4.560 \times 10^{-4}$	$4.164 \times 10^{-4}$	$3.724 \times 10^{-4}$	$4.843 \times 10^{-4}$

**Table 2.** Comparison of errors in different terms with different numbers of neurons for Example 1.

Number of Neurons	$e_{\Omega_1}$	$e_{\Omega_2}$	$e_{\Gamma}$	$e_{\partial\Omega}$
10	$1.007 \times 10^{-3}$	$3.101 \times 10^{-4}$	$3.336 \times 10^{-4}$	$5.140 \times 10^{-4}$
20	$5.397 \times 10^{-4}$	$4.652 \times 10^{-4}$	$3.488 \times 10^{-4}$	$6.003 \times 10^{-4}$
40	$2.912 \times 10^{-4}$	$1.573 \times 10^{-4}$	$1.208 \times 10^{-4}$	$1.967 \times 10^{-4}$

**Example 2.** This example used the same interface geometric parameter equation as in Example 1. The same problem setting and analysis solution as in the previous example and the balance weight in the loss function were used. Here,  $a = 0.50012563$ ,  $b = 0.250012563$ ,  $m = 0$ , and  $n = 12$ . The geometry of interface  $\Gamma$  and subdomains is illustrated in Figure 5a. The numerical solution was calculated on the uniform sampled points with  $N_{r1} = 500$ ,  $N_{r2} = 300$ ,  $N_{\Gamma} = 300$ , and  $N_{\partial\Omega} = 800$ , see Figure 5b. The PINN solution is shown in Figure 6a. The relative error in  $L_2$  norm is plotted in Figure 6. It can be seen that the present method effectively approximated the solution of the interface problem, and the range of error was between  $-3 \times 10^{-3}$  to  $3 \times 10^{-3}$ . This example shows that, even at uniform sampling points, the present dual neural network technology could produce sufficient accurate approximation for two-dimensional elliptic problems with complex interface geometry. Figure 7 displays the evolution process of the loss function with adaptive and fixed activation functions for Sub-Net1 and Sub-Net2, respectively. It is shown that the adaptive activation function was beneficial in improving the convergence speed of the loss function in the process of deep learning.



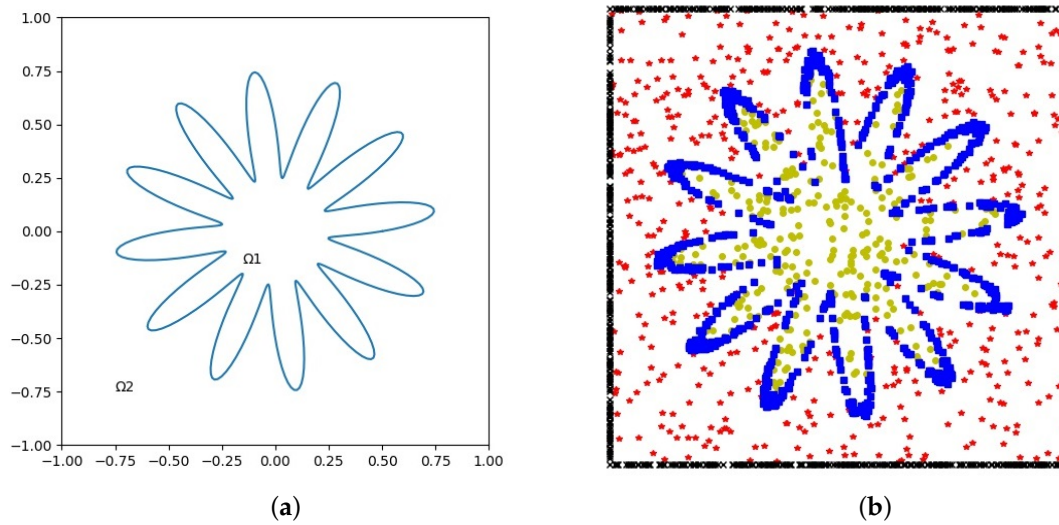


Figure 5. (a) Domain and interface geometry and (b) training points.

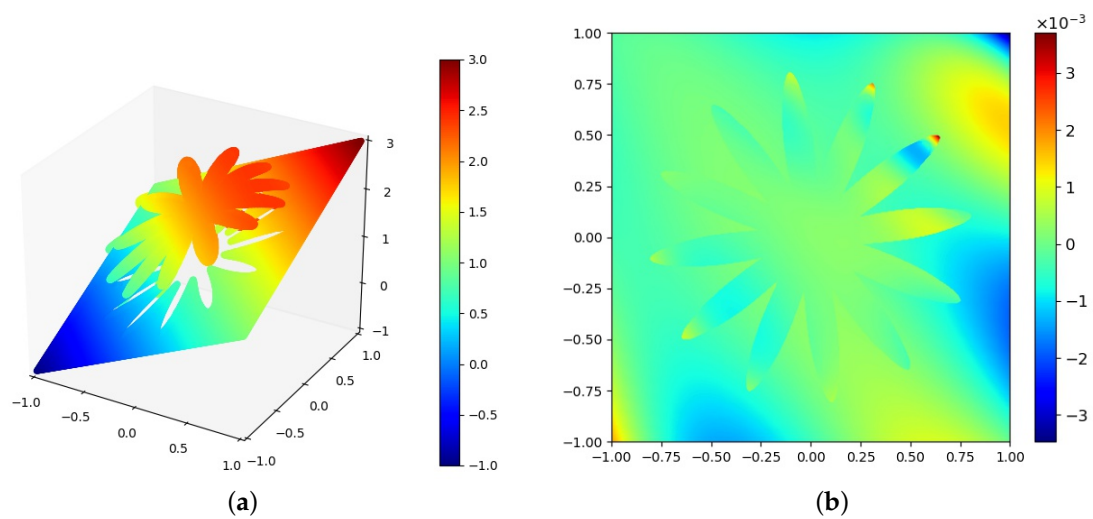


Figure 6. (a) The PINN solution and (b) the error between PINN and the exact solution for Example 2.

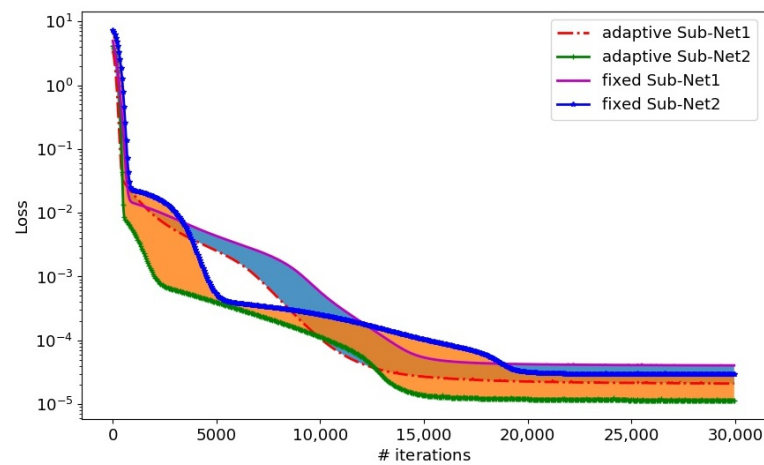


Figure 7. The evolution of average loss error for Example 2.

Table 3 compares the relative  $L_2$  errors in a different number of training points with adaptive and fixed activation functions, respectively. We can see that the error decreased with increase in the number of training points on the boundary and interface, and the error of the adaptive activation function method was better than that of the fixed activation function. Table 4 shows the relative  $L_2$  errors of the same training points with a different number of neurons. It is shown that the calculation accuracy increased with increase of the number of neurons, but the calculation time cost increased accordingly.

**Table 3.** Comparison of errors by adaptive and fixed activation function strategies with different numbers of training points for Example 2.

Activation Function	$N_{\partial\Omega}/N_{r1}/N_{r2}/N_{\Gamma}$	$e_{\Omega_1}$	$e_{\Omega_2}$	$e_{\Gamma}$	$e_{\partial\Omega}$
Adaptive	200/500/300/75	$4.517 \times 10^{-4}$	$3.827 \times 10^{-4}$	$6.621 \times 10^{-4}$	$4.786 \times 10^{-4}$
	400/500/300/150	$4.213 \times 10^{-4}$	$1.653 \times 10^{-4}$	$2.800 \times 10^{-4}$	$5.483 \times 10^{-4}$
	800/500/300/300	$4.167 \times 10^{-4}$	$1.719 \times 10^{-4}$	$2.409 \times 10^{-4}$	$4.691 \times 10^{-4}$
Fixed	200/500/300/75	$8.257 \times 10^{-4}$	$7.546 \times 10^{-4}$	$6.348 \times 10^{-4}$	$7.221 \times 10^{-4}$
	400/500/300/150	$6.458 \times 10^{-4}$	$6.114 \times 10^{-4}$	$6.227 \times 10^{-4}$	$7.102 \times 10^{-4}$
	800/500/300/300	$4.139 \times 10^{-4}$	$3.449 \times 10^{-4}$	$4.677 \times 10^{-4}$	$5.677 \times 10^{-4}$

**Table 4.** Comparison of errors in different terms with different numbers of neurons for Example 2.

Number of Neurons	$e_{\Omega_1}$	$e_{\Omega_2}$	$e_{\Gamma}$	$e_{\partial\Omega}$
10	$7.108 \times 10^{-4}$	$3.548 \times 10^{-4}$	$5.548 \times 10^{-4}$	$4.591 \times 10^{-4}$
20	$4.517 \times 10^{-4}$	$3.827 \times 10^{-4}$	$6.621 \times 10^{-4}$	$4.786 \times 10^{-4}$
40	$4.559 \times 10^{-4}$	$7.968 \times 10^{-4}$	$8.676 \times 10^{-4}$	$3.454 \times 10^{-4}$

**Example 3.** Consider the following level set function to describe an annular region with inner and outer radii  $r_{inn} = 0.151$  and  $r_{out} = 0.911$  with an immersed star interface.

$$\phi(x, y) = \sqrt{(x^2 + y^2)} - r_0 \left( 1 + \sum_{k=1}^3 \beta_k \cos \left( n_k (\arctan(\frac{y}{x}) - \theta_k) \right) \right), \tag{9}$$

with parameters:

$$r_0 = 0.483, \quad \begin{pmatrix} n_1 \\ \beta_1 \\ \theta_1 \end{pmatrix} = \begin{pmatrix} 3 \\ 0.3 \\ 0.5 \end{pmatrix}, \quad \begin{pmatrix} n_2 \\ \beta_2 \\ \theta_2 \end{pmatrix} = \begin{pmatrix} 4 \\ -0.1 \\ 1.8 \end{pmatrix}, \quad \text{and} \quad \begin{pmatrix} n_3 \\ \beta_3 \\ \theta_3 \end{pmatrix} = \begin{pmatrix} 7 \\ 0.15 \\ 0 \end{pmatrix}. \tag{10}$$

The exact solution is

$$u = \begin{cases} \left( 16 \left( \frac{y-x}{3} \right)^5 - 20 \left( \frac{y-x}{3} \right)^3 \right) + 5 \left( \frac{y-x}{3} \right) \log(x + y + 3), & (x, y) \in \Omega^+, \\ \sin(2x) \cos(2y), & (x, y) \in \Omega^-. \end{cases} \tag{11}$$

We tested the problems on dual neural networks. Each neural network had 3 layers and each layer had 20 neurons. The geometry of interface  $\Gamma$  and subdomains is illustrated in Figure 8a. The numerical solution was calculated on the uniform sampled points with  $N_{r1} = 500$ ,  $N_{r2} = 300$ ,  $N_{\Gamma} = 300$ , and  $N_{b1} = 200$ ,  $N_{b2} = 800$ , see Figure 8b. The PINN solution is shown in Figure 9a. The errors between PINN and the exact solutions is shown in Figure 9b. It can be seen that the presented method effectively approximated the solution of the interface problem, and the range of error was from  $-2.0 \times 10^{-3}$  to  $6.0 \times 10^{-3}$ . Figure 10 shows the evolution process of the loss function with adaptive and fixed activation functions for Sub-Net1 and Sub-Net2, respectively.

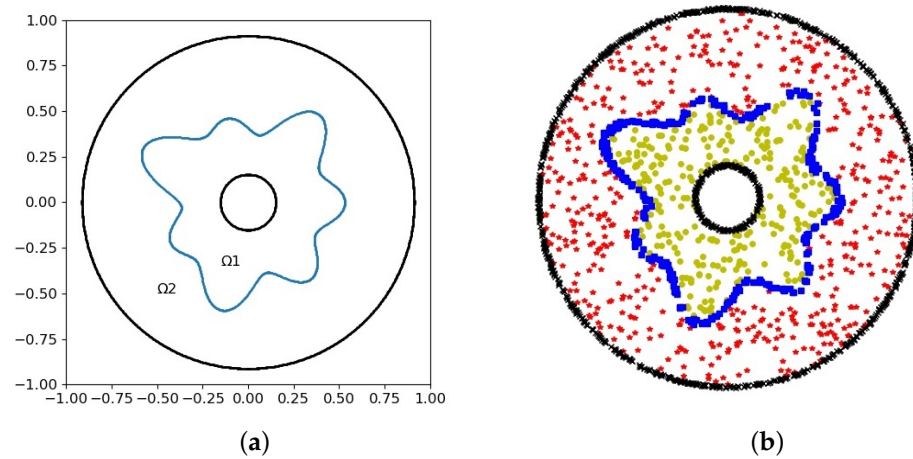


Figure 8. (a) Problem geometry and (b) training points.

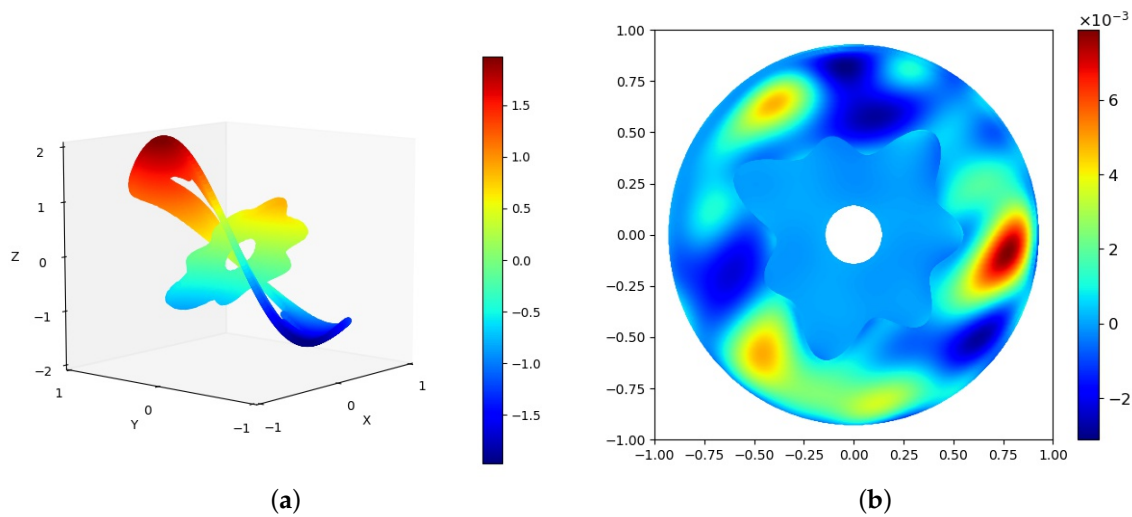


Figure 9. (a) The PINN solution and (b) the error between PINN and exact solutions for Exampe 3.

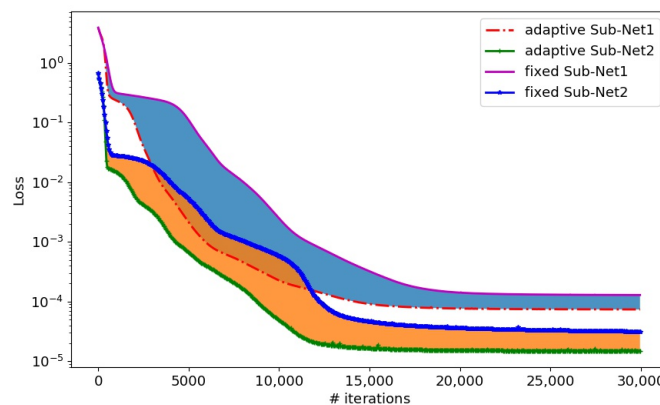


Figure 10. The evolution plot of loss  $L_2$  error for Example 3.

Table 5 compares the relative  $L_2$  errors with a different number of training points under adaptive and fixed activation functions. We can see that the error decreased with increase in the number of training points on the boundary and interface, and the error of the adaptive activation function method was better than that of the fixed activation function.  $e_{\partial\Omega_1}$  and  $e_{\partial\Omega_2}$  were the loss errors on the internal and the outer boundary, respectively.

Table 6 illustrates the relative  $L_2$  errors of the same training points with a different number of neurons. It is shown that the calculation accuracy increased with increase in the number of neurons.

**Table 5.** Comparison of errors by adaptive and fixed activation function strategies with different numbers of training points for Example 3.

Activation Function	$N_{\partial\Omega_1}/N_{\partial\Omega_2}/N_{r1}/N_{r2}/N_{\Gamma}$	$e_{\Omega_1}$	$e_{\Omega_2}$	$e_{\Gamma}$	$e_{\partial\Omega_1}$	$e_{\partial\Omega_2}$
Adaptive	200/50/500/300/75	$2.870 \times 10^{-3}$	$3.765 \times 10^{-4}$	$6.621 \times 10^{-4}$	$3.138 \times 10^{-4}$	$7.113 \times 10^{-4}$
	400/100/500/300/150	$1.897 \times 10^{-3}$	$2.103 \times 10^{-4}$	$7.083 \times 10^{-4}$	$3.203 \times 10^{-4}$	$4.120 \times 10^{-4}$
	800/200/500/300/300	$1.475 \times 10^{-3}$	$2.403 \times 10^{-4}$	$2.853 \times 10^{-4}$	$3.608 \times 10^{-4}$	$2.001 \times 10^{-4}$
Fixed	200/50/500/300/75	$5.894 \times 10^{-3}$	$5.556 \times 10^{-4}$	$6.790 \times 10^{-4}$	$6.185 \times 10^{-4}$	$5.327 \times 10^{-4}$
	400/100/500/300/150	$5.501 \times 10^{-3}$	$4.778 \times 10^{-4}$	$5.025 \times 10^{-4}$	$5.110 \times 10^{-4}$	$6.311 \times 10^{-4}$
	800/200/500/300/300	$4.540 \times 10^{-3}$	$3.633 \times 10^{-4}$	$5.515 \times 10^{-4}$	$5.904 \times 10^{-4}$	$5.522 \times 10^{-4}$

**Table 6.** Comparison of errors in different terms with different numbers of neurons for Example 3.

Number of Neurons	$e_{\Omega_1}$	$e_{\Omega_2}$	$e_{\Gamma}$	$e_{\partial\Omega_1}$	$e_{\partial\Omega_2}$
10	$4.754 \times 10^{-3}$	$9.017 \times 10^{-4}$	$1.550 \times 10^{-3}$	$7.443 \times 10^{-4}$	$1.134 \times 10^{-3}$
20	$2.870 \times 10^{-3}$	$3.765 \times 10^{-4}$	$6.621 \times 10^{-4}$	$3.138 \times 10^{-4}$	$7.113 \times 10^{-4}$
40	$2.282 \times 10^{-3}$	$4.866 \times 10^{-4}$	$8.330 \times 10^{-4}$	$2.915 \times 10^{-4}$	$3.276 \times 10^{-4}$

**Example 4.** A two-dimensional elliptic problem with irregular interface geometry.

We tested an interface problem with variable coefficients, and the computational domain is shown in Figure 11. The boundary is given in polar coordinates  $r = 1.5 + 0.14 \sin(4\theta) + 0.12 \cos(6\theta) + 0.09 \cos(5\theta)$ , where  $\theta \in [0, 2\pi)$ , and the boundary points are obtained as  $x_1 = r \cos(\theta)$  and  $x_2 = r \sin(\theta)$ . The computational domain is further divided into two highly irregular, non-convex subdomains, where the interface is given by  $r = 0.6 + 0.216 \sin(3\theta) + 0.096 \cos(2\theta) + 0.24 \cos(5\theta)$  and the corresponding interface points are obtained as  $x_1 = r \cos(\theta)$  and  $x_2 = r \sin(\theta)$ . The coefficient  $\kappa^{\pm}$  is defined as

$$\kappa(x, y) = \begin{cases} xy, & (x, y) \in \Omega^+, \\ x^2 + y^2, & (x, y) \in \Omega^-. \end{cases} \tag{12}$$

The exact solution is

$$u(x, y) = \begin{cases} \sin(x + y), & (x, y) \in \Omega^+, \\ \cos(x + y), & (x, y) \in \Omega^-. \end{cases} \tag{13}$$

The necessary forcing term  $f$ , boundary and interface conditions are divided from the exact solution.

The geometry of interface  $\Gamma$  and subdomains is illustrated in Figure 11a. The numerical solution was calculated on the uniform sampled points with  $N_{r1} = 500$ ,  $N_{r2} = 300$ ,  $N_{\Gamma} = 300$ , and  $N_{\partial\Omega} = 800$ , see Figure 11b. The balance weights in this example were chosen as  $\omega_1 = 0.99999$ ,  $\omega_2 = 0.00001$ ,  $\omega_3 = 0.99999$ ,  $\omega_4 = 0.99999$ ,  $\omega_5 = 0$ . The PINN solution is shown in Figure 12a. The error between PINN and exact solutions was between  $-1.5 \times 10^{-3}$  and  $2.5 \times 10^{-3}$ , as shown in Figure 12b. Figure 13 shows the evolution process of the loss function with adaptive and fixed activation functions for Sub-Net1 and Sub-Net2, respectively.

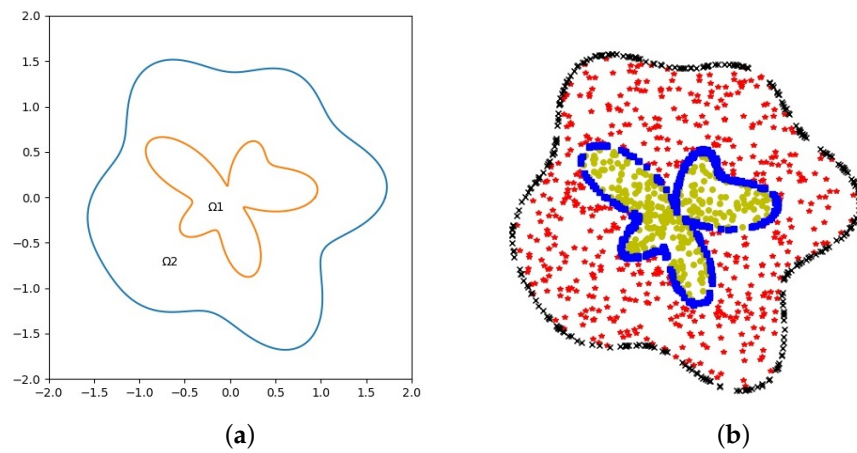


Figure 11. (a) Computational domain and interface shape and (b) training points.

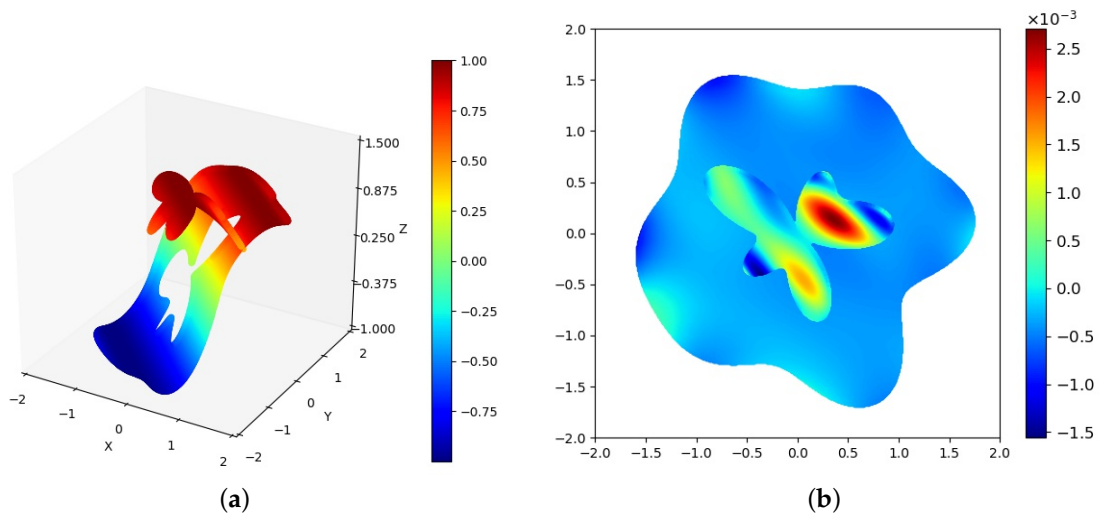


Figure 12. (a) The PINN solution and (b) the error between PINN and the exact solution for Example 4.

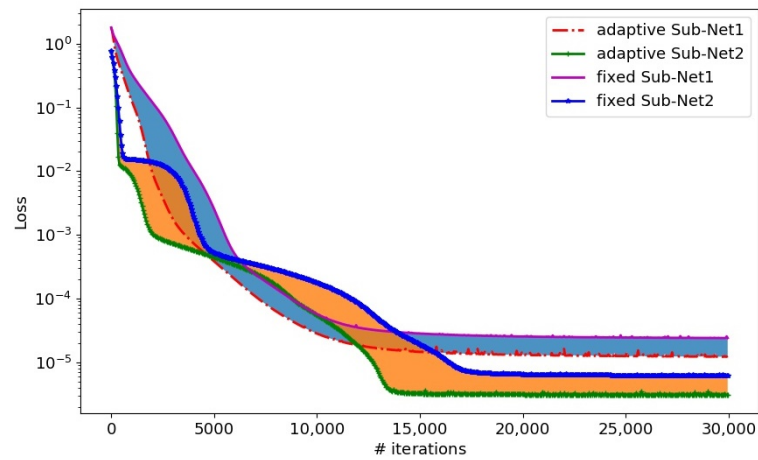


Figure 13. The plot of  $L_s$  average error loss for Example 4.

Table 7 compares the relative  $L_2$  errors with different numbers of training points under adaptive activation function and fixed activation function. We can see that the error decreased with increase in the number of training points on the boundary and interface,



and the error of the adaptive activation function method was better than that of the fixed activation function. Table 8 illustrates the relative  $L_2$  errors of the same training points with a different number of neurons. It is shown that the calculation accuracy decreased with increase in the number of neurons, but the calculation time cost increased accordingly.

**Table 7.** Comparison of errors by adaptive and fixed activation function strategies with different numbers of training points for Example 4.

Activation Function	$N_{\partial\Omega}/N_{r1}/N_{r2}/N_{\Gamma}$	$e_{\Omega_1}$	$e_{\Omega_2}$	$e_{\Gamma}$	$e_{\partial\Omega}$
Adaptive	200/500/300/75	$1.933 \times 10^{-4}$	$5.989 \times 10^{-4}$	$5.774 \times 10^{-4}$	$3.591 \times 10^{-4}$
	400/500/300/150	$1.733 \times 10^{-4}$	$3.367 \times 10^{-4}$	$8.803 \times 10^{-4}$	$7.697 \times 10^{-4}$
	800/500/300/300	$1.443 \times 10^{-4}$	$1.102 \times 10^{-4}$	$7.815 \times 10^{-4}$	$2.587 \times 10^{-4}$
Fixed	200/500/300/75	$2.866 \times 10^{-4}$	$6.241 \times 10^{-4}$	$7.144 \times 10^{-4}$	$5.460 \times 10^{-4}$
	400/500/300/150	$3.221 \times 10^{-4}$	$6.001 \times 10^{-4}$	$6.456 \times 10^{-4}$	$4.564 \times 10^{-4}$
	800/500/300/300	$2.754 \times 10^{-4}$	$7.596 \times 10^{-4}$	$6.843 \times 10^{-4}$	$3.219 \times 10^{-4}$

**Table 8.** Comparison of errors in different terms with different numbers of neurons for Example 4.

Number of Neurons	$e_{\Omega_1}$	$e_{\Omega_2}$	$e_{\Gamma}$	$e_{\partial\Omega}$
10	$3.011 \times 10^{-4}$	$7.143 \times 10^{-4}$	$9.133 \times 10^{-4}$	$5.119 \times 10^{-4}$
20	$1.933 \times 10^{-4}$	$5.989 \times 10^{-4}$	$5.774 \times 10^{-4}$	$3.591 \times 10^{-4}$
40	$2.278 \times 10^{-4}$	$1.473 \times 10^{-4}$	$1.284 \times 10^{-4}$	$4.471 \times 10^{-4}$

**Example 5.** Constant gradient jump over a flat interface.

We also tested a more complicated three-dimensional cube interface problem. Along the  $z$ -axis, there were two zones that made up the cube. The conductivity values of the bottom and top sections were 1 and 8 W/m·K, respectively, and the material contact was planar. The prescribed temperatures were 5 and 10 °C along the top and bottom surfaces, respectively, while the remaining surfaces were insulated. Figure 14 depicts the problem’s size and boundary conditions.

With the origin of the Cartesian coordinates system located at the lower front corner of the cubic domain, a distributed heat source  $f(x, y, z) = 5000(z^2 - z + 1)$  was applied, yielding the following exact temperature field in the domain:

$$u(x, y, z) = \begin{cases} -33750z^4 + 833.3z^3 - 2500z^2 + 70.9z + 10, & z < 0.05, \\ -4218.7z^4 + 104.2z^3 - 312.5z^2 + 8.86z + 7.56, & z \geq 0.05. \end{cases} \tag{14}$$

We adopted a dual neural network structure, which included two neural networks. Each network had 3 layers, and each layer had 20 neurons. The numerical solution was calculated on the uniform sampled points with  $N_{r1} = 2000$ ,  $N_{r2} = 2000$ ,  $N_{\Gamma} = 2000$ , and  $N_{\partial\Omega_1} = 3200$ ,  $N_{\partial\Omega_2} = 3200$ . Figure 15a shows the PINN solution. The balance weights in this problem were chosen as  $\omega_1 = 0.00001$ ,  $\omega_2 = 0.99999$ ,  $\omega_3 = 0.99999$ ,  $\omega_4 = 0.99999$ ,  $\omega_5 = 0.99999$ . The error between PINN and exact solutions is shown in Figure 15b and the error range was between  $-5.0 \times 10^{-3}$  and  $1.5 \times 10^{-2}$ . This method was shown to accurately approximate the solution of three-dimensional interface problems. Figure 16 shows the evolution process of the loss function with adaptive and fixed activation functions for Sub-Net1 and Sub-Net2, respectively. The effectiveness of the presented algorithm for complex three-dimensional problems on uniform sampling points can be demonstrated by this example.



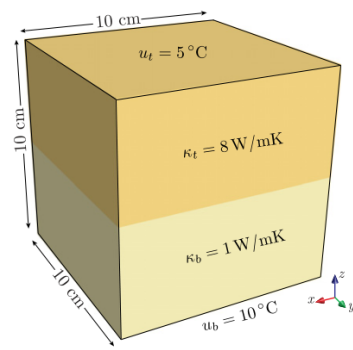


Figure 14. Physical model and interface geometry.

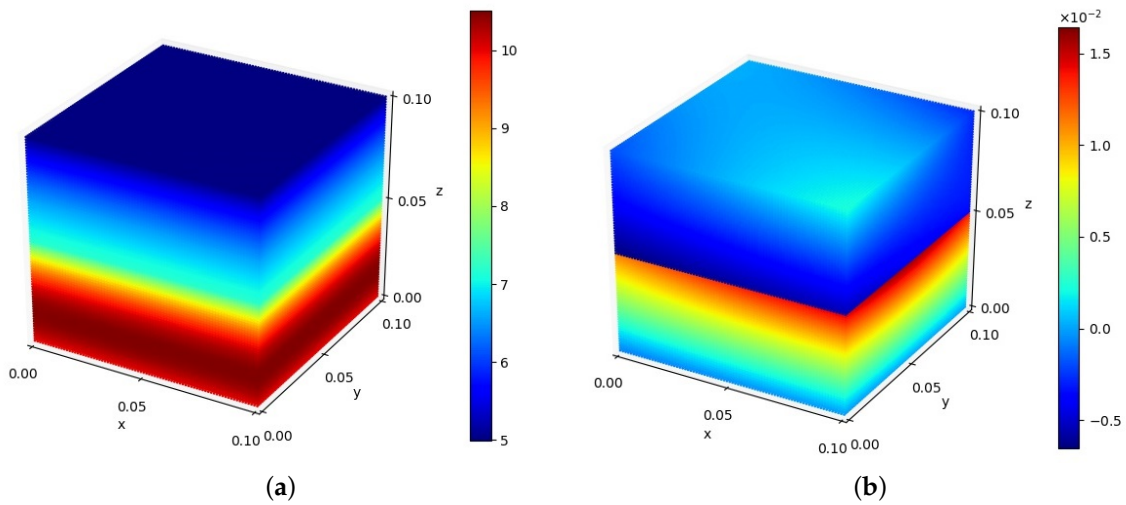


Figure 15. (a) The PINN solution and (b) the error between PINN and exact solution for Example 5.

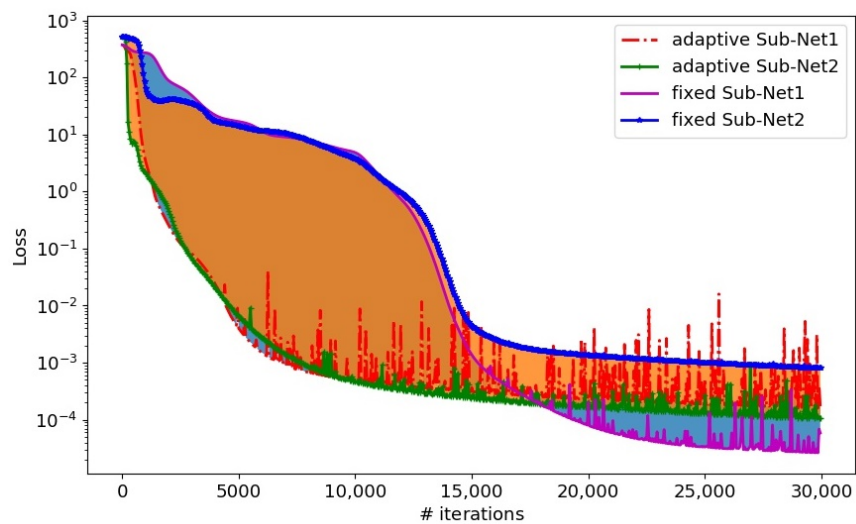


Figure 16. The plot of  $L_s$  average error loss for Example 5.

Table 9 compares the relative  $L_2$  errors with different numbers of training points under the adaptive activation function and the fixed activation function. We can see that the error decreased with increase in the number of training points on the boundary and interface, and the error of the adaptive activation function method was better than that of the fixed activation function. Table 10 illustrates the relative  $L_2$  errors of the same training points

with a different number of neurons. It was shown that the calculation accuracy decreased with increase in the number of neurons, but the calculation time cost increased accordingly.

**Table 9.** Comparison of errors by adaptive and fixed activation function strategies with different numbers of training points for Example 5.

Activation Function	$N_{\partial\Omega_1}/N_{\partial\Omega_2}/N_{r1}/N_{r2}/N_{\Gamma}$	$e_{\Omega_1}$	$e_{\Omega_2}$	$e_{\Gamma}$	$e_{\partial\Omega_1}$	$e_{\partial\Omega_2}$
Adaptive	800/800/2000/2000/500	$6.781 \times 10^{-4}$	$1.612 \times 10^{-3}$	$1.711 \times 10^{-3}$	$5.117 \times 10^{-4}$	$9.959 \times 10^{-4}$
	1600/1600/2000/2000/1000	$5.353 \times 10^{-4}$	$1.144 \times 10^{-3}$	$1.555 \times 10^{-3}$	$4.531 \times 10^{-4}$	$6.983 \times 10^{-4}$
	3200/3200/2000/2000/2000	$6.731 \times 10^{-4}$	$1.040 \times 10^{-3}$	$1.415 \times 10^{-3}$	$3.375 \times 10^{-4}$	$7.691 \times 10^{-4}$
Fixed	800/800/2000/2000/500	$1.457 \times 10^{-3}$	$4.612 \times 10^{-3}$	$4.556 \times 10^{-3}$	$9.456 \times 10^{-4}$	$1.528 \times 10^{-3}$
	1600/1600/2000/2000/1000	$9.458 \times 10^{-4}$	$2.784 \times 10^{-3}$	$3.455 \times 10^{-3}$	$7.781 \times 10^{-4}$	$7.546 \times 10^{-4}$
	3200/3200/2000/2000/2000	$8.456 \times 10^{-4}$	$7.458 \times 10^{-4}$	$2.785 \times 10^{-3}$	$3.441 \times 10^{-4}$	$6.745 \times 10^{-4}$

**Table 10.** Comparison of errors in different terms with different numbers of neurons for Example 5.

Number of Neurons	$e_{\Omega_1}$	$e_{\Omega_2}$	$e_{\Gamma}$	$e_{\partial\Omega_1}$	$e_{\partial\Omega_2}$
10	$1.323 \times 10^{-3}$	$4.117 \times 10^{-3}$	$3.213 \times 10^{-3}$	$9.094 \times 10^{-4}$	$2.362 \times 10^{-3}$
20	$6.781 \times 10^{-4}$	$1.612 \times 10^{-3}$	$1.711 \times 10^{-3}$	$5.117 \times 10^{-4}$	$9.959 \times 10^{-4}$
40	$4.935 \times 10^{-4}$	$8.247 \times 10^{-4}$	$1.203 \times 10^{-3}$	$3.752 \times 10^{-4}$	$4.445 \times 10^{-4}$

**Example 6.** Three-dimensional sphere with a complicated inner region.

Consider a three-dimensional spherical shell with internal and exterior radii of  $r_{int} = 0.151$  and  $r_{out} = 0.911$ , respectively, in which there is an immersed-type complex star interface and the interface geometry is given by the level set function shown below.

$$\phi(x, y, z) = \sqrt{(x^2 + y^2 + z^2)} - r_0 \left( 1 + \left( \frac{x^2 + y^2}{x^2 + y^2 + z^2} \right)^2 \sum_{k=1}^3 \beta_k \cos \left( n_k \left( \arctan \left( \frac{y}{x} \right) - \theta_k \right) \right) \right), \quad (15)$$

The exact solution is taken as follows with the same parameters (10) as for the two-dimensional case.

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

The problem geometry is illustrated in Figure 17.

The numerical solution is calculated on the uniform sampled points with  $N_{r1} = 500$ ,  $N_{r2} = 300$ ,  $N_{\Gamma} = 600$ , and  $N_{\partial\Omega_1} = 400$ ,  $N_{\partial\Omega_2} = 400$ . The balance weights in the loss function were chosen as  $\omega_1 = 0.00001$ ,  $\omega_2 = 0.99999$ ,  $\omega_3 = 0.99999$ ,  $\omega_4 = 0.99999$ ,  $\omega_5 = 0.99999$ .

Figure 18a depicts the PINN solution on the domain inside the interface. The error between PINN and the exact solutions in the internal domain is shown in Figure 18b. The relative error range of the internal domain was between  $-2.0 \times 10^{-3}$  and  $1.5 \times 10^{-3}$ . Figure 19a depicted the PINN solution on the domain outside the interface. The error between PINN and the exact solutions in the external domain is shown in Figure 19b. It can be seen that the relative error range of the external domain was between  $-1.0 \times 10^{-2}$  and  $1.0 \times 10^{-2}$ . Figure 20 shows the evolution process of the loss function with adaptive and fixed activation functions for Sub-Net1 and Sub-Net2, respectively.

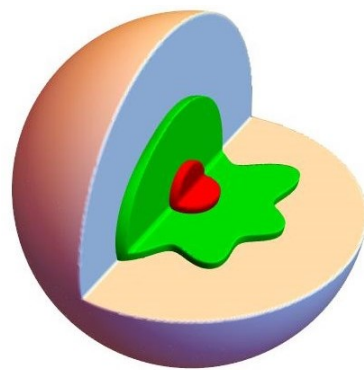


Figure 17. Computational geometry and interface.

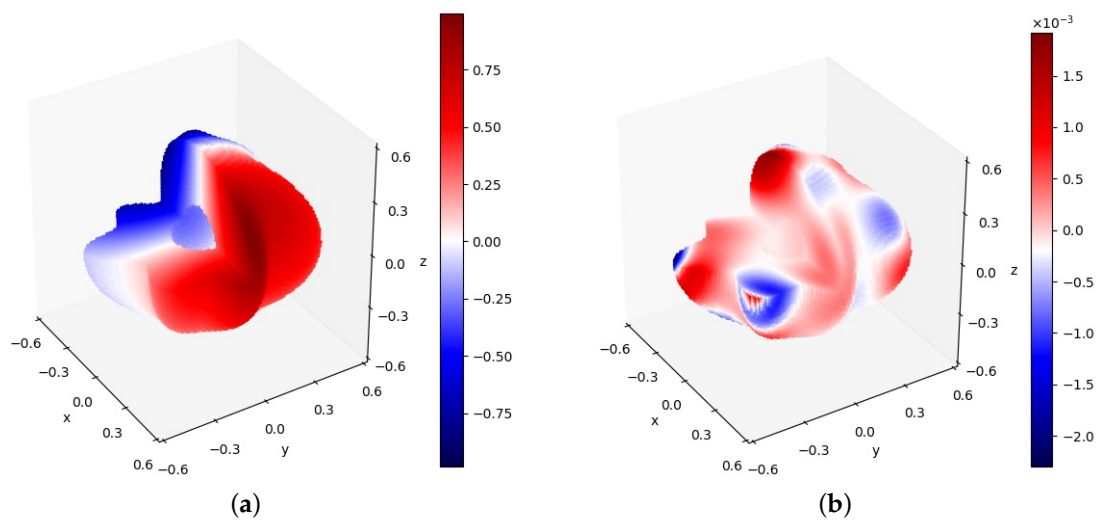


Figure 18. (a) The PINN solution and (b) the error between PINN and the exact solution for internal domain.

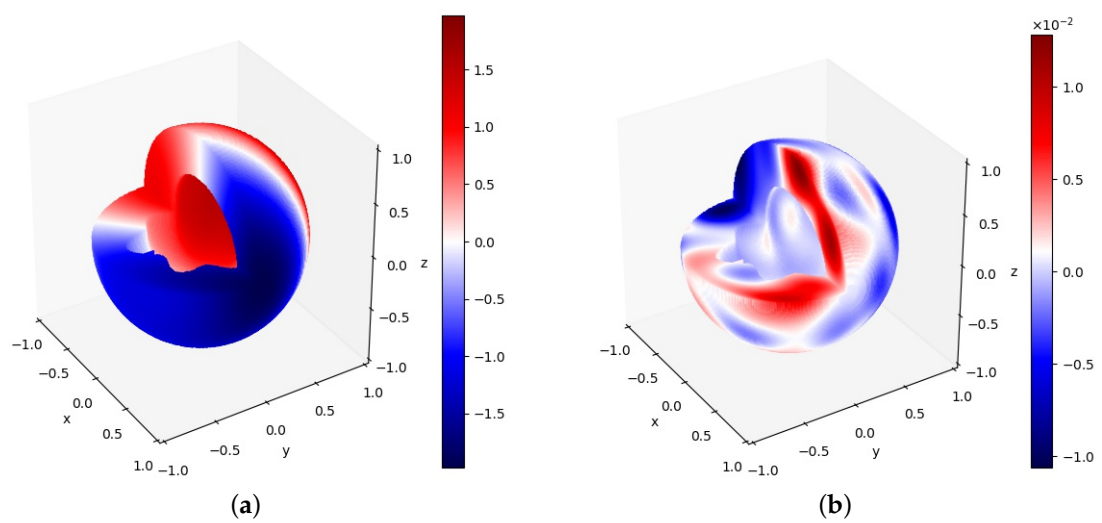


Figure 19. (a) The PINN solution and (b) the error between PINN and exact solution for external domain.

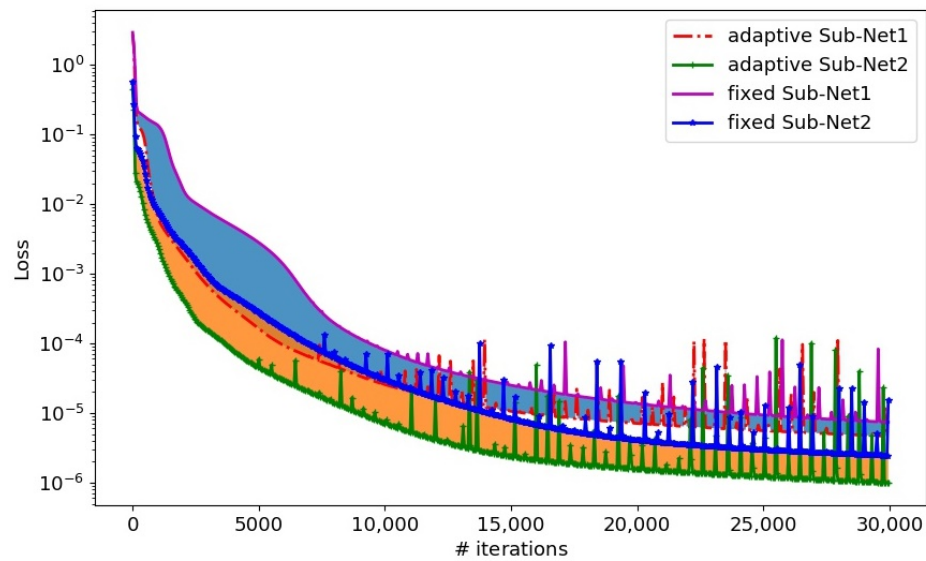


Figure 20. The evolution of  $L_S$  average error loss for Example 6.

Table 11 compares the relative  $L_2$  errors with different numbers of training points under the adaptive activation function and the fixed activation function. We can see that the error decreased with increase in the number of training points on the boundary and interface, and the error of the adaptive activation function method was better than that of the fixed activation function. Table 12 illustrates the relative  $L_2$  errors of the same training points with a different number of neurons. It is shown that the calculation accuracy decreased with increase in the number of neurons, but the calculation time cost increased accordingly.

Table 11. Comparison of errors by adaptive and fixed activation function strategies with different numbers of training points for Example 6.

Activation Function	$N_{\partial\Omega_1}/N_{\partial\Omega_2}/N_{r1}/N_{r2}/N_{\Gamma}$	$e_{\Omega_1}$	$e_{\Omega_2}$	$e_{\Gamma}$	$e_{\partial\Omega_1}$	$e_{\partial\Omega_2}$
Adaptive	100/100/500/300/150	$4.851 \times 10^{-3}$	$1.075 \times 10^{-3}$	$1.746 \times 10^{-3}$	$7.861 \times 10^{-3}$	$7.923 \times 10^{-4}$
	200/200/500/300/300	$2.359 \times 10^{-3}$	$6.597 \times 10^{-4}$	$1.036 \times 10^{-3}$	$1.901 \times 10^{-3}$	$4.145 \times 10^{-4}$
	400/400/500/300/600	$1.888 \times 10^{-3}$	$5.012 \times 10^{-4}$	$1.167 \times 10^{-3}$	$1.229 \times 10^{-3}$	$3.762 \times 10^{-4}$
Fixed	100/100/500/300/150	$5.540 \times 10^{-3}$	$2.552 \times 10^{-3}$	$3.290 \times 10^{-3}$	$5.315 \times 10^{-3}$	$2.215 \times 10^{-3}$
	200/200/500/300/300	$4.180 \times 10^{-3}$	$1.576 \times 10^{-3}$	$2.277 \times 10^{-3}$	$3.299 \times 10^{-3}$	$1.561 \times 10^{-3}$
	400/400/500/300/600	$2.902 \times 10^{-3}$	$1.337 \times 10^{-3}$	$2.032 \times 10^{-3}$	$1.692 \times 10^{-3}$	$8.895 \times 10^{-4}$

Table 12. Comparison of errors in different terms with different numbers of neurons for Example 6.

Number of Neurons	$e_{\Omega_1}$	$e_{\Omega_2}$	$e_{\Gamma}$	$e_{\partial\Omega_1}$	$e_{\partial\Omega_2}$
10	$5.394 \times 10^{-3}$	$2.087 \times 10^{-3}$	$3.303 \times 10^{-3}$	$7.094 \times 10^{-3}$	$2.062 \times 10^{-3}$
20	$4.851 \times 10^{-3}$	$1.075 \times 10^{-3}$	$1.746 \times 10^{-3}$	$7.861 \times 10^{-3}$	$7.923 \times 10^{-4}$
40	$2.068 \times 10^{-3}$	$1.226 \times 10^{-3}$	$1.637 \times 10^{-3}$	$2.391 \times 10^{-3}$	$6.592 \times 10^{-4}$

Example 7. Consider a sphere with an inner doughnut, the level set function of the inner doughnut interface is given as:

$$\phi(x, y, z) = (\sqrt{x^2 + y^2} - 0.6)^2 + z^2 - 0.3^2. \tag{17}$$

The elliptic coefficients and exact solution are given as prior:

$$\kappa(x, y, z) = \begin{cases} 80, & (x, y, z) \in \Omega^+, \\ 2, & (x, y, z) \in \Omega^-. \end{cases} \tag{18}$$

$$u = \begin{cases} xy + x^4 + y^4 + xz^2 + \cos(2x + y^2 + z^3), & (x, y, z) \in \Omega^+, \\ x^3 + xy^2 + y^3 + z^4 + \sin(3(x^2 + y^2)), & (x, y, z) \in \Omega^-. \end{cases} \tag{19}$$

In this problem, a dual neural network structure with two neural networks was adopted. Each network had 4 layers, and each layer had 20 neurons. The numerical solution was calculated on the uniform sampled points with  $N_{r_1} = 2000$ ,  $N_{r_2} = 300$ ,  $N_{\Gamma} = 300$ , and  $N_{\partial\Omega} = 2000$ . The balance weights were chosen as  $\omega_1 = 0.24999$ ,  $\omega_2 = 0.00024$ ,  $\omega_3 = 0.49999$ ,  $\omega_4 = 0.24999$ ,  $\omega_5 = 0$ . Figure 21a shows the PINN solution within the interface. The error between PINN and the exact solutions on the internal domain is shown in Figure 21b. It is shown that the relative error range of the internal domain was between  $-4 \times 10^{-2}$  and  $2 \times 10^{-2}$ . Figure 22 plots the PINN solution outside the interface. The errors between PINN and the exact solutions on the external domain is shown in Figure 22b. It can be seen that the relative error range of the external domain was between  $-4 \times 10^{-2}$  and  $2 \times 10^{-2}$ . Figure 23 shows the evolution process of the loss function with adaptive and fixed activation functions for Sub-Net1 and Sub-Net2, respectively.

Table 13 compares the relative  $L_2$  errors with different numbers of training points under the adaptive activation function and the fixed activation function. We can see that the error decreased with increase in the number of training points on the boundary and interface, and the error of the adaptive activation function method was better than that of the fixed activation function. Table 14 illustrates the relative  $L_2$  errors of the same training points with a different number of neurons. It is shown that the calculation accuracy decreased with increase in the number of neurons, while the calculation time cost increased accordingly.

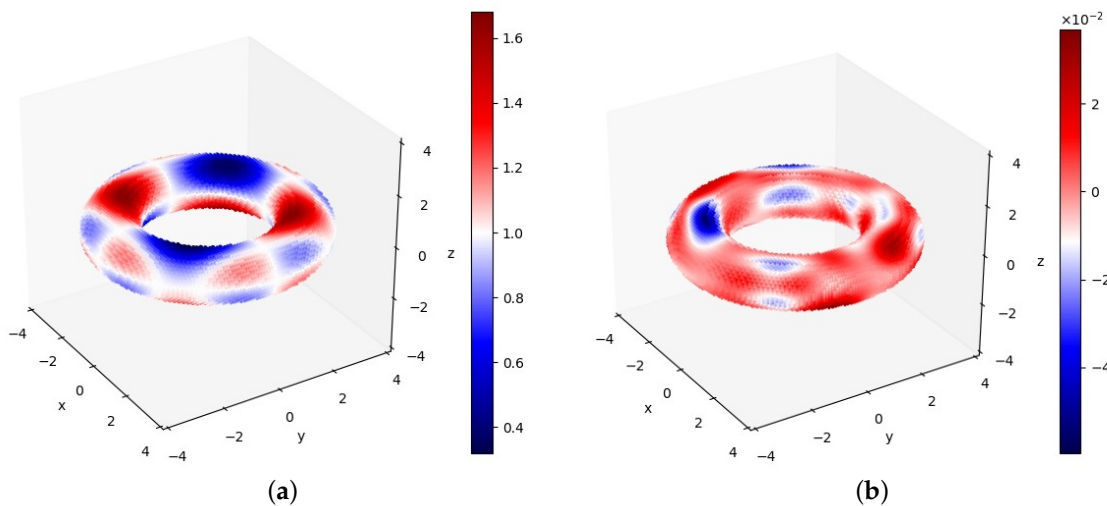


Figure 21. (a) The PINN solution and (b) the error between PINN and the exact solution on external area.

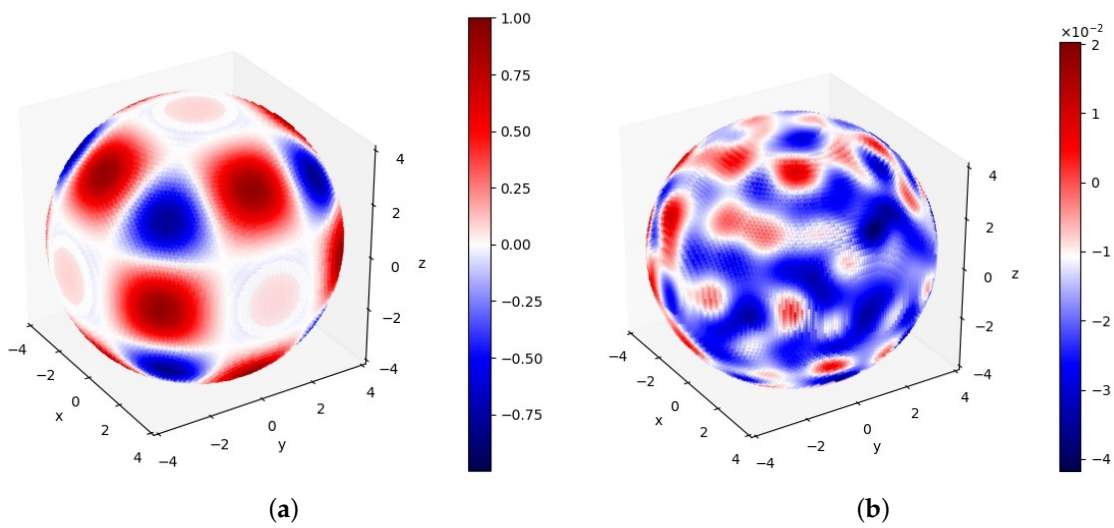


Figure 22. (a) The PINN solution and (b) the error between PINN and the exact solution on internal domain.

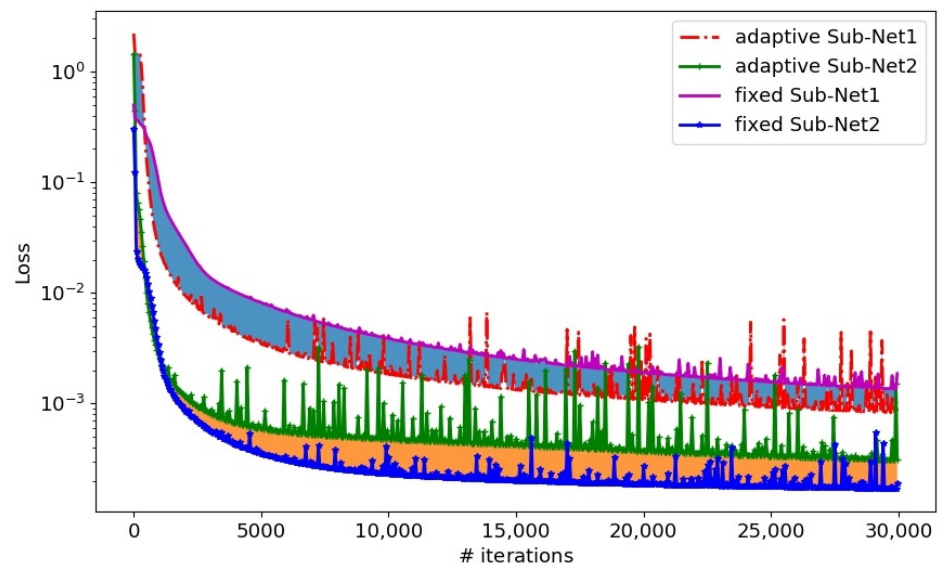


Figure 23. The plot of  $L_s$  average error loss for Example 7.

Table 13. Comparison of errors by adaptive and fixed activation function strategies with different numbers of training points for Example 7.

Activation Functions	$N_{\partial\Omega}/N_1/N_2/N_i$	$e_{\Omega_1}$	$e_{\Omega_2}$	$e_{\Gamma}$	$e_{\partial\Omega}$
Adaptive	500/2000/300/75	$6.022 \times 10^{-2}$	$2.514 \times 10^{-2}$	$3.580 \times 10^{-2}$	$1.011 \times 10^{-1}$
	1000/2000/300/150	$3.742 \times 10^{-2}$	$1.437 \times 10^{-2}$	$2.165 \times 10^{-2}$	$8.032 \times 10^{-2}$
	2000/2000/300/300	$5.330 \times 10^{-2}$	$1.061 \times 10^{-2}$	$1.425 \times 10^{-2}$	$8.363 \times 10^{-2}$
Fixed	500/2000/300/75	$8.459 \times 10^{-2}$	$6.155 \times 10^{-2}$	$4.486 \times 10^{-2}$	$3.785 \times 10^{-1}$
	1000/2000/300/150	$7.489 \times 10^{-2}$	$4.128 \times 10^{-2}$	$2.445 \times 10^{-2}$	$1.567 \times 10^{-1}$
	2000/2000/300/300	$6.446 \times 10^{-2}$	$3.546 \times 10^{-2}$	$3.457 \times 10^{-2}$	$8.458 \times 10^{-2}$



**Table 14.** Comparison of errors in different terms with different numbers of neurons for Example 7.

Number of Neurons	$e_{\Omega_1}$	$e_{\Omega_2}$	$e_{\Gamma}$	$e_{\partial\Omega}$
10	$6.125 \times 10^{-2}$	$3.221 \times 10^{-2}$	$4.208 \times 10^{-2}$	$9.971 \times 10^{-2}$
20	$5.330 \times 10^{-2}$	$1.061 \times 10^{-2}$	$1.425 \times 10^{-2}$	$8.363 \times 10^{-2}$
40	$2.175 \times 10^{-2}$	$7.691 \times 10^{-3}$	$1.228 \times 10^{-2}$	$2.674 \times 10^{-2}$

## 6. Conclusions

A soft constraint physics-informed neural network is proposed to solve nonhomogeneous elliptic interface problems. Due to the complexity of geometry, the solution may change dramatically across the interface. Traditional numerical methods for solving these problems are challenging. As the interface divides the computational domain into two disjoint parts, a dual neural network structure with two separate neural networks in two disjoint subdomains is employed. The two neural networks are joined by jumping conditions on the interface, which are necessary to capture the solution singularity on the interface. The PDEs, boundary conditions and jump condition on the interface are then formulated into the loss function by PINN, and the terms in the loss function balanced by penalty weights. An adaptive activation function method is used to increase the computing efficiency in more difficult situations. This method may be optimized by dynamically changing the topology of the loss function used in the optimization process to attain the optimal network performance. Numerical experiments for 2D and 3D situations demonstrated that the proposed method can solve interface problems with flexibility, effectiveness, and enough accuracy.

**Author Contributions:** Methodology, F.C.; Software, X.G.; Validation, X.G. and F.G.; Writing—review & editing, D.Y. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was partially supported by the National Natural Science Foundation of China (12261067, 12161067, 62201298, 12001015, 51961031), the Inner Mongolia Autonomous Region “Youth Science and Technology Talents” Support Program (NJYT20B15), the Inner Mongolia Scientific Fund Project (2020MS06010, 2021LHMS01006, 2022MS01008), and Innovation Fund Project of Inner Mongolia University of Science and Technology-Excellent Youth Science Fund Project (2019YQL02).

**Conflicts of Interest:** The authors declare that they have no competing interests.

## References

- Oevermann, M.; Scharfenberg, C.; Klein, R. A sharp interface finite volume method for elliptic equations on Cartesian grids. *J. Comput. Phys.* **2009**, *228*, 5184–5206. [[CrossRef](#)]
- Preskill, B.; Sethian, J.A. Jump splicing schemes for elliptic interface problems and the incompressible Navier-Stokes equations. *arXiv* **2016**, arXiv:1612.09342.
- Guyomarc’h, G.; Lee, C.O.; Jeon, K. A discontinuous Galerkin method for elliptic interface problems with application to electroporation. *Commun. Numer. Methods Eng.* **2009**, *25*, 991–1008. [[CrossRef](#)]
- López-Ruiz, G.; Bravo-Castillero, J.; Brenner, R.; Cruz, M.E.; Guinovart-Díaz, R.; Pérez-Fernández, L.D.; Rodríguez-Ramos, R. Variational bounds in composites with nonuniform interfacial thermal resistance. *Appl. Math. Model.* **2015**, *39*, 7266–7276. [[CrossRef](#)]
- Rocha, R.P.A.; Cruz, M.E. Computation of the effective conductivity of unidirectional fibrous composites with an interfacial thermal resistance. *Numer. Heat Transf. Part A Appl.* **2001**, *39*, 179–203. [[CrossRef](#)]
- Costa, R.; Nobrega, J.M.; Clain, S.; Machado, G.J. Very high-order accurate polygonal mesh finite volume scheme for conjugate heat transfer problems with curved interfaces and imperfect contacts. *Comput. Methods Appl. Mech. Eng.* **2019**, *357*, 112560. [[CrossRef](#)]
- Meften, G.A.; Ali, A.H. Continuous dependence for double diffusive convection in a Brinkman model with variable viscosity. *Acta Univ. Sapientiae Math.* **2022**, *14*, 125–146. [[CrossRef](#)]
- Meften, G.A.; Ali, A.H.; Yaseen, M.T. Continuous dependence for thermal convection in a Forchheimer-Brinkman model with variable viscosity. *AIP Conf. Proc.* **2023**, *2457*, 020005.
- Huang, P.; Wu, H.; Xiao, Y. An unfitted interface penalty finite element method for elliptic interface problems. *Comput. Methods Appl. Mech. Eng.* **2017**, *323*, 439–460. [[CrossRef](#)]

10. Barrett, J.; Elliott, C. Fitted and unfitted finite element methods for elliptic equations with smooth interfaces. *IMA J. Numer. Anal.* **1987**, *7*, 283–300. [[CrossRef](#)]
11. Guo, H.; Yang, X. Gradient recovery for elliptic interface problem: I. body-fitted mesh. *arXiv* **2016**, arXiv:1607.05898.
12. Li, J.; Wohlmuth, J.M.J.B.; Zou, J. Optimal a priori estimates for higher order finite elements for elliptic interface problems. *Appl. Numer. Math.* **2010**, *60*, 19–37. [[CrossRef](#)]
13. Zheng, X.; Lowengrub, J. An interface-fitted adaptive mesh method for elliptic problems and its application in free interface problems with surface tension. *Adv. Comput. Math.* **2016**, *42*, 1225–1257. [[CrossRef](#)]
14. Cao, F.; Sheng, Z.; Yuan, G. Monotone finite volume schemes for diffusion equation with imperfect interface on distorted meshes. *J. Sci. Comput.* **2018**, *76*, 1055–1077. [[CrossRef](#)]
15. Peskin, C. Numerical analysis of blood flow in the heart. *J. Comput. Phys.* **1977**, *25*, 220–252. [[CrossRef](#)]
16. Li, Z.; Ito, K. *The Immersed Interface Method: Numerical Solutions of PDEs Involving Interfaces and Irregular Domains*; Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 2006.
17. Ji, H.; Zhang, Q.; Wang, Q.; Xie, Y. A partially penalised immersed finite element method for elliptic interface problems with non-homogeneous jump conditions. *East Asia J. Appl. Math.* **2018**, *8*, 1–23. [[CrossRef](#)]
18. Wu, H.; Xiao, Y. An unfitted hp-interface penalty finite element method for elliptic interface problems. *J. Comput. Math.* **2019**, *37*, 316.
19. Guo, R.; Lin, T. A higher degree immersed finite element method based on a Cauchy extension for elliptic interface problems. *SIAM J. Numer. Anal.* **2019**, *57*, 1545–1573. [[CrossRef](#)]
20. Wang, Q.; Zhang, Z.; Wang, L. New immersed finite volume element method for elliptic interface problems with non-homogeneous jump conditions. *J. Comput. Phys.* **2020**, *427*, 110075. [[CrossRef](#)]
21. Weinan, E.; Yu, B. The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems. *Commun. Math. Stat.* **2017**, *6*, 1–12.
22. Sirignano, J.A.; Konstantinos, S. DGM: A deep learning algorithm for solving partial differential equations. *J. Comput. Phys.* **2018**, *375*, 1339–1364. [[CrossRef](#)]
23. Raissi, M.; Perdikaris, P.; Karniadakis, G.E. Physics-informed neural network: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **2019**, *378*, 686–707. [[CrossRef](#)]
24. Lu, L.; Meng, X.; Mao, Z.; Karniadakis, G.E. DeepXDE: A deep learning library for solving differential equations. *SIAM Rev.* **2021**, *63*, 208–228. [[CrossRef](#)]
25. Pang, G.; Lu, L.; Karniadakis, G.E. fPINNs: Fractional physics-informed neural networks. *SIAM J. Sci. Comput.* **2019**, *41*, A2603–A2626. [[CrossRef](#)]
26. Zhang, D.; Lu, L.; Guo, L.; Karniadakis, G.E. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. *J. Comput. Phys.* **2019**, *397*, 108850. [[CrossRef](#)]
27. Zhang, D.; Guo, L.; Karniadakis, G.E. Learning in modal space: Solving time-dependent stochastic PDEs using physics-informed neural networks. *SIAM J. Sci. Comput.* **2020**, *42*, A639–A665. [[CrossRef](#)]
28. Chen, Y.; Lu, L.; Karniadakis, G.E.; Negro, L.D. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. *Opt. Express* **2020**, *28*, 11618–11633. [[CrossRef](#)]
29. Lu, L.; Pestourie, R.; Yao, W.; Wang, Z.; Verdugo, F.; Johnson, S.G. Physics-informed neural networks with hard constraints for inverse design. *arXiv* **2021**, arXiv:2102.04626.
30. Raissi, M.; Yazdani, A.; Karniadakis, G.E. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science* **2020**, *367*, 1026–1030. [[CrossRef](#)]
31. Yazdani, A.; Lu, L.; Raissi, M.; Karniadakis, G.E. Systems biology informed deep learning for inferring parameters and hidden dynamics. *PLoS Comput. Biol.* **2020**, *16*, e1007575. [[CrossRef](#)]
32. Jagtap, A.D.; Karniadakis, G.E. Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Commun. Comput. Phys.* **2020**, *28*, 2002–2041.
33. Jagtap, A.D.; Kharazmi, E.; Karniadakis, G.E. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Comput. Methods Appl. Mech. Eng.* **2020**, *365*, 113028. [[CrossRef](#)]
34. Li, W.; Xiang, X.; Xu, Y. Deep domain decomposition method: Elliptic problems. *PMLR* **2020**, *107*, 269–286.
35. Wang, Z.J.; Zhang, Z.W. A mesh-free method for interface problems using the deep learning approach. *J. Comput. Phys.* **2020**, *400*, 108963. [[CrossRef](#)]
36. Hu, W.F.; Lin, T.S.; Lai, M.C. A discontinuity capturing shallow neural network for elliptic interface problems. *arXiv* **2021**, arxiv:2106.05587.
37. He, C.Y.; Hu, X.Z.; Mu, L. A mesh-free method using piecewise deep neural network for elliptic interface problems. *J. Comput. Appl. Math.* **2022**, *412*, 114358. [[CrossRef](#)]
38. Wu, S.D.; Zhu, A.Q.; Tang, Y.F.; Lu, B.Z. On convergence of neural network methods for solving elliptic interface problems. *arXiv* **2022**, arxiv:2203.03407v2.
39. Jagtap, A.D.; Kawaguchi, K.; Karniadakis, G.E. Locally adaptive activation functions with slope recovery term for deep and physics-informed neural networks. *Proc. R. Soc. A* **2020**, *476*, 20200334. [[CrossRef](#)]

40. Jagtap, A.D.; Kawaguchi, K.; Karniadakis, G.E. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *J. Comput. Phys.* **2020**, *404*, 109136. [[CrossRef](#)]
41. Wang, S.F.; Yu, X.L.; Perdikaris, P. When and why PINNs fail to train: A neural tangent kernel perspective. *J. Comput. Phys.* **2022**, *449*, 110768. [[CrossRef](#)]
42. McClenny, L.D.; Braga-Neto, U.M. Self-Adaptive Physics-Informed Neural Networks using a Soft Attention Mechanism. *arXiv* **2022**, arXiv:2009.04544.
43. van der Meer, R.; Oosterlee, C.W.; Borovykh, A. Optimally weighted loss functions for solving PDEs with Neural Networks. *J. Comput. Appl. Math.* **2022**, *405*, 113887. [[CrossRef](#)]

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.