Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Adaptive Neural Subspaces with Geometry-Aware Quadrature for Interface Problems

Feiyi Liao
Academy of Mathematics and Systems Science, Chinese Academy of Sciences
Zhongshuo Lin
Peking University
Haochen Liu
Academy of Mathematics and Systems Science, Chinese Academy of Sciences
Yifan Wang
Central University of Finance and Economics
Hehu Xie
hhxie@lsec.cc.ac.cn

Academy of Mathematics and Systems Science, Chinese Academy of Sciences
Mingzhu Zhang
Academy of Mathematics and Systems Science, Chinese Academy of Sciences

Research Article

Keywords: Neural network, numerical quadrature schemes, adaptive subspace approximation, machine learning, partial differential equation, interface problem

Posted Date: November 10th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11292477/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Adaptive Neural Subspaces with Geometry-Aware Quadrature for Interface Problems*

Feiyi Liao†, Zhongshuo Lin‡, Haochen Liu§, Yifan Wang¶, Hehu Xie‖ and Mingzhu Zhang**

Abstract

Based on the combination of neural networks, numerical quadrature schemes and adaptive subspace approximation, we propose a new machine learning method for solving interface problems. The neural network is employed to construct the basis functions of a finite-dimensional subspace, upon which the discrete solution is obtained via subspace approximation. In particular, an a posteriori error estimator can be derived from the hypercircle technique within this subspace approximation framework. This estimator can further be used as the loss function to adaptively refine the neural network parameters. The high accuracy achieved by the proposed method also demonstrates that machine learning approaches equipped with high precision integration are capable of automatically handling singularities arising in interface problems. Numerical results illustrate that the proposed method attains high accuracy for interface problem simulations when equipped with a high precision numerical integration scheme for designing loss functions.

Keywords. Neural network, numerical quadrature schemes, adaptive subspace approximation, machine learning, partial differential equation, interface problem.

AMS subject classifications. 65N30, 65N25, 65L15, 65B99.

*This work was supported by the Strategic Priority Research Program of the Chinese Academy of Sciences (Grant No. XDB0620203, XDB0640000, XDB0640300, XDA0480504), National Key Laboratory of Computational Physics (No. 6142A05230501), National Natural Science Foundations of China (NSFC 12331015), National Center for Mathematics and Interdisciplinary Science, CAS.

†SKLMS, ICMSEC, NCMIS, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, 100190 China, and School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing, 100049, China (liaofeiyi@amss.ac.cn).

‡School of Mathematical Sciences, Peking University, Beijing 100871, China, and Peking University Chongqing Research Institute of Big Data, Chongqing 401329, China (linzhongshuo@pku.edu.cn).

§SKLMS, ICMSEC, NCMIS, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, 100190 China, and School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing, 100049, China (liuhaochen@lsec.cc.ac.cn).

¶School of Statistics and Mathematics, Central University of Finance and Economics, Beijing 102206, China (wangyifan@lsec.cc.ac.cn).

‖SKLMS, ICMSEC, NCMIS, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, 100190 China, and School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing, 100049, China (hhxie@lsec.cc.ac.cn).

**SKLMS, ICMSEC, NCMIS, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, 100190 China, and School of Mathematical Sciences, University of Chinese Academy of Sciences, Beijing, 100049, China (mzzhang@amss.ac.cn).

1 Introduction

In recent years, the neural network (NN) based machine learning methods for solving partial differential equations (PDEs) have attracted increasing attention from computational mathematics and scientific computing societies. These methods are expected to be alternatives to classical numerical methods such as the Finite Element Method (FEM), especially for solving high-dimensional problems and singular problems. However, different from FEM, the lack of error control in the NN-based methods makes it difficult for the NN-based method to achieve high accuracy in solving PDEs, although it benefits from the strong representational capacity of NNs.

More specifically, there are several types of well-known NN-based methods such as deep Ritz [9], deep Galerkin method [27], PINN [24], and weak adversarial networks [39] for solving PDEs by designing different loss functions. Among these methods, the loss functions always include computing integrations for the functions defined by NNs. Direct numerical integration for high-dimensional functions always meets the "curse of dimensionality". Always, the high-dimensional integration is computed using the Monte Carlo method along with some sampling tricks [9, 10]. Following this idea, these NN-based machine learning methods always use the Monte Carlo method, even in low-dimensional cases. Due to the low convergence rate of the Monte Carlo method, it is challenging to achieve high accuracy and a stable convergence process.

Recently, to control the integration error, we propose a type of tensor neural network (TNN) and the corresponding machine learning method to solve high-dimensional problems with high accuracy [31, 33, 32]. The reason why the TNN-based method achieves high accuracy is that the high-dimensional integration of TNN in the loss functions can be transformed into one-dimensional integration, which allows the classical quadrature schemes with high accuracy to be implemented. The TNN-based machine learning method has already been used to solve high-dimensional eigenvalue and boundary value problems based on the Ritz type of loss functions [31]. Furthermore, in [34], the multi-eigenpairs can also be computed with the machine learning method designed by combining the TNN and the Rayleigh-Ritz process. So far, the TNN-based machine learning method has shown a strong ability to solve high-dimensional problems.

In addition to high-dimensional problems, solving low-dimensional PDEs with singularities and discontinuities is also tricky. The existence of a singularity makes the solution of PDE have different approximate difficulties in different locations. Classical numerical methods, such as the FEM, have developed adaptive techniques for solving singular and interface problems. There are three important adaptive finite element methods (AFEMs): the h-adaptive, the p-adaptive, and the r-adaptive methods. Especially, we should introduce the adaptive mesh redistribution (i.e., Moving Mesh Method) in the finite element method [13, 15, 20]. Techniques for the moving mesh method continuously reposition a fixed number of mesh nodes and degrees of freedom, and so they improve the resolution in particular locations of the computational domain. These methods not only inherit the error estimation theories of FEM but also introduce a reliable a posteriori error estimate as the indicator to find a better trial function space suitable for singular problems. Based on this idea, in our previous work [33], the a posteriori error estimator is adopted as the loss function of the TNN-based method for solving boundary value problems and eigenvalue problems, and the numerical experiments on the model problems obtain the

accuracy close to the machine error and an efficient optimization path.

In this paper, inspired by the error analysis framework of FEM, we decompose and analyze the error during the process of the NN-based method for solving PDE. We propose a new adaptive neural network subspace method (ANNSM) based on error decomposition. ANNSM can adaptively update the trial function space represented by NN with the training process based on the a posteriori error estimator, thus reducing the subspace approximation error. In particular, for the second-order elliptic interface problems, we propose a new loss function form based on a posteriori error estimation. In addition, ANNSM employs an alternating optimization process to iteratively obtain a better NN subspace and an optimal Galerkin approximation on this subspace, thereby reducing the difficulty of the optimization and controlling the optimization error.

An outline of the paper goes as follows. In Section 2, following the idea of FEMs, we build the framework for error estimates of NN-based method for solving PDEs. As applications of the understanding in Section 2, we will consider the NN subspace based machine learning method in Section 3. Section 5 is devoted to designing the machine learning methods for elliptic problems with interface and discontinuous coefficients, respectively. Some concluding remarks are given in the last section.

2 Rethinking machine learning method for solving PDEs

In this section, by replacing the finite element space with neural network space, we develop an algorithmic framework of the NN-based machine learning method for solving PDEs based on the idea of the FEM. Furthermore, we provide an error analysis of the proposed method, grounded in the error estimate theory of the FEM.

Both the finite element method and the neural network approach for solving partial differential equations can be divided into three corresponding steps in their solution procedures. For the finite element method (FEM), the procedure is as follows:

- **Space Construction:** A finite element space is built from the mesh.
- **Formulation:** The Ritz or Galerkin weak form of the PDE is constructed on the finite element space.
- **Solution:** The resulting discrete optimization or linear system is solved to obtain the FEM solution.

Similarly, the machine learning approach follows a parallel workflow:

- **Space Construction:** A neural network is built to serve as the trial space.
- **Formulation:** A loss function is designed based on the Ritz or residual form of the concerned PDE by NN.
- **Solution:** The network parameters are trained by optimizing the discrete loss function, yielding the neural network solution.

To more clearly illustrate the solution processes of these two methods, we consider, as an example, the following second-order elliptic equation: Find $u \in H_0^1(\Omega)$ such that

$$\begin{cases} -\nabla \cdot (\mathcal{A}\nabla u(x)) + \beta u = f(x), & x \in \Omega, \\ u(x) = 0, & x \in \partial\Omega, \end{cases} \tag{2.1}$$

where $\mathcal{A} \in \mathbb{R}^{d\times d}$ is a symmetric positive definite coefficient matrix and the coefficient function β is non-negative on the computing domain.

In order to describe the finite element method, we define the Ritz form for the problem (2.1) as follows

$$u = \inf_{v\in H_0^1(\Omega)} \int_\Omega \left(\frac{1}{2} \left(\nabla v \cdot \mathcal{A}\nabla v + \beta v^2 \right) - fv \right) d\Omega. \tag{2.2}$$

From the variational theory, the problem (2.2) is equivalent to the original problem (2.1). The finite element method is defined by restricting the Ritz problem (2.1) to the finite dimensional subspace which is built by using the piecewise polynomial on the triangulation for the computing domain Ω. Overall, the standard procedure of the finite element method for solving problem (2.1) can be described by Algorithm 1.

Algorithm 1: Procedure of finite element method for solving problem (2.1)

---

1. Generate the mesh $\mathcal{T}_h$ for the computing domain and build the finite element space V_h on the mesh $\mathcal{T}_h$. The finite element space V_h consists of piecewise polynomial basis functions $\varphi_{j,h}$ which can be denoted as follows

$$V_h := \{\varphi_{1,h}, \cdots, \varphi_{N_h,h}\}, \tag{2.3}$$

   where $N_h := \dim V_h$.

2. Build the Ritz form for problem (2.1) on the finite element space V_h

$$u_h := \inf_{v_h\in V_h} \int_\Omega \left(\frac{1}{2} \left(\nabla v_h \cdot \mathcal{A}\nabla v_h + \beta v_h^2 \right) - f v_h \right) d\Omega, \tag{2.4}$$

   where $u_h = \sum_{j=1}^{N_h} c_j \varphi_{j,h}$ denotes the finite element solution.

3. Solve the deduced discrete optimization problem (2.4) to obtain the coefficient $\{c_j\}_{j=1}^{N_h}$ and the finite element solution $u_h = \sum_{j=1}^{N_h} c_j \varphi_{j,h}$.

Similar to the idea of the finite element method, the machine learning method for problem (2.1) can be formulated by restricting the Ritz problem (2.2) to a trial space constructed by using a neural network. Let V_{NN} denote the neural network-based trial space. The corresponding restricted Ritz problem is then defined as follows

$$u_{\mathrm{NN}}^{(1)} := \inf_{v_N\in V_{\mathrm{NN}}} \int_\Omega \left(\frac{1}{2} \left(\nabla v_N \cdot \mathcal{A}\nabla v_N + \beta v_N^2 \right) - f v_N \right) d\Omega, \tag{2.5}$$

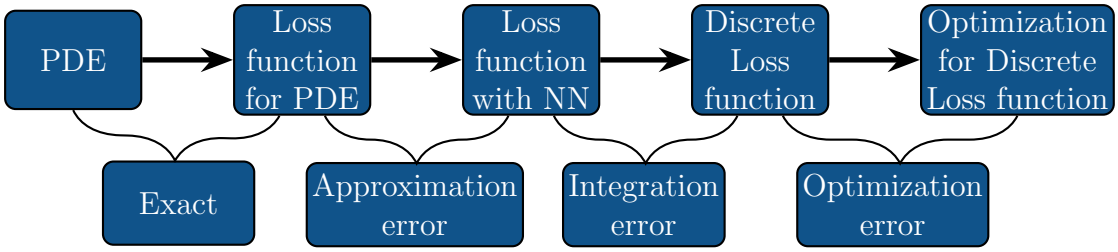


Figure 1: Errors of machine learning method for problem (2.1).

where $u_{\mathrm{NN}}^{(1)}$ denotes the solution to the problem (2.5).

To perform the integration in (2.5), usually we need to select quadrature points x_i and the corresponding weights w_i, $i = 1, \cdots, K$, where K denotes the total number of quadrature points. Based on this quadrature scheme, we define the computational loss function and formulate the associated discrete optimization problem as follows

$$u_{\mathrm{NN}}^{(2)} := \inf_{v_N \in V_{\mathrm{NN}}} \sum_{i=1}^{K} \left(\frac{1}{2} \left(\nabla v_N(x_i) \mathcal{A}(x_i) \nabla v_N(x_i) + \beta(x_i) v_N(x_i)^2 \right) - f(x_i) v_N(x_i) \right) w_i, \quad (2.6)$$

where $u_{\mathrm{NN}}^{(2)}$ denotes the neural network solution to (2.6). To devise a numerical quadrature scheme with reasonable accuracy, we require the geometric information of the computational domain. This is why we refer to this way as Geometry-Aware in this paper.

The training steps in machine learning methods essentially perform optimization for the discrete problem (2.6). We denote by $u_{\mathrm{NN}}^{(3)}$ the approximation solution obtained from $u_{\mathrm{NN}}^{(2)}$ after sufficient training steps. Based on these considerations, the machine learning method for solving the problem (2.1) can be described by the procedure outlined in Algorithm 2, and the corresponding error decomposition is given in Figure 1.

Algorithm 2: General procedure of machine learning method for solving problem (2.1)

1. Choose some types of architecture to build the neural network V_{NN} as the trial space;
2. Build the loss function by using the Ritz form and define the corresponding discrete problem (2.6) wiht some type of quadrature scheme.
3. Implement sufficient training steps to the optimization problem (2.6) to obtain the approximation $u_{\mathrm{NN}}^{(3)}$ as the final neural network solution.

During training, the neural network parameters are iteratively adjusted to improve the quality of its solutions. It's important to note that the number of parameters remains constant throughout the training process. This characteristic is similar to that of the moving mesh method used in finite element schemes. From this viewpoint, the neural network-based machine learning approach can be understood within the framework of the finite element method, enabling us to perform a corresponding error analysis.

For the following analysis, we define the bilinear form $a(\cdot,\cdot)$ as follows

$$a(u,v)=\int_{\Omega}\left(\nabla v\cdot\mathcal{A}\nabla u+\beta uv\right)d\Omega,\quad \forall u\in H_0^1(\Omega),\ \forall v\in H_0^1(\Omega), \tag{2.7}$$

and then the energy norm as $\|v\|_a=\sqrt{a(v,v)}$.

It is easy to know that the exact solution u of (2.1) satisfies the following Galerkin formula

$$a(u,v)=(f,v),\quad \forall v\in H_0^1(\Omega), \tag{2.8}$$

where $(\cdot,\cdot)$ denote the $L^2(\Omega)$-inner product

$$(f,v)=\int_{\Omega}fvd\Omega.$$

Similar to the Céa lemma, it holds that the function $u_{\mathrm{NN}}^{(1)}$ defined by (2.5) and the exact solution (2.2) satisfy the following error estimate

$$\left\|u-u_{\mathrm{NN}}^{(1)}\right\|_a=\inf_{v\in V_{\mathrm{NN}}}\|u-v\|_a. \tag{2.9}$$

Meanwhile, from the definition of the discrete loss function (2.6) and the error estimate for the numerical integration, we have the following inequality for the error between $u_{\mathrm{NN}}^{(1)}$ and $u_{\mathrm{NN}}^{(2)}$

$$\left\|u_{\mathrm{NN}}^{(1)}-u_{\mathrm{NN}}^{(2)}\right\|_a\le\varepsilon_{\mathrm{int}}. \tag{2.10}$$

The machine learning method then proceeds to the training phase, which involves optimizing the discrete loss function to obtain the final neural network solution $u_{\mathrm{NN}}^{(3)}$. For this step, the optimization error can be defined as follows:

$$\left\|u_{\mathrm{NN}}^{(3)}-u_{\mathrm{NN}}^{(2)}\right\|_a\le\varepsilon_{\mathrm{opt}}. \tag{2.11}$$

Based on the above analysis, the total error of the neural network solution can be expressed as follows:

$$\left\|u-u_{\mathrm{NN}}^{(3)}\right\|_a\le\inf_{v\in V_{\mathrm{NN}}}\|u-v\|_a+\varepsilon_{\mathrm{int}}+\varepsilon_{\mathrm{opt}}. \tag{2.12}$$

Regarding the approximation error, it is well known that neural networks possess strong expression ability which leads that the error $\inf_{v\in V_{\mathrm{NN}}}\|u-v\|_a$ can be sufficiently small with high efficiency. As for the integration error, from the definitions of loss functions (2.5) and (2.6), there includes integration which are computed always by using Monte Carlo or sampling methods in common machine learning methods such as Deep-Ritz or Deep-Galerkin methods. Of course, when Ω is a low-dimensional domain, the classical quadrature schemes can also be used to perform the integration with high accuracy. Unfortunately, since neural network functions are generally not polynomial, classical

numerical quadrature inevitably introduce integration errors. In addition, the optimization error should not be overlooked, as the loss function associated with neural networks is generally non-convex. Therefore, it is essential to employ suitable optimization algorithms and incorporate specialized techniques to enhance the convergence speed and ensure stability during the training process.

Based on the analysis and understanding of machine learning methods for solving PDEs, we hope to simultaneously control these errors. Correspondingly, we propose an adaptive neural network subspace method. Detailed considerations of the method will be presented in the following sections.

3 Adaptive neural network subspace method

Inspired by the subspace approximation theory of finite element method, we introduce the concept of NN subspace, based on which we design the corresponding machine learning method for solving PDEs.

In order to describe the adaptive subspace based machine learning method, we first define the neural network subspace as follows

$$V_p := \text{span}\Big\{\varphi_j(x;\theta), \quad j = 1, \cdots, p\Big\}, \tag{3.1}$$

where $\varphi_j(x;\theta)$ denote the j-th output of the concerned NN $\psi(x,\theta) : \mathbb{R}^d \to \mathbb{R}^p$. In what follows, we aim to find the best approximation $\Psi(x;c,\theta)$ to the PDE solution within the subspace V_p, where Ψ takes the form

$$\Psi(x;c,\theta) = \sum_{j=1}^{p} c_j \varphi_j(x;\theta), \tag{3.2}$$

with $c = (c_1, c_2, \cdots, c_p)^\top \in \mathbb{R}^p$ denoting the coefficient vector associated with the NN subspace V_p. Here, the specific architecture of the network $\psi(x,\theta)$ is flexible, and can be chosen as either a fully connected neural network (FNN) (cf. Figure (2a)) or a tensor neural network (cf. Figure (2b)) [31, 33, 32]. The parameter θ represents the weights and biases of the NN, and p is the dimension of the subspace. Based on this subspace V_p, we can follow the subspace approximation process to obtain the solution $u_p \in V_p$. For this aim, we define the variational form for the problem (2.1): Find $u \in V$ such that

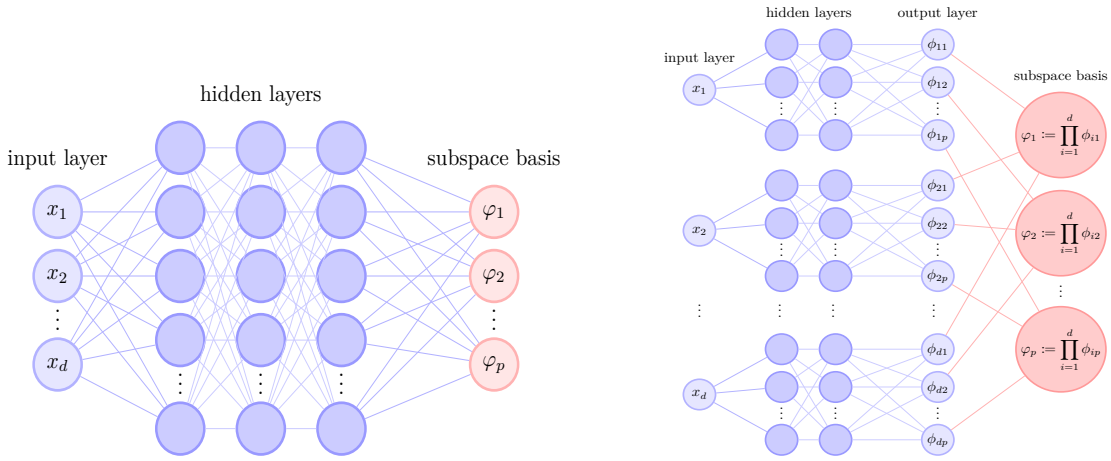
$$a(u,v) = (f,v), \quad \forall v \in V, \tag{3.3}$$

where $V := H_0^1(\Omega)$. Here and hereafter in this paper, the bilinear form $a(\cdot,\cdot)$ is defined as (2.7) and the energy norm $\|\cdot\|_a$ is defined as $\|v\|_a = \sqrt{a(v,v)}$.

The standard Galerkin approximation scheme on subspace V_p for the problem (3.3) is: Find $u_p \in V_p$ such that

$$a(u_p, v_p) = (f, v_p), \quad \forall v_p \in V_p. \tag{3.4}$$

Based on the basis of the subspace V_p in (3.1), we can assemble the corresponding stiffness matrix and right hand side term as in the finite element method. By solving the deduced linear system, we obtain the Galerkin approximation Ψ (3.2) in subspace V_p.



(a) FNN (b) TNN

Figure 2: The neural network architecture of FNN and TNN.

Then following the idea of the adaptive finite element methods or moving mesh methods, we update the subspace V_p adaptively during the training process. By "adaptively", we mean that the NN parameters and thus the fixed-structure NN subspace are updated based on some criterion, which in this case is the optimization of the loss function, to achieve better accuracy performance. For example, if the Ritz-type loss function

$$\mathcal{L}(\Psi) = \frac{1}{2}a(\Psi, \Psi) - (f, \Psi) \tag{3.5}$$

is used, then the training process is actually to minimize the error $\|u - \Psi\|_a$ of the NN approximation Ψ in the energy norm.

The details of the proposed method are given in Algorithm 3.

Remark 3.1. *In [33], the TNN is adopted to build the NN subspace for solving high-dimensional PDEs. Furthermore, based on the finite dimensional subspace approximation, the a posteriori error estimator is designed for the TNN approximation. Then, following the idea of adaptive finite element method, a type of TNN-based machine learning method is designed for solving high-dimensional PDEs with high accuracy, where the a posteriori error estimator acts as the loss function and the training step is adopted to update the TNN subspace. For more information, please refer to [33].*

Remark 3.2. *Sheng and his collaborators develop a subspace-based neural network, which includes a subspace layer, to solve PDEs. Their approach involves doing some training steps to the subspace neural network to improve the approximation accuracy than the random initial neural network. About more information about the random neural networks and their applications to solving PDEs, please refer to [2, 7, 8, 16, 21, 25, 28]. Using the learned subspace, the final solution to the PDE is obtained through a least-squares method. In their works [6, 22, 37], they applied this method to solve some types of equations, including one-dimensional Helmholtz equations, one-dimensional convection equations, one-dimensional parabolic equations, two-dimensional Poisson equations, anisotropic diffusion equations, and eigenvalue problems, achieving impressive computational accuracy.*

Algorithm 3: Adaptive neural network subspace method for elliptic problem (2.1)

1. Initialization step: Set $\ell = 0$ and build initial NN $\Psi(x; c^{(\ell)}, \theta^{(\ell)})$. Set the loss function $\mathcal{L}(\Psi(x; c, \theta))$, the maximum training steps M, the parameters setting of optimization algorithm.

2. Define the p-dimensional space $V_p^{(\ell)}$ as follows

$$V_p^{(\ell)} := \text{span}\left\{\varphi_j(x; \theta^{(\ell)}), \quad j = 1, \cdots, p\right\}.$$

   Assemble the stiffness matrix $A^{(\ell)}$ and the right-hand side term $B^{(\ell)}$ on $V_p^{(\ell)}$ as follows

$$A_{m,n}^{(\ell)} = a(\varphi_n^{(\ell)}, \varphi_m^{(\ell)}), \quad B_m^{(\ell)} = (f, \varphi_m^{(\ell)}), \ 1 \le m, n \le p.$$

3. Solve the following linear equation to obtain the solution $c \in \mathbb{R}^p$:

$$A^{(\ell)} c = B^{(\ell)}.$$

   Update the coefficient parameter as $c^{(\ell+1)} = c$. Then the Galerkin approximation on the space $V_p^{(\ell)}$ for problem (3.3) is $\Psi(x; c^{(\ell+1)}, \theta^{(\ell)})$.

4. Update the neural network parameter $\theta^{(\ell)}$ of NN through some optimization steps, i.e., optimizing the loss function $\mathcal{L}(\Psi(x; c^{(\ell+1)}, \theta^{(\ell)}))$.

5. Set $\ell = \ell + 1$ and go to Step 2 for the next step until $\ell = M$.

Different from the fixed neural network subspace in these methods, the approach proposed in this paper improves accuracy by continuously and adaptively updating the neural network subspace during training.

In Algorithm 3, the NN subspace is updated by minimizing the loss function which defines the target of the training steps. Similar to adaptive finite element methods, the a posteriori error estimators can also be employed to construct the loss functions in machine learning approaches for solving PDEs. In order to derive the a posteriori error estimate for the subspace approximation u_p, we introduce the following eigenvalue problem: Find $(\lambda, u) \in \mathbb{R} \times V$ such that

$$a(u, v) = \lambda(u, v), \quad \forall v \in V. \tag{3.6}$$

It is well known that the eigenvalue problem (3.6) has the following eigenvalue series

$$0 < \lambda_1 \le \lambda_2 \le \cdots \le \lambda_k \le \cdots \to +\infty.$$

Then the mimimum eigenvalue $\lambda_{\min} = \lambda_1$ and the following inequality holds

$$\frac{\|v\|_a^2}{\|v\|_0^2} \ge \lambda_{\min} > 0, \quad \forall v \in V. \tag{3.7}$$

The following integration by parts formula plays as a key role to derive the a posteriori error estimate.

Lemma 3.1. *Let $\Omega \subset \mathbb{R}^d$ be a bounded Lipschitz domain with unit outward normal $\mathbf{n}$ to the boundary $\partial\Omega$. Then the following Green's formula holds*

$$\int_\Omega v\mathrm{div}\mathbf{y}d\Omega + \int_\Omega \mathbf{y}\cdot\nabla v d\Omega = \int_{\partial\Omega} v\mathbf{y}\cdot\mathbf{n}ds, \quad \forall v \in H^1(\Omega),\ \forall \mathbf{y}\in\mathbf{W}, \tag{3.8}$$

where $\mathbf{W} := \mathbf{H}(\mathrm{div},\Omega)$.

Then the Galerkin approximation u_p satisfies the following a posteriori error estimation.

Theorem 3.1. *[29, Theorem 1] Assume $u \in V$ and $u_p \in V_p \subset V$ are the exact solution of (3.3) and subspace approximation u_p defined by (3.4), respectively. The following upper bound holds*

$$\|u - u_p\|_a \le \inf_{\mathbf{y}\in\mathbf{W}} \eta(u_p,\mathbf{y}), \tag{3.9}$$

where $\eta(u_p,\mathbf{y})$ is defined as follows

$$\eta(u_p,\mathbf{y}) := \left(\|\beta^{-\frac{1}{2}}(f-\beta u_p + \mathrm{div}(\mathcal{A}\mathbf{y}))\|_0^2 + \|\mathcal{A}^{\frac{1}{2}}(\mathbf{y}-\nabla u_p)\|_0^2\right)^{\frac{1}{2}}. \tag{3.10}$$

These a posteriori error estimates provide an upper bound of the approximation errors in the energy norm. As we know, the numerical methods aim to produce approximations that have optimal errors in the sense of the energy norm. This inspires us to use these a posteriori error estimates as the loss functions of the NN-based machine learning method for solving the concerned problems in this paper.

Take the homogeneous Dirichlet type of boundary value problem as an example. Denote the Galerkin approximation on the space V_p for problem (3.3) as $u_p = u_p(V_p)$. Now, consider the subspace $V_p \subset V$ with a fixed dimension p. From Theorem 3.1, as long as $V_p \subset V$, the following inequality holds

$$\inf_{\substack{V_p\subset V\\ \dim(V_p)=p}} \|u - u_p(V_p)\|_a \le \inf_{\substack{V_p\subset V\\ \dim(V_p)=p}} \inf_{\mathbf{y}\in\mathbf{W}} \eta(u_p(V_p),\mathbf{y}). \tag{3.11}$$

The error of the best p-dimensional-subspace Galerkin approximation can be controlled by solving the optimization problem on the right-hand side of the inequality (3.11). Based on the above discussion, we can define the following loss function at the ℓ-th step in Algorithm 3

$$\mathcal{L}(u_p) = \inf_{\mathbf{y}\in\mathbf{W}} \eta^2(u_p(V_p^{(\ell)}),\mathbf{y}).$$

In real implementation, in order to improve the computational efficiency, we choose $\mathbf{y} = \nabla u_p \in \mathbf{W}$ and define the loss function as follows

$$\mathcal{L}(u_p) = \eta^2(u_p,\nabla u_p) = \left\|\beta^{-\frac{1}{2}}(f-\beta u_p + \nabla\cdot(\mathcal{A}\nabla u_p))\right\|_0^2. \tag{3.12}$$

To further interprete the loss function (3.12), we introduce the equality relationship between the a posteriori error estimate and the errors associated with the primal problem (3.3) as well as its dual problem. From [29], the minimization problem in (3.9) is equivalent to another variational problem, which is called the dual problem to (3.3), that is: Find $\mathbf{y}^* \in \mathbf{W}$ such that

$$a^*(\mathbf{y}^*, \mathbf{z}) = \mathcal{F}^*(\mathbf{z}), \quad \forall \mathbf{z} \in \mathbf{W}, \tag{3.13}$$

where

$$a^*(\mathbf{y}^*, \mathbf{z}) = \int_\Omega \left(\beta^{-1}\mathrm{div}(\mathcal{A}\mathbf{y}^*)\mathrm{div}(\mathcal{A}\mathbf{z}) + \mathbf{y}^* \cdot \mathcal{A}\mathbf{z}\right) d\Omega, \quad \mathcal{F}^*(\mathbf{z}) = -\int_\Omega \beta^{-1} f \mathrm{div}(\mathcal{A}\mathbf{z}) d\Omega.$$

It is obvious that the bilinear form $a^*(\cdot,\cdot)$ induces an inner product in $\mathbf{W}$ and the corresponding norm is $\|\mathbf{z}\|_* = \sqrt{a^*(\mathbf{z},\mathbf{z})}$. Due to the Riesz representation theorem, the dual problem (3.13) has an unique solution $\mathbf{y}^* = \boldsymbol{\nabla} u$ and $\eta\left(u_p, \mathbf{y}^*\right) = \|u - u_p\|_a$ [29].

Assume that $\mathbf{y}$ is any approximation to $\mathbf{y}^*$ for the dual problem (3.13), and u_p is the Galerkin approximation. Then the following equality holds

$$\|u - u_p\|_a^2 + \|\mathbf{y}^* - \mathbf{y}\|_*^2 = \eta^2(u_p, \mathbf{y}). \tag{3.14}$$

Therefore, the loss function in (3.12) actually demonstrates the sum of the errors $\|u - u_p\|_a^2$ and $\|\mathbf{y}^* - \nabla u_p\|_*^2$, i.e.,

$$\mathcal{L}(u_p) = \eta^2(u_p, \nabla u_p) = \|u - u_p\|_a^2 + \|\mathbf{y}^* - \nabla u_p\|_*^2. \tag{3.15}$$

Furthermore, since $\mathbf{y}^* = \nabla u$, we have

$$\mathcal{L}(u_p) = \|u - u_p\|_a^2 + \|\nabla u - \nabla u_p\|_*^2. \tag{3.16}$$

For more information and the detailed proof, please refer to [29, Corollary 1].

For the case of $\beta = 0$, we modify the loss function (3.12) to the following form

$$\mathcal{L}(u_p) = \eta\left(u_p, \nabla u_p\right) = \|f - \beta u_p + \nabla \cdot (\mathcal{A}\nabla u_p)\|_0^2 . \tag{3.17}$$

The corresponding a posteriori error estimate in this case can be deduced in the same way. We omit the proof here.

From another perspective, if we devide the unknown parameters in the trial functions (3.2) into two parts, i.e., the linear coefficients c and the neural network parameters θ, then Algorithm 3 can be interpreted as an alternating optimization procedure, where the linear coefficients c are updated via the Galerkin approximation scheme and the neural network parameters θ are updated by minimizing the loss function. It is well known that the Galerkin approximation yields the best approximation in the energy norm within the subspace. Numerical results has indicated that obtaining the optimal linear coefficient c in the energy norm or in the least-squares sense [2, 25, 37] rather than jointly optimizing all parameters, can lead to better convergence and stability of the training process. Overall, ANNSM proposed in this paper takes into account both the optimization error and the approximation error, and when combined with high-precision integration scheme, it aims to achieve improved accuracy for solve PDEs.

4 Adaptive neural network subspace method for singular problem

In order to show that the machine learning method has the ability to solve the singular problem when the integration in the loss function is computed with high accuracy, we consider the following Laplace problem on the unit square with singularity: Find $u \in V$ such that

$$\begin{cases} -\Delta u &= f, \quad \text{in } \Omega, \\ u &= 0, \quad \text{on } \partial\Omega, \end{cases} \tag{4.1}$$

where $\Omega = (0,1) \times (0,1)$ and the right hand side term f is chosen as

$$\begin{aligned} f = &- x_2(x_2-1)\left[\left(x_1-\frac{1}{2}\right)^{-\frac{2}{3}}\left(\frac{70}{9}x_1\left(x_1-1\right)+\frac{11}{6}\right)+2\left(x_2-\frac{1}{2}\right)^{\frac{4}{3}}\right] \\ &- x_1(x_1-1)\left[\left(x_2-\frac{1}{2}\right)^{-\frac{2}{3}}\left(\frac{70}{9}x_2\left(x_2-1\right)+\frac{11}{6}\right)+2\left(x_1-\frac{1}{2}\right)^{\frac{4}{3}}\right], \end{aligned}$$

such that the exact solution is

$$u_* = x_1(1-x_1)x_2(1-x_2)\left(\left(x_1-\frac{1}{2}\right)^{\frac{4}{3}}+\left(x_2-\frac{1}{2}\right)^{\frac{4}{3}}\right). \tag{4.2}$$

It ie easy to observe that the solution exhibits "line" singularities along the lines $x_1 = \frac{1}{2}$ and $x_2 = \frac{1}{2}$. This problem is commonly used to evaluate the performance of adaptive finite element methods [23]. We solve this problem using the standard h-adpative finite element method (AFEM) with linear and quadratic elements. Using adaptive refinement technique, we obtain the finite element space of 128888 elements for linear element and 86926 elements for quadratic element, only achieving the accuracy of 2.320817×10^{-3} and 2.333962×10^{-3} for the finite element solution with respect to the relative error in the L^2 norm. In this work, we adopt the NN subspace-based machine learning method to solve the problem (4.1). For this aim, we build the trial function u_{NN} as follows

$$u_{\text{NN}}(x,\theta) := x_1(1-x_1)x_2(1-x_2)\Psi(x,\theta). \tag{4.3}$$

Then the trial function u_{NN} is guaranteed to satisfy the Dirichlet boundary condition.

We define the Ritz type of loss function for (4.1) as follows

$$\mathcal{L}(u_{\text{NN}}) := \frac{1}{2}\int_\Omega |\nabla u_{\text{NN}}|^2 d\Omega - \int_\Omega f u_{\text{NN}} d\Omega. \tag{4.4}$$

In Section 2, we provide the error estimate (2.12) for the neural network solution obtained by the machine learning algorithm. This result indicates that, in order to achieve high-accuracy numerical solutions with machine learning, it is essential to ensure the high accuracy of the loss function computation.

In the loss function (4.4), the term $\frac{1}{2}\|\nabla u_{\text{NN}}\|_0^2$ can be computed using the Gauss-Legendre quadrature. However, the singularity of the right-hand side function f poses

difficulties in achieving high accuracy for evaluating the term (f, u_{NN}). To handle the singular integral components $(x_1-\frac{1}{2})^{-2/3}$ and $(x_2-\frac{1}{2})^{-2/3}$ appearing in f, the integration domain is partitioned into subdomains. Through a suitable coordinate transformation, these singular terms are treated as weight functions, enabling the use of Gauss-Jacobi quadrature for accurate numerical integration.

We now briefly introduce the Gauss-Jacobi quadrature which is commonly used to compute weighted integrals over $[-1, 1]$ with the weight function $(1-x)^\alpha(1+x)^\beta$, where $\alpha > -1$ and $\beta > -1$. Specifically, using n nodes $\{z_i\}$ and weights $\{w_i\}$, the Gauss-Jacobi quadrature formula is given by

$$\int_{-1}^{1} g(x)(1-x)^\alpha(1+x)^\beta dx \approx \sum_{i=1}^{n} w_i g(z_i),$$

where the n nodes are the roots of the n-th Jacobi polynomial $P_n^{(\alpha,\beta)}(x)$, and the weights can be computed using the formula:

$$w_i = \frac{2^{\alpha+\beta+1}\Gamma(n+\alpha+1)\Gamma(n+\beta+1)}{n!\Gamma(n+\alpha+\beta+1)(1-z_i^2)[P_n^{(\alpha,\beta)'}(z_i)]^2}.$$

The Gauss-Jacobi quadrature formula with n nodes is exact for polynomials of degree up to $2n-1$. Here, we denote the Gauss-Jacobi quadrature formula with the weight function $(1-x)^\alpha(1+x)^\beta$ as the (α, β)-Gauss-Jacobi quadrature. For a detailed introduction to the Gauss-Jacobi quadrature, please refer to [26].

Next, we describe the computating details for the term (f, u_{NN}). Here, we omit the notation θ in u_{NN} and express it simply as $u_{\text{NN}}(x_1, x_2)$:

$$\begin{aligned}
&(f, u_{\text{NN}}) \\
=&\int_0^1\int_0^1 u_{\text{NN}}(x_1,x_2)\Bigg(-x_2(x_2-1)\left[\left(x_1-\frac{1}{2}\right)^{-\frac{2}{3}}\left(\frac{70}{9}x_1(x_1-1)+\frac{11}{6}\right)+2\left(x_2-\frac{1}{2}\right)^{\frac{4}{3}}\right] \\
&-x_1(x_1-1)\left[\left(x_2-\frac{1}{2}\right)^{-\frac{2}{3}}\left(\frac{70}{9}x_2(x_2-1)+\frac{11}{6}\right)+2\left(x_1-\frac{1}{2}\right)^{\frac{4}{3}}\right]\Bigg)dx_1dx_2 \\
=&I_1+I_2+I_3+I_4.
\end{aligned}$$

First, we define

$$\begin{aligned}
\widehat{u}_1(x_1,x_2) &= -x_2(x_2-1)\left(\frac{70}{9}x_1(x_1-1)+\frac{11}{6}\right)u_{\text{NN}}(x_1,x_2), \\
\widehat{u}_2(x_1,x_2) &= -x_1(x_1-1)\left(\frac{70}{9}x_2(x_2-1)+\frac{11}{6}\right)u_{\text{NN}}(x_1,x_2).
\end{aligned}$$

Then I_1, I_2, I_3, and I_4 are expressed as follows:

$$\begin{aligned}
I_1 &= \int_0^1\int_0^1\left(x_1-\frac{1}{2}\right)^{-\frac{2}{3}}\widehat{u}_1(x_1,x_2)dx_1dx_2, \\
I_2 &= \int_0^1\int_0^1\left(x_2-\frac{1}{2}\right)^{-\frac{2}{3}}\widehat{u}_2(x_1,x_2)dx_1dx_2,
\end{aligned}$$

$$I_3 = \int_0^1 \int_0^1 -2x_2(x_2-1)\left(x_2-\frac{1}{2}\right)^{\frac{4}{3}} u_{\mathrm{NN}}(x_1,x_2)dx_1dx_2,$$
$$I_4 = \int_0^1 \int_0^1 -2x_1(x_1-1)\left(x_1-\frac{1}{2}\right)^{\frac{4}{3}} u_{\mathrm{NN}}(x_1,x_2)dx_1dx_2.$$

Now we introduce the computational method for I_1. The integral I_2 can be computed in a similar manner, while I_3 and I_4 can be computed using the Gauss-Legendre quadrature over $[0,1]^2$. The detailed computation of I_1 is as follows:

$$\begin{aligned}
I_1 &= \int_0^1 \int_0^1 \left(x_1-\frac{1}{2}\right)^{-\frac{2}{3}} \widehat{u}_1(x_1,x_2)dx_1dx_2 \\
&= \int_0^1 \int_0^{\frac{1}{2}} \left(x_1-\frac{1}{2}\right)^{-\frac{2}{3}} \widehat{u}_1(x_1,x_2)dx_1dx_2 + \int_0^1 \int_{\frac{1}{2}}^1 \left(x_1-\frac{1}{2}\right)^{-\frac{2}{3}} \widehat{u}_1(x_1,x_2)dx_1dx_2 \\
&= 4^{\frac{1}{3}} \int_0^1 \int_{-1}^1 (1-x_1)^{-\frac{2}{3}} \widehat{u}_1\left(\frac{x_1+1}{4},x_2\right)dx_1dx_2 \\
&\quad + 4^{\frac{1}{3}} \int_0^1 \int_{-1}^1 (1+x_1)^{-\frac{2}{3}} \widehat{u}_1\left(\frac{x_1+3}{4},x_2\right)dx_1dx_2.
\end{aligned}$$

We treat $(1-x_1)^{-\frac{2}{3}}$ as the weight function with parameters $(\alpha,\beta)=(-\frac{2}{3},0)$ and $(1+x_1)^{-\frac{2}{3}}$ as the weight function with $(\alpha,\beta)=(0,-\frac{2}{3})$. Therefore, the integration is carried out as follows. For the first integral term, we employ the $(-\frac{2}{3},0)$-Gauss-Jacobi quadrature in the x_1 direction and the composite Gauss-Legendre quadrature over $[0,1]$ in the x_2 direction to evaluate the two-dimensional integral of the function $\widehat{u}_1\left(\frac{x_1+1}{4},x_2\right)$. For the second integral term, we use the $(0,-\frac{2}{3})$-Gauss-Jacobi quadrature in the x_1 direction, and again apply the composite Gauss-Legendre quadrature over $[0,1]$ in the x_2 direction to compute the integral of $\widehat{u}_1\left(\frac{x_1+3}{4},x_2\right)$.

Next, we describe the numerical experiment settings for this example. We choose TNN structure to build the NN subspace with dimension $p=100$. Each sub-network of the TNN (cf. Figure (2b)) is an FNN with three hidden layers, each of width 100. When computing the integrals in the loss function, we use 200 quadrature points for the Gauss-Jacobi quadrature method. For the Gauss-Legendre quadrature scheme included, the integration region is divided into 100 subintervals, with 8 Gauss-Legendre quadrature points selected in each subinterval.

The Adam optimizer with a learning rate of 0.001 is used for the first 50,000 training steps, followed by the LBFGS optimizer with a learning rate of 0.1 for the following 10,000 steps to obtain the final result. Figure 3 shows the comparison between the numerical solution obtained by the algorithm and the exact solution, as well as the error distribution. The L^2 relative error between the numerical solution and the exact solution is 2.893995×10^{-5}.

Additionally, we calculate the relative error of the numerical solution u_{NN} against the exact solution u_* on a uniformly distributed 1001×1001 test points $\{x^{(k)},k=1,\ldots,K\}$

over $[0,1]^2$. The relative error at the test points is defined as

$$\frac{\sqrt{\sum_{k=1}^{K}\left(u_{\mathrm{NN}}(x^{(k)})-u_*(x^{(k)})\right)^2}}{\sqrt{\sum_{k=1}^{K}\left(u_*(x^{(k)})\right)^2}},$$

which yields 2.190126×10^{-5}.

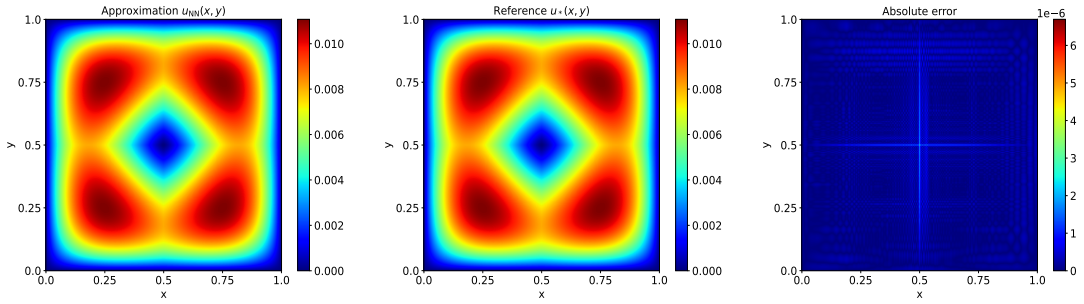


Figure 3: Numerical results for solving problem (4.1): the left panel shows the approximate solution, the middle panel shows the exact solution, and the right panel shows the error distribution.

To further verify the assertion that the integration accuracy affects the accuracy of the machine learning solution, we use 4000 Gauss-Legendre quadrature points per dimension to compute the loss function. Note that although a significantly larger number of quadrature points are employed, the singularity in the loss function is not explicitly treated. All other parameters remain unchanged, and under this setting, the resulting relative error on uniformly distributed test points is only 4.601047×10^{-2}. When the quadrature is refined near the singularity regions $[0.49,\ 0.51]$ and $[0.499,\ 0.501]$, the relative errors are reduced to 1.249979×10^{-2} and 5.822244×10^{-3}, respectively. These results indicate that as the integration accuracy improves, the overall precision of the algorithm also increases. In particular, the introduction of the Gauss-Jacobi quadrature method addresses the singularities in the integrand more effectively, resulting in the highest accuracy among all tests.

Finally, we also investigate the performance of the subspace which is built by a two-dimensional-input FNN with 4 hidden layers, each of width 64, and dimension $p = 100$. The two-dimensional quadrature points were generated as the Cartesian product of one-dimensional quadrature points. The number of quadrature points, optimization process, and optimization parameters are kept consistent with those used in the TNN case. The resulting relative L^2 error between the numerical solution and the exact solution is 4.567496×10^{-5}, and the relative error on the test points is 4.621387×10^{-5}. Overall, these results demonstrate that the proposed method, when combined with high-precision numerical integration, is capable of solving the singular problem with high accuracy.

5 Adaptive neural network subspace method for interface problem

In this section, we apply the proposed machine learning method to an important class of problems: second-order elliptic interface problems with discontinuous coefficients. These problems commonly arise in fields such as materials science, fluid dynamics, and electromagnetics, where discontinuities in coefficients across material interfaces lead to solutions that are smooth within subdomains but lack global regularity. Due to the low global smoothness and potentially complex interface geometries, achieving high accuracy by the finite element method is often challenging. Various finite element methods [3, 4, 5] and machine learning methods based on NNs have been developed for solving interface problems [35, 36, 38].

Unlike problems where the coefficients of the diffusion term are continuous, the gradient of the neural network approximation in interface problems is no longer globally continuous but instead piecewise continuous and may exhibit discontinuities across interfaces. This necessitates corresponding modifications to the a posteriori error estimator. This section will demonstrate how to design the NN-based machine learning method for solving interface problems. Additionally, we will also investigate the performances for both Ritz-type loss function and the one by the a posteriori error estimator.

Let $\Omega \subset \mathbb{R}^2$ be a convex polygonal and $\Omega_1 \subset \Omega$ be an open domain with Lipschitz boundary $\Gamma = \partial\Omega_1 \subset \Omega$. Let $\Omega_2 = \Omega \backslash \Omega_1$ (see Figure 4).

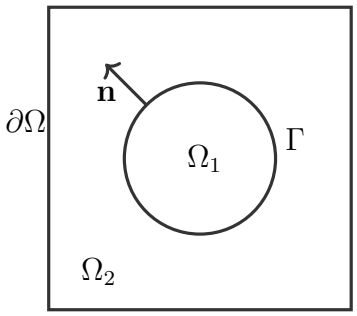


Figure 4: The boundary $\partial\Omega$, subdomains Ω_1, Ω_2 and interface Γ.

Here, we consider the following elliptic interface problem: Find $u \in H_0^1(\Omega)$ such that

$$\begin{cases} -\nabla \cdot (\alpha \nabla u) + \beta u &= f, \quad \text{in } \Omega_1 \cup \Omega_2, \\ u &= 0, \quad \text{on } \partial\Omega, \\ [u] = 0, \quad \left[\alpha \frac{\partial u}{\partial \mathbf{n}}\right] &= g, \quad \text{across } \Gamma, \end{cases} \tag{5.1}$$

where $[v]$ is the jump of a quantity v across the interface Γ and $\mathbf{n}$ the unit outward normal to the boundary $\partial\Omega_1$. For definiteness, let $[v](x) = v_1(x) - v_2(x)$, $x \in \Gamma$, with v_1 and v_2 the restrictions of v on Ω_1 and Ω_2, respectively. For the ease of description, we assume that the coefficient function $\beta \geq 0$ and α is positive and piecewise constant, i.e.

$$\alpha(x) = \alpha_1 \ \text{ for } x \in \Omega_1; \quad \alpha(x) = \alpha_2 \ \text{ for } x \in \Omega_2.$$

But the results of this section can be easily extended to more general elliptic interface problems.

For building the trial function, we use the functions $\omega(x)$ and $\gamma(x)$ to denote the boundaries $\partial\Omega$ and Γ implicity, i.e.

$$\partial\Omega := \left\{x \in \mathbb{R}^2 | \omega(x) = 0\right\}, \quad \Gamma := \left\{x \in \mathbb{R}^2 | \gamma(x) = 0\right\}.$$

In order to derive the a posteriori error estimates as the loss function, we first define the corresponding variational form for (5.1) as follows: Find $u \in H_0^1(\Omega)$ such that

$$a(u, v) = (f, v) + \langle g, v\rangle, \quad \forall v \in H_0^1(\Omega), \tag{5.2}$$

where the bilinear forms $a(\cdot,\cdot)$, $(\cdot,\cdot)$ and $\langle\cdot,\cdot\rangle$ are defined as follows

$$a(u, v) = \int_\Omega (\nabla v \cdot \alpha \nabla u + \beta uv)\, d\Omega, \quad (f, v) = \int_\Omega fv d\Omega, \quad \langle g, v\rangle = \int_\Gamma gv ds.$$

Based on the bilinear form $a(\cdot,\cdot)$, we can define the energy norm $\|\cdot\|_a$ as follows

$$\|v\|_a := \sqrt{a(v, v)}, \quad \forall v \in H_0^1(\Omega).$$

It is easy to know the Ritz form for the problem (5.2) is

$$u = \inf_{u\in V} \left(\frac{1}{2}a(v, v) - (f, v) - \langle g, v\rangle\right). \tag{5.3}$$

Then the Ritz-type loss function for the NN function ψ can be defined as follows

$$\mathcal{L}(\psi) = \frac{1}{2}a(\psi, \psi) - (f, \psi) - \langle g, \psi\rangle. \tag{5.4}$$

Similarly, we can also derive the loss function by using the a posteriori error estimates for the interface problem. It is easy to know $\nabla u \in H(\mathrm{div}, \Omega_1) \cap H(\mathrm{div}, \Omega_2)$ but $\nabla u \notin H(\mathrm{div}, \Omega)$ [5]. Before deriving the a posteriori error estimate, we introduce the following formula of integration by parts

$$(\mathbf{p}, \nabla v) + (\mathrm{div}\mathbf{p}, v) = \langle[\mathbf{n} \cdot \mathbf{p}], v\rangle, \quad \forall \mathbf{p} \in H(\mathrm{div}, \Omega_1) \cap H(\mathrm{div}, \Omega_2), \quad \forall v \in H_0^1(\Omega). \tag{5.5}$$

In order to measure the relation between two norms $\|\cdot\|_{0,\Gamma}$ and $\|\cdot\|_a$, we define a Steklov eigenvalue problem: Find $(\lambda_\Gamma, u) \in \mathbb{R} \times H_0^1(\Omega)$ such that

$$\begin{cases} -\nabla \cdot (\alpha \nabla u) + \beta u &= 0, \quad \text{in } \Omega_1 \cup \Omega_2, \\ u &= 0, \quad \text{on } \partial\Omega, \\ [u] = 0, \quad \left[\alpha \frac{\partial u}{\partial \mathbf{n}}\right] &= \lambda_\Gamma u, \quad \text{across } \Gamma. \end{cases} \tag{5.6}$$

The corresponding weak form is defined as follows: Find $(\lambda_\Gamma, u) \in \mathbb{R} \times H_0^1(\Omega)$ such that

$$a(u, v) = \lambda_\Gamma \langle u, v\rangle, \quad \forall v \in H_0^1(\Omega). \tag{5.7}$$

It is well-known that eigenvalue problem (5.7) has the eigenvalue series

$$0 < \lambda_{1,\Gamma} \le \lambda_{2,\Gamma} \le \cdots \lambda_{km\Gamma} \le \cdots \to +\infty.$$

Then we have the following inequality

$$\frac{\|v\|_a^2}{\|v\|_{0,\Gamma}^2} \ge \lambda_{1,\Gamma} \quad \forall 0 \ne v \in H_0^1(\Omega). \tag{5.8}$$

Theorem 5.1. *Assume $u \in V$ is the exact solution of (5.2) and the trial function $u_N \in H_0^1(\Omega)$. The following upper bound holds*

$$\|u - u_N\|_a \leq \sqrt{1 + \frac{1}{\lambda_{\min}} + \frac{1}{\lambda_{1,\Gamma}}}\eta(u_N, \mathbf{p}), \quad \forall \mathbf{p} \in \bigcap_{i=1}^{2} H(\mathrm{div}; \Omega_i), \tag{5.9}$$

where $\eta(u_p, \mathbf{p})$ is defined as follows

$$\eta(u_N, \mathbf{p}) := \left(\|f - \beta u_N + \mathrm{div}\mathbf{p}\|_0^2 + \|\alpha^{-\frac{1}{2}}(\mathbf{p} - \alpha\nabla u_N)\|_0^2 + \|g - [\mathbf{n}\cdot\mathbf{p}]\|_{0,\Gamma}^2\right)^{\frac{1}{2}}. \tag{5.10}$$

Proof. Combining (5.2), (5.5) and (5.8), for any function $u_N \in H_0^1(\Omega)$, we can define $w = u - u_N \in V$, and the following inequalities hold for any $\mathbf{p} \in H(\mathrm{div}, \Omega_1) \cap H(\mathrm{div}, \Omega_2)$

$$\begin{aligned}
&\|u - u_N\|_a^2 = a(u - u_N, w) = (f, w) + \langle g, w\rangle - a(u_N, w)\\
&= (f, w) + \langle g, w\rangle - (\alpha\nabla u_N, \nabla w) - (\beta u_n, w) + (\mathbf{p}, \nabla w) + (\mathrm{div}\mathbf{p}, w) - \langle[\mathbf{n}\cdot\mathbf{p}], w\rangle\\
&= (f - \beta u_N + \mathrm{div}\mathbf{p}, w) + (\mathbf{p} - \alpha\nabla u_N, \nabla w) + \langle g - [\mathbf{n}\cdot\mathbf{p}], w\rangle\\
&\leq \|f - \beta u_N + \mathrm{div}\mathbf{p}\|_0\|w\|_0 + \|\alpha^{-\frac{1}{2}}(\mathbf{p} - \alpha\nabla u_N)\|_0\|\alpha^{\frac{1}{2}}\nabla w\|_0\\
&\quad + \|g - [\mathbf{n}\cdot\mathbf{p}]\|_{0,\Gamma}\|w\|_{0,\Gamma}\\
&\leq \left(\|f - \beta u_N + \mathrm{div}\mathbf{p}\|_0^2 + \|\alpha^{-\frac{1}{2}}(\mathbf{p} - \alpha\nabla u_N)\|_0^2 + \|g - [\mathbf{n}\cdot\mathbf{p}]\|_{0,\Gamma}^2\right)^{\frac{1}{2}}\\
&\quad \times \left(\|w\|_0^2 + \|\alpha^{\frac{1}{2}}\nabla w\|_0^2 + \|w\|_{0,\Gamma}^2\right)^{\frac{1}{2}}\\
&\leq \left(\|f - \beta u_N + \mathrm{div}\mathbf{p}\|_0^2 + \|\alpha^{-\frac{1}{2}}(\mathbf{p} - \alpha\nabla u_N)\|_0^2 + \|g - [\mathbf{n}\cdot\mathbf{p}]\|_{0,\Gamma}^2\right)^{\frac{1}{2}}\\
&\quad \times \left(1 + \frac{1}{\lambda_{\min}} + \frac{1}{\lambda_{1,\Gamma}}\right)^{\frac{1}{2}}\|w\|_a,
\end{aligned} \tag{5.11}$$

where $\lambda_{\min}$ is the constant in the inequality (3.7), $\lambda_{1,\Gamma}$ is the minimum eigenvalue of (5.7).

Then based on (5.11), the following inequality holds for any $\mathbf{p} \in \cap_{i=1}^{n} H(\mathrm{div}, \Omega_i)$

$$\begin{aligned}
\|u - u_N\|_a^2 \leq &\left(1 + \frac{1}{\lambda_{\min}} + \frac{1}{\lambda_{1,\Gamma}}\right)\Big(\|f - \beta u_N + \mathrm{div}\mathbf{p}\|_0^2 + \|\alpha^{-\frac{1}{2}}(\mathbf{p} - \alpha\nabla u_N)\|_0^2\\
&+ \|g - [\mathbf{n}\cdot\mathbf{p}]\|_{0,\Gamma}^2\Big).
\end{aligned}$$

This is the desired result (5.9) and the proof is complete. □

The trial function u_{NN} by the NN functions $\Psi(x; c, \theta)$ and $\Psi_1(x; c_1, \theta_1)$ (3.2) are defined as the following way

$$u_{\mathrm{NN}} = \omega(x)\Psi(x; c, \theta) + \gamma(x)\Psi_1(x; c_1, \theta_1), \tag{5.12}$$

where $\Psi(x; c, \theta)$ and $\Psi_1(x; c_1, \theta_1)$ are two NN functions which are defined on Ω and Ω_1, respectively. Then it is easy to know that $[u_{\mathrm{NN}}] = 0$ and $u_{\mathrm{NN}} \in H_0^1(\Omega)$. But $\alpha\nabla u_{\mathrm{NN}}$ is discontinuous across the interface Γ since the supporting set of Ψ_1 is only Ω_1. Then we only have $\alpha\nabla u_{\mathrm{NN}} \in H(\mathrm{div}, \Omega_1) \cap H(\mathrm{div}, \Omega_2)$.

With the help of $\alpha\nabla u_{\text{NN}} \in H(\text{div}, \Omega_1) \cap H(\text{div}, \Omega_2)$, we can choose $\mathbf{p} = \alpha\nabla u_{\text{NN}}$ and the following a posteriori error estimate holds

$$\|u - u_{\text{NN}}\|_a^2 \leq \left(1 + \frac{1}{\lambda_{\min}} + \frac{1}{\lambda_{1,\Gamma}}\right) \left(\|f - \beta u_{\text{NN}} + \nabla \cdot (\alpha \nabla u_{\text{NN}})\|_0^2 + \|g - [\mathbf{n} \cdot (\alpha \nabla u_{\text{NN}})]\|_{0,\Gamma}^2\right). \quad (5.13)$$

Based on the a posteriori error estimate, the loss function can be defined as

$$\mathcal{L}(u_{\text{NN}}) := \eta^2(u_{\text{NN}}, \alpha\nabla u_{\text{NN}}) = \|f - \beta u_{\text{NN}} + \nabla \cdot (\alpha \nabla u_{\text{NN}})\|_0^2 + \|g - [\mathbf{n} \cdot (\alpha \nabla u_{\text{NN}})]\|_{0,\Gamma}^2 \quad (5.14)$$

Then we can design the NN-based machine learning method by using Algorithm 3 with the loss function defined by (5.14) for solving interface problem (5.1).

5.1 Typical two-material diffusion problems

Consider the following diffusion problem with two materials in the computational domain:

$$\begin{cases} -\nabla \cdot (\alpha \nabla u) = f, & \text{in } \Omega_1 \cup \Omega_2, \\ u = 0, & \text{on } \partial\Omega, \\ [u] = 0, \quad \left[\alpha \frac{\partial u}{\partial \mathbf{n}}\right] = g, & \text{across } \Gamma \end{cases} \quad (5.15)$$

where $\Omega = (0,1) \times (0,1)$, the discontinuous coefficient α is defined as

$$\alpha(x_1, x_2) = \begin{cases} \alpha_1, & (x_1, x_2) \in \left(0, \frac{2}{3}\right] \times (0,1) =: \Omega_1, \\ \alpha_2, & (x_1, x_2) \in \left(\frac{2}{3}, 1\right) \times (0,1) =: \Omega_2. \end{cases}$$

The exact solution of this equation is

$$u_*(x_1, x_2) = \begin{cases} c_1 \sin(k_{11}\pi x_1) \sin(k_{12}\pi x_2), & (x_1, x_2) \in \left(0, \frac{2}{3}\right] \times (0,1), \\ c_2 \sin(k_{21}\pi x_1) \sin(k_{22}\pi x_2), & (x_1, x_2) \in \left(\frac{2}{3}, 1\right) \times (0,1). \end{cases} \quad (5.16)$$

The source term f and the flux jump g across the interface can be derived from the exact solution.

First, we construct the approximation function u_{NN}. According to the discontinuity of the diffusion coefficient, the computational domain is divided into two subdomains according to the interface: $\Omega_1 := \left(0, \frac{2}{3}\right) \times (0,1)$, $\Omega_2 := \left(\frac{2}{3}, 1\right) \times (0,1)$. We define three NNs $u_{N,1}$, $u_{N,2}$, $u_{N,3}$ in the form of (3.2) on the subdomains Ω_1, Ω_2, and the entire domain Ω, respectively. By multiplying these networks with different boundary functions, we obtain $\widehat{u}_{N,1}$, $\widehat{u}_{N,2}$, $\widehat{u}_{N,3}$, which satisfy the Dirichlet boundary condition and the continuity condition across the interface:

$$\begin{aligned} \widehat{u}_{N,1} &= x_1 \left(\frac{2}{3} - x_1\right) x_2(1 - x_2) u_{N,1}, \\ \widehat{u}_{N,2} &= \left(x_1 - \frac{2}{3}\right) (1 - x_1) x_2 (1 - x_2) u_{N,2}, \\ \widehat{u}_{N,3} &= x_1(1 - x_1) x_2 (1 - x_2) u_{N,3}. \end{aligned}$$

The final approximation function is given by:

$$u_{\text{NN}} = \widehat{u}_{N,1} + \widehat{u}_{N,2} + \widehat{u}_{N,3}.$$

It follows that:

$$u_{\mathrm{NN}} = \begin{cases} \widehat{u}_{N,1} + \widehat{u}_{N,3}, & (x_1, x_2) \in \left(0, \frac{2}{3}\right] \times (0,1), \\ \widehat{u}_{N,2} + \widehat{u}_{N,3}, & (x_1, x_2) \in \left(\frac{2}{3}, 1\right) \times (0,1). \end{cases}$$

Next, we perform numerical experiments using the Ritz type of loss function (5.3) and the one by the a posteriori error estimator (5.14).

5.1.1 Testing the loss functions

In this subsection, we investigate the performance of two types of loss functions. Specifically, we test the performance of the adaptive NN subspace based machine learning method using the Ritz loss function (5.3) and the a posteriori error estimator-based loss function (5.14).

The parameters for problem (5.15) are set as follows: the true solution parameters are $c_1 = c_2 = 1$, $k_{11} = 1$, $k_{12} = 2$, $k_{21} = 4$, and $k_{22} = 2$. We test the numerical performance under two different material coefficient settings:

- Test-1-1: $\alpha_1 = 4$, $\alpha_2 = 1$;
- Test-1-2: $\alpha_1 = 4$, $\alpha_2 = 2$.

In Test-1-1, the flux across the interface is continuous, whereas in Test-1-2, the flux jump across the interface is non-zero. For both tests, we use TNNs to construct the NN space. Each sub-network in the TNN consists of a FNN with three hidden layers, each of width 50, and employs $\sin(x)$ as the activation function.

For integration in subdomains Ω_1 and Ω_2, the one-dimensional regions are divided into 100 subintervals, with each subinterval using 16 Gauss-Lobatto quadrature points. During training, the Adam optimizer with a learning rate of 0.001 is used for the first 5000 training steps, the LBFGS optimizer with a learning rate of 0.1 for the following 100 steps to obtain the final results.

We use the following three metrics to measure the errors between the NN approximation u_{NN} and the exact solution u_*:

1. Relative energy norm error: $e_E := \frac{\|u_{\mathrm{NN}} - u_*\|_a}{\|u_*\|_a}$;
2. Relative L^2 norm error: $e_{L^2} := \frac{\|u_{\mathrm{NN}} - u_*\|_{L^2}}{\|u_*\|_{L^2}}$;
3. Relative error on test points:

$$e_{\mathrm{test}} := \frac{\sqrt{\sum_{k=1}^{K} \left(u_{\mathrm{NN}}(x^{(k)}) - u_*(x^{(k)})\right)^2}}{\sqrt{\sum_{k=1}^{K} \left(u_*(x^{(k)})\right)^2}},$$

where the test points $\{x^{(k)}, k = 1, \ldots, K\}$ are uniformly distributed over $[0,1]^2$ with a resolution of 301×301 grid points.

Table 1: Numerical results using the Ritz-type loss function

	e_E	e_{L^2}	e_{test}
Test-1-1	1.379726e-06	1.514646e-07	1.473214e-07
Test-1-2	3.571384e-07	1.642144e-07	4.6365045e-08

Table 2: Numerical results using the a posteriori error estimator-based loss function

	e_E	e_{L^2}	e_{test}
Test-1-1	4.442695e-08	1.003098e-08	1.331919e-09
Test-1-2	7.731275e-08	1.089644e-07	8.876892e-09

Tables 1 and 2 present the results of solving the problem using the two loss functions. From Tables 1 and 2, it can be observed that the machine learning method based on NN spaces can achieve high accuracy in solving this problem. The continuity of the flux across the interface does not significantly affect the performance of our algorithm. Moreover, the accuracy for using the a posteriori error estimator-based loss function is slightly higher compared to that by using the Ritz loss function.

For example, using the a posteriori error estimator loss function to solve Test-1-2, Figure 5 shows the numerical results of the machine learning method designed in this section. Figure 6 displays the trajectory of the approximate solution at the material interface ($x = \frac{2}{3}$). We see that the machine learning method provides accurate approximations for the function values and their partial derivatives at the interface.

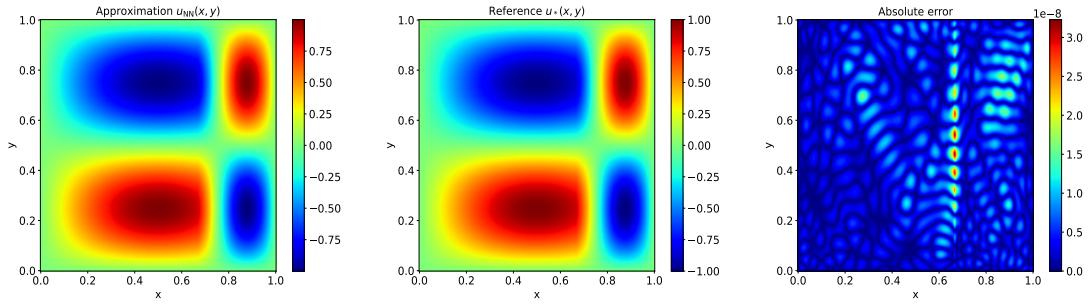


Figure 5: Test-1-2: Numerical results of solving (5.15) using the a posteriori error estimator-based loss function: Left: Approximate solution, Middle: Exact solution, Right: Error distribution.

5.1.2 Tests with large coefficient differences

In this subsection, we will use the machine learning method with the a posteriori error estimator-based loss function to solve the interface problem with large coefficient differences. This is to test the performance of the machine learning method when the material coefficients have significant differences.

We set the true solution parameters of the equation (5.15) as $c_1 = c_2 = 1$, $k_{11} = 1$,

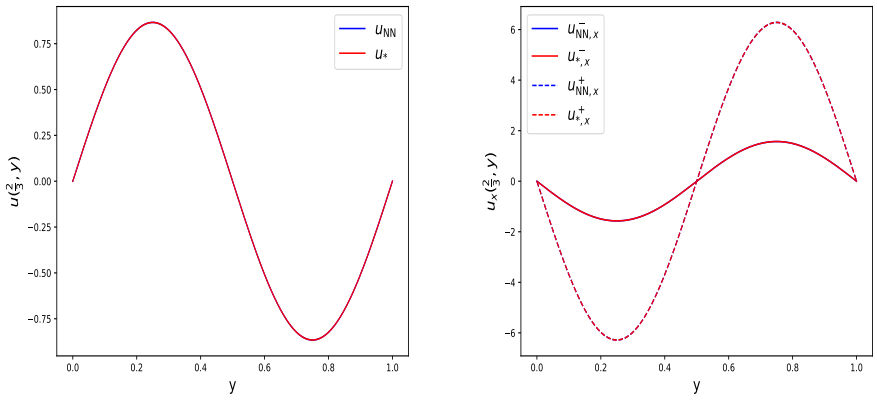


Figure 6: Test-1-2: The approximate solution at the material interface ($x = \frac{2}{3}$) using the a posteriori error estimator-based loss function for (5.15), Left: Function value approximation, Right: Left and right partial derivative approximation.

$k_{12} = 2$, $k_{21} = 4$, $k_{22} = 2$. Here, we test the numerical results for three different material coefficient settings:

- Test-2-1: $\alpha_1 = 4 \times 10^{-4}$, $\alpha_2 = 1$;
- Test-2-2: $\alpha_1 = 4$, $\alpha_2 = 1$;
- Test-2-3: $\alpha_1 = 4000$, $\alpha_2 = 1$.

In Test-2-1 and Test-2-3, the solution exhibits flux jumps at the interface, and there is a large difference in the coefficients on both sides of the interface. In all three tests, we use TNN to construct the NN space. Each TNN has sub-networks selected as FNNs with three hidden layers, each layer containing 50 units, and $\sin(x)$ as the activation function. When integrating over the subregions Ω_1 and Ω_2, we divide the one-dimensional region into 100 subintervals, and for each subinterval, we use 16 Gauss-Lobatto integration points. During training, we first use the Adam optimizer with a learning rate of 0.001 for the first 5000 training steps, and then use the LBFGS optimizer with a learning rate of 0.1 for the following 100 iterations to obtain the final result.

Table 3 presents the numerical errors for the three coefficient settings. It can be observed that the difficulty of solving the equation increases when there is a large coefficient difference, but the algorithm still achieves high accuracy. Figure 7 shows the error distribution for the three coefficient settings. From this, we can see that when $\alpha_1 = 4$, the error distribution is relatively uniform. However, when the coefficient difference is large, in the region with the smaller coefficient, the ellipticity of the equation is poorer, leading to larger solution errors, which is consistent with theoretical expectations.

5.1.3 Tests with high-frequency exact solution

Then, in this subsection, we test the performance and stability of the machine learning method when the exact solution contains high-frequency components. For this purpose,

Table 3: numerical results for different coefficient differences

	e_E	e_{L^2}	e_{test}
Test-2-1: $\alpha_1 = 4 \times 10^{-4}$	7.546526e-07	4.964822e-06	4.923701e-06
Test-2-2: $\alpha_1 = 4$	4.442695e-08	1.003098e-08	1.331919e-09
Test-2-3: $\alpha_1 = 4000$	6.601858e-08	3.747886e-07	3.731660e-07

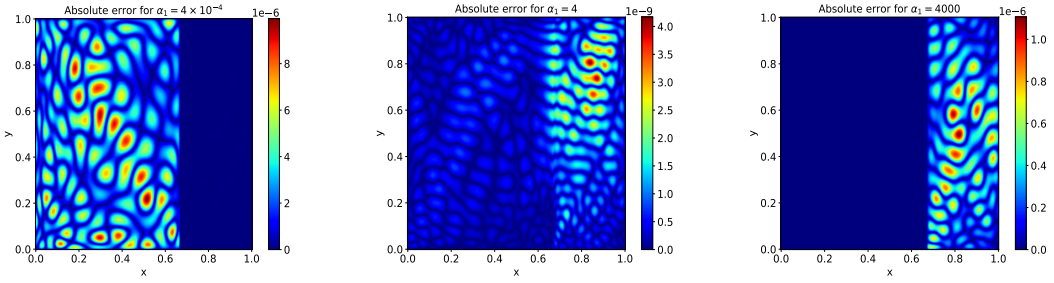


Figure 7: Error distribution for the machine learning method with three coefficient settings. Left, middle, and right correspond to $\alpha_1 = 4 \times 10^{-4}$, 4, and 4000, respectively.

we consider the following equation:

$$\begin{cases} -\nabla \cdot (\alpha \nabla u) &= f, & \text{in } \Omega_1 \cup \Omega_2, \\ u &= 0, & \text{on } \partial\Omega, \\ [u] = 0, \quad \left[\alpha \frac{\partial u}{\partial \mathbf{n}}\right] &= g, & \text{across } \Gamma, \end{cases} \tag{5.17}$$

where $\Omega = (0,1) \times (0,1)$, and the discontinuous coefficient α is defined as follows

$$\alpha(x_1, x_2) = \begin{cases} 10, & (x_1, x_2) \in \left(0, \frac{2}{9}\right] \times (0,1) =: \Omega_1, \\ 2, & (x_1, x_2) \in \left(\frac{2}{9}, 1\right) \times (0,1) =: \Omega_2, \end{cases}$$

The exact solution of the equation is

$$u_*(x_1, x_2) = \begin{cases} \sin(\pi x_1)\sin(2\pi x_2), & (x_1, x_2) \in \left(0, \frac{2}{9}\right] \times (0,1), \\ \sin(10\pi x_1)\sin(2\pi x_2), & (x_1, x_2) \in \left(\frac{2}{9}, 1\right) \times (0,1). \end{cases} \tag{5.18}$$

The corresponding right-hand side f and the flux jump g at the interface can be derived from the exact solution. It can be seen from the definition, the exact solution has high-frequency components in the right region. We label this test as Test-3-1.

We adopt the same approach as in solving (5.15), defining three NNs for the two subregions and the entire region, and using corresponding boundary factor functions to enforce Dirichlet boundary conditions and interface continuity conditions. The TNN is used to construct the NN space, selecting FNNs with four hidden layers as sub-networks, each with 50 units per layer, and $\sin(x)$ as the activation function.

The integration method and optimization parameters are exactly the same as those used in Test-1-1 in Section 5.1.1, and we test both loss functions. Table 4 shows the corresponding numerical errors. It can be seen that the machine learning method here can effectively capture the high-frequency information.

Figure 8 shows the relative energy error and relative L^2 error during the training process. Along the steps correspond to the learning process of finding the optimal subspace, the machine learning method appears stable. Figure 9 presents the error distribution obtained using the a posteriori error estimator-based loss function.

Table 4: Numerical results for high-frequency exact solutions using machine learning method

	e_E	e_{L^2}	e_{test}
Ritz-type	4.190128e-06	1.359826e-06	1.399258e-06
a posteriori error estimator-type:	3.702056e-07	2.251677e-07	1.690076e-07

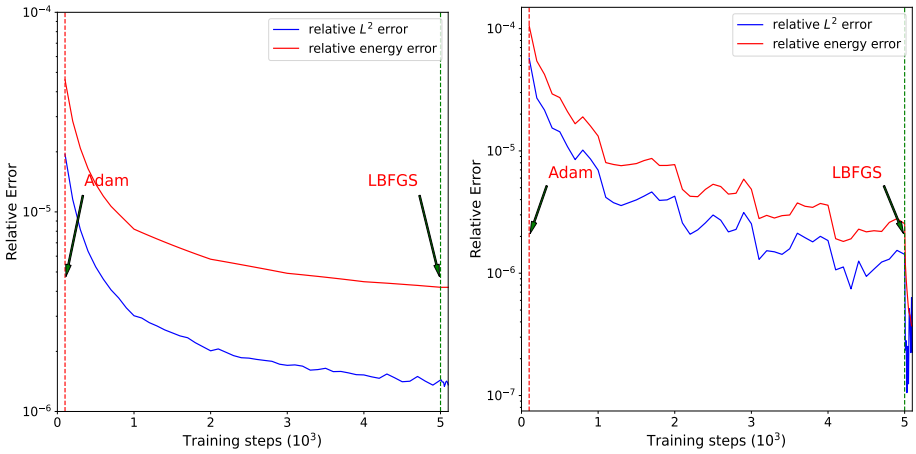


Figure 8: Test-3-1: Relative energy error and relative L^2 error variation during the training process of solving (5.15) with two loss functions. Left: Ritz-type, right: The a posteriori error estimator-type.

5.2 Multi-material diffusion problems

In this subsection, we test the machine learning proposed in this paper through the multi-material diffusion problem. For this aim, we consider the following interface problem:

$$\begin{cases} -\nabla\cdot(\alpha\nabla u) = f, & \text{in } \bigcup_{i=1}^{4}\Omega_i, \\ u = 0, & \text{on } \partial\Omega, \\ [u]=0, \quad \left[\alpha\frac{\partial u}{\partial \mathbf{n}}\right] = g, & \text{across } \Gamma, \end{cases} \tag{5.19}$$

where the computational domain $\Omega = (-1,1)\times(-1,1)$ consists of 4 subdomains (see Figure 10)

$$\Omega_1 := (-1,0)\times(-1,0), \quad \Omega_2 := (0,1)\times(-1,0),$$
$$\Omega_3 := (-1,0)\times(0,1), \quad \Omega_4 := (0,1)\times(0,1).$$

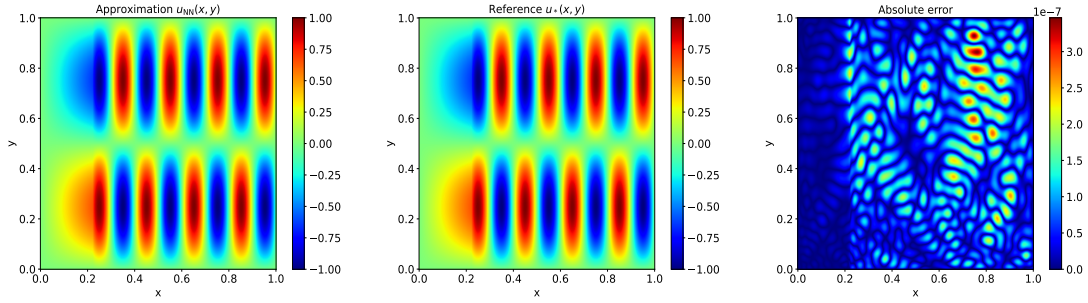


Figure 9: Test-3-1: Error distribution obtained by solving (5.15) with the a posteriori error estimator-type loss function.

The diffusion coefficient is a piecewise constant over the four subdomains:

$$\alpha(x_1, x_2) = \begin{cases} \alpha_1, & (x_1, x_2) \in (-1, 0] \times (-1, 0], \\ \alpha_2, & (x_1, x_2) \in (0, 1) \times (-1, 0], \\ \alpha_3, & (x_1, x_2) \in (-1, 0] \times (0, 1), \\ \alpha_4, & (x_1, x_2) \in (0, 1) \times (0, 1). \end{cases} \tag{5.20}$$

Here, we set the exact solution as

$$u_*(x_1, x_2) = \begin{cases} c_1 \sin(k_{11}\pi x_1) \sin(k_{12}\pi x_2), & (x_1, x_2) \in [-1, 0] \times [-1, 0], \\ c_2 \sin(k_{21}\pi x_1) \sin(k_{22}\pi x_2), & (x_1, x_2) \in (0, 1] \times [-1, 0], \\ c_3 \sin(k_{31}\pi x_1) \sin(k_{32}\pi x_2), & (x_1, x_2) \in [-1, 0] \times (0, 1], \\ c_4 \sin(k_{41}\pi x_1) \sin(k_{42}\pi x_2), & (x_1, x_2) \in (0, 1] \times (0, 1]. \end{cases} \tag{5.21}$$

The source term $f(x_1, x_2)$ and the flux jump g are derived from the exact solution (5.21).

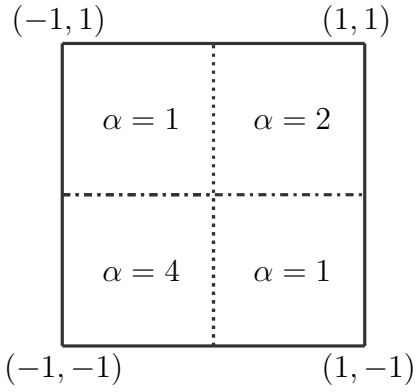


Figure 10: The 4 subdomains and piecewise diffusion constants α.

First, we construct the approximation function u_{NN}. Consistent with the approach in the previous subsections, we define NNs $u_{N,1}$, $u_{N,2}$, $u_{N,3}$, and $u_{N,4}$ in the form of (3.2) on the four subregions. Since the solution of the equation belongs to $H_0^1(\Omega)$, we enforce the NNs on the subregion boundaries to be zero to ensure the continuity of the solution.

Next, we define NNs $u_{N,5}$, $u_{N,6}$, $u_{N,7}$, and $u_{N,8}$ on the regions $\Omega_5 := \Omega_1 \cup \Omega_2$, $\Omega_6 := \Omega_1 \cup \Omega_3$, $\Omega_7 := \Omega_2 \cup \Omega_4$, and $\Omega_8 := \Omega_3 \cup \Omega_4$, respectively, and enforce them to be zero on

their respective boundaries, ensuring that the approximation function u_{NN} has degrees of freedom on the internal boundaries.

Finally, we define the NN $u_{N,9}$ on the region Ω with a boundary condition of zero to ensure the approximation function has degrees of freedom at the point $(0,0)$.

As an example, for the NN $u_{N,5}$ defined on Ω_5, we multiply it by the boundary factor function $(1+x_1)(1-x_1)x_2(1+x_2)$ to obtain $\widehat{u}_{N,5}$, ensuring that it is zero on the subregion boundaries.

Finally, we define the approximation function for the equation as the sum of these locally supported NNs, i.e.,

$$u_{\text{NN}} = \sum_{j=1}^{9} \widehat{u}_{N,j}.$$

It is easy to verify that u_{NN} satisfies the Dirichlet boundary condition and the continuity of the solution across the interface, and belongs to $H_0^1(\Omega)$.

Below, we perform numerical testing using only the Ritz-type loss function (5.3).

5.2.1 Test for continuous interface flux

We perform a numerical test for the case of continuous interface flux, denoted as Test-4-1. For this, we set the diffusion coefficients on the four subregions in (5.20) as $\alpha_1 = 4$, $\alpha_2 = \alpha_3 = 1$, $\alpha_4 = 2$, and set the frequency parameters in the exact solution (5.21) as $k_{ij} = 1$, for $i = 1, \ldots, 4$, $j = 1, 2$, and the coefficients $c_1 = 1$, $c_2 = c_3 = 4$, $c_4 = 2$. It is easy to verify that with these parameter settings, the flux is continuous at the interface.

We use TNNs to construct the NN space, where each TNN's subnetwork is selected as an FNN with 4 hidden layers, each with a width of 50. The rank of the TNN is set to be $p = 100$, and we use $\sin(x)$ as the activation function. In each subregion, the TNN function is integrated by dividing the 1D region into 100 subintervals, with 16 Gauss-Lobatto integration points per subinterval.

For the training process, the optimization parameters are consistent with those used in the two-material diffusion problem. We first use the Adam optimizer with a learning rate of 0.001 for the first 5000 steps, followed by the LBFGS optimizer with a learning rate of 0.1 for 100 steps to obtain the final result. The uniform distributed 401×401 points on $[-1,1]^2$ are chosen as the test points.

Figure 11 shows the approximate solution and its error distribution obtained by the machine learning method, while Table 5 provides the errors between the approximation and the exact solution. From these results, it can be observed that the machine learning method proposed in this paper can achieve high accuracy for this problem.

Table 5: Numerical results of the machine learning method for Test-4-1

e_E	e_{L^2}	e_{test}
5.348647e-08	5.428252e-08	5.683549e-09

This example is commonly used to test the performance of different machine learning method for solving multi-material diffusion equations. Table 6 shows the comparison of the results of PINNs with that of the proposed method in this paper. From this table, it

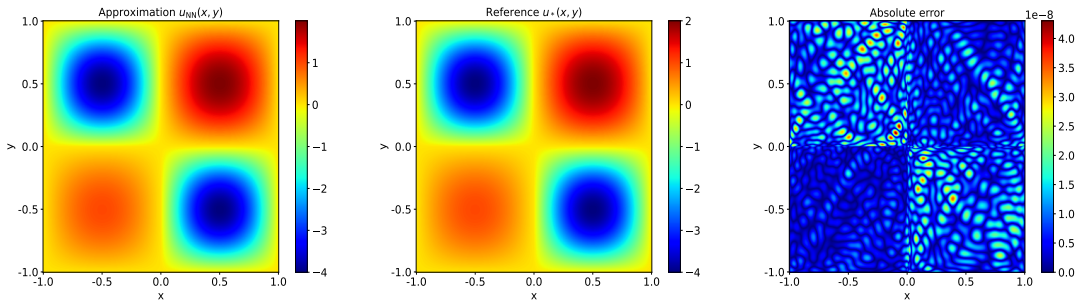


Figure 11: Test-4-1: Numerical results for solving (5.19) using the Ritz-type loss function. The left image is the approximate solution, the middle image is the exact solution, and the right image is the error distribution.

can be seen that the machine learning method designed in this paper achieves the highest accuracy for this problem.

Table 6: Comparison of error in solving the problem (5.19) in Test-4-1 using different machine learning methods

Method	e_{test}
Standard PINN [38]	8.96e-01
nDS-PINN [38]	5.20e-04
HC-PSNN [36]	9.00e-06
Proposed Method	5.68e-09

5.2.2 Test for jump in interface flux

We perform a numerical test for the case of a jump in the interface flux, denoted as Test-4-2. For this aim, we set the diffusion coefficients on the four subregions in (5.20) as $\alpha_1 = 4$, $\alpha_2 = \alpha_3 = 1$, $\alpha_4 = 10$, and set the frequency parameters in the exact solution (5.21) as $(k_{11}, k_{12}) = (3, 1)$, $(k_{21}, k_{22}) = (1, 2)$, $(k_{31}, k_{32}) = (1, 4)$, $(k_{41}, k_{42}) = (2, 2)$, with coefficients $c_1 = 2$, $c_2 = 5$, $c_3 = 2$, $c_4 = 4$. From this exact solution, it can be seen that this is an anisotropic problem with a flux jump at the interface.

We use the same NN structure, integration point configuration, and optimization settings as in Test-4-1. Figure 12 shows the approximate solution obtained by the machine learning here and the corresponding error distribution, while Table 7 provides the errors between the approximation and the exact solution. From these results, it can be seen that the machine learning method designed in this paper also achieves high accuracy for this anisotropic problem with a jump in the interface flux.

Figure 13 shows the behaviors in the relative energy error and the relative L^2 error during the training steps. From this, we can see that the training process is a relatively stable subspace learning process.

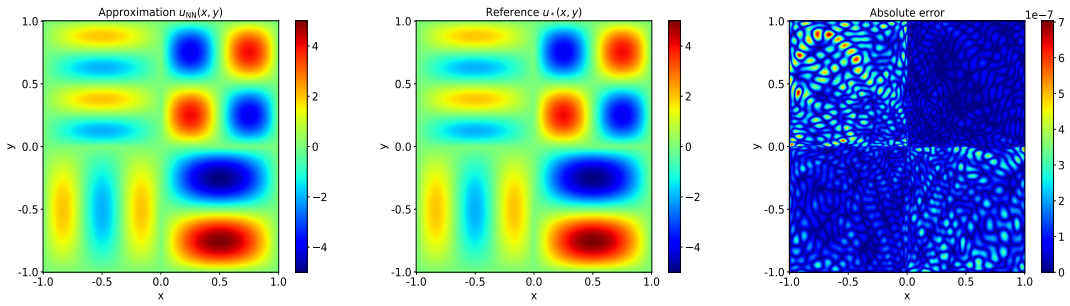


Figure 12: Test-4-2: Error distribution for solving problem (5.19) using the Ritz-type loss function.

Table 7: Numerical results of the machine learning method for Test-4-2

e_E	e_{L^2}	e_{test}
3.450140e-07	7.798271e-08	6.912460e-08

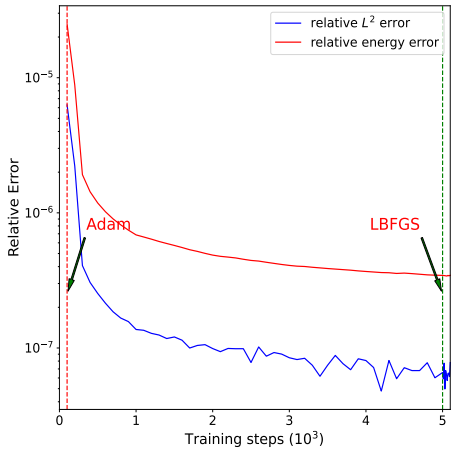


Figure 13: Test-4-2: Behaviors of the relative energy error and relative L^2 error during the training process when solving equation (5.19).

5.3 A heterogeneous material problem within the domain

In this section, we consider the case where a material is completely enveloped by another, which occurs commonly in heat transfer and oil reservoir simulation. Consider the problem as follows:

$$\begin{cases} -\nabla\cdot(\alpha(x_1,x_2)\nabla u) &= Q(x_1,x_2), & (x_1,x_2)\in\Omega_1\cup\Omega_2,\\ u(x_1,\pm 2) &= \sin\left(\frac{\pi}{4}\left(x_1^2+3\right)\right), & x_1\in[-2,2],\\ u(\pm 2,x_2) &= \sin\left(\frac{\pi}{4}\left(x_2^2+3\right)\right), & x_2\in[-2,2],\\ [u]=0,\quad \left[\alpha\frac{\partial u}{\partial \mathbf{n}}\right] &= 0, & \text{across } \Gamma, \end{cases} \tag{5.22}$$

where $\Omega_1 := \{(x_1, x_2) \mid x_1^2 + x_2^2 < 1\}$, $\Omega := (-2, 2) \times (-2, 2)$, $\Omega_2 := \Omega \backslash \overline{\Omega}_1$, $\Gamma := \{(x_1, x_2) \mid x_1^2 + x_2^2 = 1\}$, and

$$\alpha(x_1, x_2) = \begin{cases} 1, & (x_1, x_2) \in \Omega_1, \\ 4, & (x_1, x_2) \in \Omega_2, \end{cases}$$

and the source term

$$Q(x_1, x_2) = \begin{cases} -4\pi \cos\left(\pi\left(x_1^2 + x_2^2 - 1\right)\right) + 4\pi^2\left(x_1^2 + x_2^2\right) \sin\left(\pi\left(x_1^2 + x_2^2 - 1\right)\right), \\ \qquad\qquad (x_1, x_2) \in \Omega_1, \\ -4\pi \cos\left(\frac{\pi}{4}\left(x_1^2 + x_2^2 - 1\right)\right) + \pi^2\left(x_1^2 + x_2^2\right) \sin\left(\frac{\pi}{4}\left(x_1^2 + x_2^2 - 1\right)\right), \\ \qquad\qquad (x_1, x_2) \in \Omega_2. \end{cases}$$

Then the exact solution of this problem is

$$u_*(x_1, x_2) = \begin{cases} \sin\left(\pi\left(x_1^2 + x_2^2 - 1\right)\right), & (x_1, x_2) \in \Omega_1, \\ \sin\left(\frac{\pi}{4}\left(x_1^2 + x_2^2 - 1\right)\right), & (x_1, x_2) \in \Omega_2. \end{cases} \tag{5.23}$$

To deal with the non-homogeneous Dirichlet boundary condition, following the idea from finite element method, we set up a NN b_N to fit the boundary function $b(x_1, x_2)$ on $\partial\Omega$, which is defined as

$$b(x_1, x_2) = \begin{cases} \sin\left(\frac{\pi}{4}\left(x_1^2 + 3\right)\right), & (x_1, x_2) \in [-2, 2] \times \{\pm 2\}, \\ \sin\left(\frac{\pi}{4}\left(x_2^2 + 3\right)\right), & (x_1, x_2) \in \{\pm 2\} \times [-2, 2]. \end{cases}$$

Then, by introducing φ as $\varphi = u - b_N$, we transform the original problem (5.22) into the homogeneous-Dirichelet-boundary one:

$$\begin{cases} -\nabla \cdot (\alpha(x_1, x_2)\nabla\varphi) &= Q(x_1, x_2) + \nabla \cdot (\alpha(x_1, x_2)\nabla b_N), & (x_1, x_2) \in \Omega_1 \cup \Omega_2, \\ \varphi(x_1, x_2) &= 0, & (x_1, x_2) \in \partial\Omega, \\ [\varphi] = 0, \quad \left[\alpha\frac{\partial\varphi}{\partial \mathbf{n}}\right] &= -\left[\alpha\frac{\partial b_N}{\partial \mathbf{n}}\right] := g, & \text{across } \Gamma. \end{cases} \tag{5.24}$$

Just as (5.1), we define the corresponding variational form for (5.24) as follows: Find $\varphi \in H_0^1(\Omega)$ such that

$$a(\varphi, v) = (f, v), \quad \forall v \in H_0^1(\Omega),$$

where $a(\cdot, \cdot), (\cdot, \cdot)$ are defined as follows

$$\begin{aligned} a(\varphi, v) &= \int_\Omega (\nabla v \cdot \alpha \nabla \varphi)\, d\Omega, \\ (f, v) &= \int_\Omega Q v d\Omega - \int_\Omega (\nabla v \cdot \alpha \nabla b_N)\, d\Omega + \int_\Gamma \left[\alpha\frac{\partial\varphi}{\partial \mathbf{n}}\right] v ds + \int_\Gamma \left[\alpha\frac{\partial b_N}{\partial \mathbf{n}}\right] v ds \\ &= \int_\Omega Q v d\Omega - \int_\Omega (\nabla v \cdot \alpha \nabla b_N)\, d\Omega. \end{aligned}$$

As mentioned in (5.12), we define the trial function φ_N by NN functions Ψ and Ψ_1, which are supported only on Ω and Ω_1, respectively, with the following way

$$\varphi_N = w(x_1, x_2)\Psi + \gamma(x_1, x_2)\Psi_1, \tag{5.25}$$

where $w(x_1, x_2) = (x_1 - 2)(x_1 + 2)(x_2 - 2)(x_2 + 2)$ and $\gamma(x_1, x_2) = x_1^2 + x_2^2 - 1$.

Similarly, the Ritz-type loss function and the a posteriori error estimator-based loss function can be defined as

$$\begin{aligned} \mathcal{L}_R(\varphi_N) &:= \frac{1}{2}a(\varphi_N, \varphi_N) - (f, \varphi_N), \\ \mathcal{L}_P(\varphi_N) &:= \|\nabla \cdot (\alpha \nabla \varphi_N) + Q + \nabla \cdot (\alpha \nabla b_N)\|_0^2 + \|g - [\mathbf{n} \cdot (\alpha \nabla \varphi_N)]\|_{0,\Gamma}^2, \end{aligned} \tag{5.26}$$

respectively. We can then design NN-based machine learning method utilizing Algorithm 3 with the loss functions (5.26) to solve (5.24). The final approximate solution for (5.22) is given by $u_{\text{NN}} := \varphi_N + b_N$.

To interpolate the boundary function $b(x_1, x_2)$ on $\partial\Omega$, we use FNN (cf. Figure (2a)) with $p = 100$ to construct the NN space. Each FNN consists of three hidden layers with a width of 50 and employs $\sin(x)$ as the activation function. The relative $L^2(\partial\Omega)$ error between b_N and $b(x_1, x_2)$,

$$\frac{\|b_N(x_1, x_2) - b(x_1, x_2)\|_{L^2(\partial\Omega)}}{\|b(x_1, x_2)\|_{L^2(\partial\Omega)}}, \tag{5.27}$$

is used as the loss function. For the boundary integration in (5.27) over the 4 edges of $\partial\Omega$, the one-dimensional region is divided into 100 subintervals, and 16 Gauss-Lobatto quadrature points are employed. The Adam optimizer with a learning rate of 0.001 and a multi-step learning rate scheduler is utilized for 50,000 training steps. The final relative $L^2(\partial\Omega)$ error between b_N and $b(x_1, x_2)$ is 4.162097e-07.

To solve (5.24) with homogeneous Dirichlet boundary condition, we employ FNN (cf. Figure (2a)) with $p = 100$ to construct the NN subspaces. The corresponding NN functions Ψ and Ψ_1 are used to define the trial function in (5.25). Each FNN consists of three hidden layers with a width of 50 and utilizes $\sin(x)$ as the activation function. The a posteriori error estimator-based loss function (5.26) is employed to update the NN subspace.

For the two-dimensional integration in the loss function (5.26), we design different quadrature schemes for computing the integration over Ω_1, Ω_2, and Γ. Let $\mathcal{F}(\varphi_N)$ denote the integrand. We use polar coordinate transformation to compute the integration over Ω_1 and Γ:

$$\int_{\Omega_1} \mathcal{F}(\varphi_N) d\Omega_1 = \int_0^{2\pi} \int_0^1 \mathcal{F}(\varphi_N) r dr d\theta,$$
$$\int_{\Gamma} \mathcal{F}(\varphi_N) ds = \int_0^{2\pi} \mathcal{F}(\varphi_N) d\theta.$$

For the integration with respect to r, the interval $[0, 1]$ is divided into 20 subintervals, and 8 Gauss-Lobatto points are used in each subinterval. For the integration with respect to θ, $N = 160$ equispaced grid points in $[0, 2\pi]$ are used as quadrature points, with each having a quadrature weight of $\frac{1}{N}$. Next, we transform the integration over Ω_2 into the difference of integration over Ω and Ω_1:

$$\int_{\Omega_2} \mathcal{F}(\varphi_N) d\Omega_2 = \int_{\Omega} \mathcal{F}(\varphi_N) d\Omega - \int_{\Omega_1} \mathcal{F}(\varphi_N) d\Omega_1.$$

To compute the integration over Ω, the region $[-2, 2] \times [-2, 2]$ is divided into 20 subintervals for both x_1 and x_2. Eight Gauss-Lobatto points are used in each subinterval. The

two-dimensional quadrature points and their corresponding weights for integration over Ω and Ω_1 are then constructed as the Cartesian product of the one-dimensional quadrature points.

We first use the Adam optimizer with a learning rate of 0.01 for the first 5000 steps, followed by the LBFGS optimizer with a learning rate of 1 for 2000 steps to obtain the final result. The uniform distributed 401×401 points on $[-2, 2] \times [-2, 2]$ are chosen as the test points. The relative energy norm error, relative L^2 norm error and the relative error on test points between the final approximate solution $u_{\text{NN}} := \varphi_N + b_N$ and the exact solution u (5.23) is shown in Table 8. Figure 14 shows the approximate solution and its error distribution obtained by the proposed machine learning method.

Table 8: Numerical results of the machine learning method for problem (5.22)

e_E	e_{L^2}	e_{test}
3.115541e-06	5.595210e-07	5.591040e-07

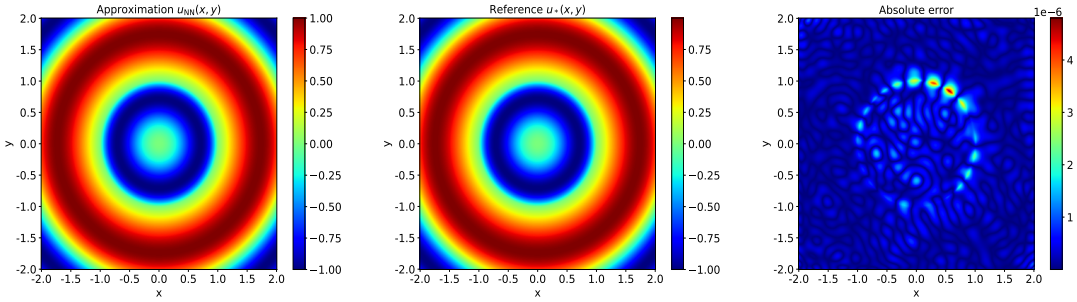


Figure 14: Numerical results for solving (5.22) using the a posteriori error estimator-based loss function. The left image is the approximate solution, the middle image is the exact solution, and the right image is the error distribution.

5.4 Complex shape interface problems

We examine a flower-shaped interface problem [12] in this section to demonstrate the capability of our algorithm in handling geometrically complex interfaces. Let $\Omega = (-1, 1)^2$, and the interface Γ is defined as follows:

$$\Gamma := \left\{ (x_1, x_2) \in \mathbb{R}^2 : r = \frac{1}{2}(1 + \delta \sin(N\theta)), \quad \theta \in [0, 2\pi] \right\},$$

where (r, θ) are the polar coordinates, δ and N are two parameters determining the shape. Figure 15 shows the shape of interfaces for different parameters.

Define Ω_1 and Ω_2 as

$$\Omega_1 = \left\{ (x_1, x_2) \in (-1, 1)^2 : r < \frac{1}{2}(1 + \delta \sin(N\theta)) \right\},$$
$$\Omega_2 = \left\{ (x_1, x_2) \in (-1, 1)^2 : r > \frac{1}{2}(1 + \delta \sin(N\theta)) \right\}.$$

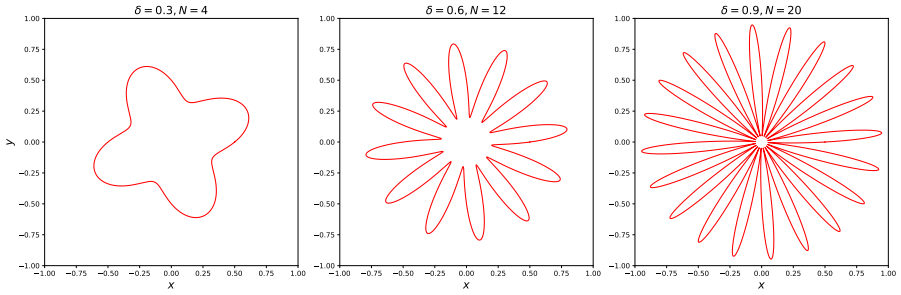


Figure 15: Flower-shaped interfaces for different parameters.

We now consider the following problem:

$$\begin{cases} -\nabla \cdot (\kappa(x_1, x_2)\nabla u(x_1, x_2)) = Q(x_1, x_2), & (x_1, x_2) \in \Omega_1 \cup \Omega_2, \\ u(x_1, x_2) = \phi(x_1, x_2), & (x_1, x_2) \in \partial\Omega, \\ [u(x_1, x_2)] = \Phi(x_1, x_2), & \text{across } \Gamma, \\ [\kappa(x_1, x_2)\nabla u(x_1, x_2) \cdot \mathbf{n}] = \Psi(x_1, x_2), & \text{across } \Gamma, \end{cases} \tag{5.28}$$

where

$$\kappa(x_1, x_2) = \begin{cases} \frac{1}{7}(x_1^2 - x_2^2 + 3), & (x_1, x_2) \in \Omega_1, \\ \frac{1}{5}(x_1 x_2 + 2), & (x_1, x_2) \in \Omega_2. \end{cases}$$

We choose the exact solution to be

$$u_*(x_1, x_2) = \begin{cases} 7x_1^2 + 7x_2^2 + 6, & (x_1, x_2) \in \Omega_1, \\ 5 - 5x_1^2 - 5x_2^2, & (x_1, x_2) \in \Omega_2. \end{cases} \tag{5.29}$$

and $Q(x_1, x_2), \phi(x_1, x_2), \Phi(x_1, x_2)$ and $\Psi(x_1, x_2)$ can be obtained from the exact solution.

Before constructing the trial function u_1^N and u_2^N to approximate the solution on Ω_1 and Ω_2, we first construct the extension $\widehat{\Phi}(x_1, x_2)$ of the function $\Phi(x_1, x_2)$ over the entire domain Ω, and then introduce $\widehat{u}_1^N := u_1^N$ and $\widehat{u}_2^N := u_2^N + \widehat{\Phi}(x_1, x_2)$ to solve the following problem, where the solution is continuous across the interface

$$\begin{cases} -\nabla \cdot (\kappa(x_1, x_2)\nabla \widehat{u}(x_1, x_2)) = \widehat{Q}(x_1, x_2), & (x_1, x_2) \in \Omega_1 \cup \Omega_2, \\ \widehat{u}(x_1, x_2) = \phi(x_1, x_2) + \widehat{\Phi}(x_1, x_2), & (x_1, x_2) \in \partial\Omega, \\ [\widehat{u}(x_1, x_2)] = 0, & \text{across } \Gamma, \\ [\kappa(x_1, x_2)\nabla \widehat{u}(x_1, x_2) \cdot \mathbf{n}] = \Psi(x_1, x_2) - \kappa(x_1, x_2)\nabla \widehat{\Phi}(x_1, x_2) \cdot \mathbf{n}, & \text{across } \Gamma, \end{cases} \tag{5.30}$$

where

$$\widehat{Q}(x_1, x_2) = \begin{cases} Q(x_1, x_2), & (x_1, x_2) \in \Omega_1, \\ Q(x_1, x_2) - \nabla \cdot (\kappa(x_1, x_2)\nabla \widehat{\Phi}(x_1, x_2)), & (x_1, x_2) \in \Omega_2. \end{cases}$$

Next, similarly as Section 5.3, to deal with the non-homogeneous boundary condition on $\partial\Omega$, we set up a NN b_N to fit the boundary function $\phi(x_1, x_2) + \widehat{\Phi}(x_1, x_2)$. Note that

we have extended the boundary function to the entire domain while retaining the same notation for simplicity. And then by introducing φ as $\varphi = \widehat{u} - b_N$, we consider the following problem:

$$\begin{cases} -\nabla \cdot (\kappa(x_1, x_2)\nabla\varphi) = \widehat{Q}(x_1, x_2) + \nabla \cdot (\kappa(x_1, x_2)\nabla b_N), & (x_1, x_2) \in \Omega_1 \cup \Omega_2, \\ \varphi(x_1, x_2) = 0, & (x_1, x_2) \in \partial\Omega, \\ [\varphi(x_1, x_2)] = 0, & \text{across } \Gamma, \\ \left[\kappa(x_1, x_2)\frac{\partial\varphi}{\partial \mathbf{n}}\right] = \Psi(x_1, x_2) - \kappa(x_1, x_2)\frac{\partial\widehat{\Phi}}{\partial \mathbf{n}} - \left[\kappa(x_1, x_2)\frac{\partial b_N}{\partial \mathbf{n}}\right], & \text{across } \Gamma. \end{cases} \tag{5.31}$$

Following up, we conduct the similar procedure as Section 5.3 to define the corresponding variational form, build up the trial functions φ_N by introducing some locally supported NN with some certain boundary condition, and define the Ritz-type loss function and the a posteriori error estimator-based loss function. The NN-based machine learning method can be designed to solve (5.31) in the same way, and the final approximate solution for (5.28) is given by

$$u_{\text{NN}}(x_1, x_2) := \begin{cases} \varphi_N + b_N, & (x_1, x_2) \in \Omega_1, \\ \varphi_N + b_N - \widehat{\Phi}(x_1, x_2), & (x_1, x_2) \in \Omega_2. \end{cases} \tag{5.32}$$

Denote $\rho(\theta) = \frac{1}{2}(1 + \delta \sin(N\theta))$. Now we design different quadrature scheme for the two-dimensional integration in the loss function over Ω_1, Ω_2 and Γ. Let $\mathcal{F}(\varphi_N)$ denote the integrand. Then, we use polar coordinate transformation to compute the integration over Ω_1 and Γ:

$$\int_{\Omega_1} \mathcal{F}(\varphi_N) d\Omega_1 = \int_0^{2\pi} \int_0^{\rho(\theta)} \mathcal{F}(\varphi_N(r\cos\theta, r\sin\theta)) r dr d\theta,$$

$$\int_{\Gamma} \mathcal{F}(\varphi_N) ds = \int_0^{2\pi} \mathcal{F}(\varphi_N(\rho(\theta)\cos\theta, \rho(\theta)\sin\theta)) \sqrt{(\rho(\theta))^2 + \left(\frac{d\rho(\theta)}{d\theta}\right)^2} d\theta.$$

For the integration with respect to θ, $N = 160$ equispaced grid points in $[0, 2\pi]$ are used as quadrature points, with each having a quadrature weight of $\frac{1}{N}$. To generate the two-dimensional quadrature points for the integration over Ω_1, for each grid points θ_i in $[0, 2\pi]$, the interval $[0, \rho(\theta_i)]$ is divided into 20 subintervals, and 8 Gauss-Legendre points are used in each subinterval for the integration with respect to r.

Next, we transform the integration over Ω_2 into the difference of integration over Ω and Ω_1:

$$\int_{\Omega_2} \mathcal{F}(\varphi_N) d\Omega_2 = \int_{\Omega} \mathcal{F}(\varphi_N) d\Omega - \int_{\Omega_1} \mathcal{F}(\varphi_N) d\Omega_1.$$

To compute the integration over Ω, the region $[-1, 1] \times [-1, 1]$ is divided into 20 subintervals for both x_1 and x_2. Eight Gauss-Legendre points are used in each subinterval. The two-dimensional quadrature points and their corresponding weights for integration over Ω are then constructed as the Cartesian product of the one-dimensional quadrature points.

The NN structure, the optimization process and other experiment parameters are the same as Section 5.3 are used to construct the NN subspaces with $p = 50$. The relative

energy norm error, relative L^2 norm error and the relative error on test points between the final approximate solution u_{NN} (5.32) and the exact solution u (5.29) for three shape parameters setting is shown in Table 9. Figure 16 shows the approximate solution and its error distribution obtained by the proposed machine learning method for $\delta = 0.9$ and $N = 20$. These results highlight the potential of the proposed algorithm in handling interface problems with complex-shaped interfaces.

Table 9: Numerical results of the machine learning method for problem (5.28)

	$\delta = 0.3, N = 4$	$\delta = 0.6, N = 12$	$\delta = 0.9, N = 20$
e_E	8.842379e-07	2.958395e-07	1.779185e-06
e_{L^2}	8.105071e-08	2.349744e-08	1.381434e-07
e_{test}	8.100888e-08	2.348698e-08	1.381133e-07

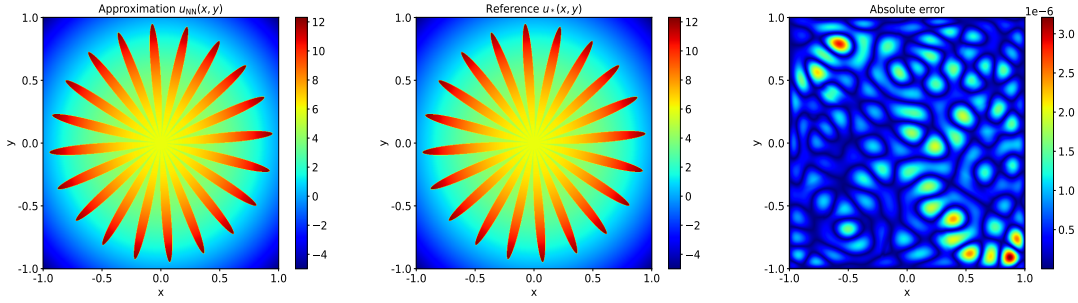


Figure 16: Numerical results for solving (5.28) for shape parameters $\delta = 0.9, N = 20$ using the a posteriori error estimator-based loss function. The left image is the approximate solution, the middle image is the exact solution, and the right image is the error distribution.

Remark 5.1. *Note that the numerical experiments constructed in this work involve exact solutions that are infinitely smooth, which allows us to evaluate the effectiveness of the proposed algorithm more conveniently. For interface problems with simple source terms, finite element approximations can serve as reference solutions. Typically, in such cases, the solution within each subdomain separated by the interface may lack smoothness, with irregularities often occurring near the interface. The high accuracy of the proposed adaptive NN subspace methods reveals that the NN space possesses the strong approximation capability and the corresponding machine learning method with high-precision integration has the ability to automatically handle the singularities in the problems. Consequently, the proposed algorithm shows promise in handling problems where the solution belongs to a function space with limited regularity, which we plan to explore in future work.*

6 Conclusion

In this paper, following the idea of subspace approximation from the finite element method, we provide the error analysis for the NN-based machine learning method for

solving elliptic PDEs. The analysis reveals that integration errors in the loss function computation, approximation errors in the approximate function, and optimization errors in the machine learning method all contribute to the overall error in the approximate solution.

Based on this understanding, we develop an adaptive NN subspace-based machine-learning method that employs numerical quadrature to define the loss functions. The proposed method adaptively updates the trial function space represented by the NN during the training process using an a posteriori error estimator, thereby reducing the subspace approximation error. In addition, we leverage a split optimization process to alternately obtain a better NN subspace and an optimal Galerkin approximation on it, thereby reducing the difficulty of the optimization and controlling the optimization error. Furthermore, we employ high-precision numerical integration rules, which is another key reason why our proposed method achieves high-precision solutions.

This article mainly applies the proposed method to interface problems. Using the NN subspace, we derived the corresponding machine learning method and provided a theoretical analysis of both Ritz-type and the a posteriori error estimation-type loss functions. Numerical experiments demonstrate that our method effectively controls errors and achieves high-precision solutions to interface problems.

Of course, the analysis and algorithms presented here can be extended to broader class of PDEs.

References

[1] F. Cao, X. Guo, F. Gao and D. Yuan, Deep learning nonhomogeneous elliptic interface problems by soft constraint physics-informed neural networks, Mathematics 11 (8) (2023), http://dx.doi.org/10.3390/math11081843.

[2] J. Chen, X. Chi, W. E and Z. Yang, Bridging traditional and machine learning-based algorithms for solving PDEs: the random feature method, J. Mach. Learn., 1 (2022), 268–298.

[3] Z. Chen and Y. Liu, An arbitrarily high order unfitted finite element method for elliptic interface problems with automatic mesh generation, J. Comput. Phys., 491 (2023), 112384.

[4] Z. Chen and Y. Liu, An arbitrarily high order unfitted finite element method for elliptic interface problems with automatic mesh generation, Part II. Piecewise-smooth interfaces, Applied Numerical Mathematics, 206 (2024), 247–268.

[5] Z. Chen and J. Zou, Finite element methods and their convergence for elliptic and parabolic interface problems, Numer. Math., 79 (1998), 175–202.

[6] X. Dai, Y. Fan and Z. Sheng, Subspace method based on neural networks for eigenvalue problems, arXiv:2405.08513v1,2024.

[7] S. Dong, Z. Li, Local extreme learning machines and domain decomposition for solving linear and nonlinear partial differential equations, Comput. Methods Appl. Mech. Engrg., 387 (2021), 114129.

[8] S. Dong, Z. Li, A modified batch intrinsic plascity method for pretraining the random coefficients of extreme learning machines, J. Comput. Phys., 445 (2021), 110585.

[9] W. E and B. Yu, The deep Ritz method: a deep-learning based numerical algorithm for solving variational problems, Commun. Math. Stat., 6 (2018), 1–12.

[10] J. Han, L. Zhang and W. E, Solving many-electron Schrödinger equation using deep neural networks. J. Comput. Phys., 399 (2019), 108929.

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

[12] S. Hou, X.-D. Liu, A numerical method for solving variable coefficient elliptic equation with interfaces, J. Comput. Phys. 202 (2) (2005), 411–445.

[13] W. Huang and R. D. Russell, Adaptive Moving Mesh Methods, Applied Mathematical Sciences, vol 174. Springer, New York, NY, 2010.

[14] Z. Hu, K. Shukla, G. E. Karniadakis and K. Kawaguchi, Tackling the curse of dimensionality with physics-informed neural networks, arXiv:2307.12306v5, 2024.

[15] G. Hu, Z. Qiao, and T. Tang, Moving finite element simulations for reaction-diffusion systems, Adv. Appl. Math. Mech., 4(03) (2012), 365–381.

[16] G. Huang, Q. Zhu, C. Siew, Extreme learning machine: theory and applications, Neurocomputing, 70 (2006), 489–501.

[17] T. Kao, J. Zhao and L. Zhang, pETNNs: partial evolutionary tensor neural networks for solving time-dependent partial differential equations, arXiv:2403.06084v1, 2024.

[18] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv:1412.6980, 2014; Published as a conference paper at ICLR 2015.

[19] J. Li, J. M. Melenk, B. Wohlmuth and J. Zou, Optimal a priori estimates for higher order finite elements for elliptic interface problems, Applied Numerical Mathematics, 60 (2010), 19–37.

[20] R. Li, T. Tang and P. Zhang, Moving mesh methods in multiple dimensions based on harmonic maps, J. Comput. Phys., 170(2) (2001), 562–588.

[21] Y. Li and F. Wang, Local randomized neural networks methods for interface problems, arXiv:2308.03087, 2023.

[22] P. Liu, Z. Xu and Z. Sheng, Subspace method based on neural networks for solving the partial differential equation in weak form, arXiv:2405.08513, 2024.

[23] W. F. Mitchell, A collection of 2D elliptic problems for testing adaptive grid refinement algorithms, Applied mathematics and computation, 220 (2013), 350–364.

[24] M. Raissi, P. Perdikaris and G. E. Karniadakis, Physics informed deep learning (part I): Data-driven solutions of nonlinear partial differential equations, arXiv:1711.10561, 2017.

[25] Y. Shang, F. Wang and J. Sun, Randomized neural network with Petrov-Galerkin methods for solving linear and nonlinear partial differential equations, Communications in Nonlinear Science and Numerical Simulation, 127 (2023), 107518.

[26] J. Shen, T. Tang and L. Wang, Spectral methods: algorithms, analysis and applications: volume 41, Springer Science & Business Media, 2011.

[27] J. Sirignano and K. Spiliopoulos, DGM: A deep learning algorithm for solving partial differential equations, J. Comput. Phys., 375 (2018), 1339–1364.

[28] J. Sun, S. Dong and F. Wang, Local randomized neural networks with discontinuous Galerkin methods for partial differential equations, J. Comput. Appl. Math., 445 (2024), 115830.

[29] T. Vejchodskỳ, Complementarity based a posteriori error estimates and their properties, Math. Comput. Simulat., 82(10) (2012), 2033–2046.

[30] T. Wang, Z. Hu, K. Kawaguchi, Z. Zhang and G. E. Karniadakis, Tensor neural networks for high-dimensional Fokker-Planck equations, arXiv:2404.05615v1, 2024.

[31] Y. Wang, P. Jin and H. Xie, Tensor neural network and its numerical integration, arXiv:2207.02754, 2022.

[32] Y. Wang, Y. Liao and H. Xie, Solving Schrödinger equation using tensor neural network, arXiv:2209.12572, 2022.

[33] Y. Wang, Z. Lin, Y. Liao, H. Liu and H. Xie, Solving high-dimensional partial differential equations using tensor neural network and a posteriori error estimators, arXiv:2311.02732, 2023.

[34] Y. Wang and H. Xie, Computing multi-eigenpairs of high-dimensional eigenvalue problems using tensor neural networks, J. Comput. Phys., 506 (2024), 112928 (arXiv:2305.12656, 2023).

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

[36] Y. Xie, H. Chi, Y. Wang and Y. Ma, Physics-specialized neural network with hard constraints for solving multi-material diffusion problems, Computer Methods in Applied Mechanics and Engineering, 430 (2024), 117223.

[37] Z. Xu and Z. Sheng, Subspace method based on neural networks for solving the partial differential equation, arXiv:2404.08223, 2024.

[38] Y. Yao, J. Guo and T. Gu, A deep learning method for multi-material diffusion problems based on physics-informed neural networks, Computer Methods in Applied Mechanics and Engineering, 417 (2023), 116395.

[39] Y. Zang, G. Bao, X. Ye and H. Zhou, Weak adversarial networks for high-dimensional partial differential equations, J. Comput. Phys., 411 (2020), 109409.

[40] H. Zhou and Z. Sheng, Improved randomized neural network methods with boundary processing for solving elliptic equations, arXiv:2407.18457, 2024.