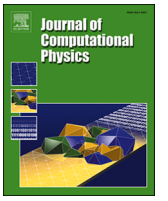
Journal of Computational Physics 564 (2026) 115153

Contents lists available at ScienceDirect

Journal of Computational Physics

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



A categorical embedding physics-informed neural network for anisotropic elliptic interface problems

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

[a] Department of Mathematics, National Central University, Taoyuan, 32001, Taiwan
[b] Department of Applied Mathematics, National Yang Ming Chiao Tung University, Hsinchu, 30010, Taiwan
[c] Department of Applied Mathematics, National University of Kaohsiung, 81148, Kaohsiung, Taiwan

ARTICLE INFO

Keywords:
Anisotropic elliptic interface problems
Physics-informed neural networks
Piecewise continuous functions
Categorical embedding
Heterogeneous materials
Crystal growth

ABSTRACT

Anisotropic elliptic interface problems are crucial for modeling heat and mass transport in devices composed of heterogeneous materials. The solutions to such problems are typically piecewise-smooth functions, making them challenging to solve using traditional numerical methods, especially when many components with distinct material properties are involved. In this paper, we propose a physics-informed neural network with categorical embedding for solving anisotropic elliptic interface problems. The key factors include mapping domain segments to mutually disconnected sets in a high-dimensional representation space, and subsequently extracting salient low-dimensional features from this representation. The former mitigates the importance of ordinal or nominal labels for each subdomain, while the latter significantly reduces the number of trainable parameters introduced by the former. By automatically learning low-dimensional features, the proposed categorical embedding technique avoids the need for explicit domain labeling, allowing a single neural network to accurately approximate piecewise-smooth functions, even when the number of discontinuous pieces ranges from tens to hundreds. An anisotropic elliptic interface problem is then solved by training the network within a physics-informed framework, in which the mean squared loss function comprises the residual of the governing equations. Numerical experiments demonstrate that the proposed network-based meshless method achieves accuracy and efficiency comparable to traditional grid-based numerical methods.

1. Introduction

Anisotropic elliptic interface problems are crucial to real-world applications, enabling accurate modeling and prediction in multiphase fluid flows, composite materials, biological systems, and geological processes. For example, the growth of liquid crystals involves interfaces between liquid and solid phases [1,2], so that the heat and mass transports during this process are anisotropic due to the underlying crystal structure. When modeling semiconductor heterostructures such as quantum wells, where abrupt changes in composite materials create interfaces, anisotropic elliptic partial differential equations can be used to simulate spin transport and diffusion [3] within these structures. Furthermore, placing anisotropic fluids such as nematic liquid crystals inside a Hele-Shaw cell [4,5] enables the study of the interplay between the fluid-inherent anisotropy and the cell-confined geometry.

* Corresponding author.
E-mail addresses: wfhu@math.ncu.edu.tw (W.-F. Hu), teshenglin@nycu.edu.tw (T.-S. Lin), yhtseng@go.nuk.edu.tw (Y.-H. Tseng), mclai@nycu.edu.tw (M.-C. Lai).

https://doi.org/10.1016/j.jcp.2026.115153
Received 5 October 2025; Received in revised form 17 May 2026; Accepted 19 June 2026
Available online 22 June 2026

Owing to the essential role in many natural phenomena and industrial applications, finding effective numerical solutions to anisotropic elliptic interface problems is a significant task in the scientific computing community. High-order numerical methods have been developed for elliptic interface problems [6,7]; however, the case involving anisotropic diffusion tensors presents significantly greater challenges. The primary difficulty arises because the diffusion tensor is typically piecewise continuous, while the solution itself exhibits discontinuities across interfaces. To address this issue, nonlinear finite volume schemes have been proposed to preserve solution positivity [8,9]. Immersed finite element methods for three-dimensional anisotropic magnetostatic and electrostatic interface problems were developed in [10]; see also related approaches in [11–14]. More recently, Li et al. [15] employed deep neural networks with a first-order formulation to approximate such problems. Their results only demonstrate smooth solutions; interface problems with piecewise continuous solutions remain considerably difficult. The low global regularity of these solutions makes the design of efficient and accurate numerical methods both essential and highly nontrivial.

Recent advances have demonstrated the effectiveness of neural network-based methods in tackling problems that remain challenging for classical numerical approaches, including inverse problems [16], triangulated mesh prediction [17], and solutions with corner singularities [18]. From a theoretical perspective, the expressive power of neural networks has been rigorously established [19–22], showing that continuous functions can be approximated to arbitrary precision. This provides a versatile function representation that can be leveraged to tackle a wide range of scientific and engineering problems. Notable examples include Physics-Informed Neural Networks (PINNs) [23,24] and the deep Ritz method [25], both of which have inspired extensive research on solving partial differential equations (PDEs) in complex geometries and high-dimensional settings. More recently, operator-learning frameworks such as Deep Operator Networks [26] and Fourier Neural Operator [27] have emerged, which directly approximate nonlinear operators mapping between function spaces, thereby expanding the scope and applicability of neural network approaches in scientific computing. On the other hand, for discontinuous functions, Llanas et al. [28] proved that neural networks can approximate piecewise continuous functions almost uniformly. Their approach, however, relied on continuous activation functions to approximate step functions, which inevitably introduces an overall smoothness to the network representation. An alternative is to employ discontinuous activation functions [29–31], commonly encountered in electronic circuits with switches or dry friction, which have shown effectiveness in optimization tasks such as linear and quadratic programming [32]. Nevertheless, how to systematically train such neural networks remains an open research question.

In this work, we propose to solve anisotropic elliptic interface problems within the PINN framework by introducing a methodology for accurately representing discontinuous solutions composed of multiple smooth pieces. He et al. [33] partitioned the input space into subdomains separated by discontinuities, assigning an individual neural network to each partition. While intuitive, this strategy suffers from scalability issues, as the number of required networks scales with the number of partitions. Alternative formulations based on weak-form losses [34] alleviate some of these difficulties but are hindered by the limited accuracy of Monte Carlo integration. Recent developments also include hybrid neural-discretization frameworks such as JAX-DIPS [35]. To overcome these challenges, Hu et al. introduced the Discontinuity-Capturing Shallow Neural Network (DCSNN) [36], a single network architecture that accurately represents piecewise-continuous functions. However, when a piecewise-smooth function has a large number of pieces, DCSNN encounters training efficiency issues mainly due to the choice of domain labeling. In this paper, we propose a categorical embedding neural network for solving anisotropic elliptic interface problems, where the solutions are typically piecewise-smooth. The network architecture comprises three hidden layers: a discontinuity-capturing (DC) layer that utilizes one-hot encoding maps domain segments into disconnected sets in a higher-dimensional space; a categorical embedding (CE) layer that extracts the low-dimensional features from the high-dimensional DC layer; and a fully connected layer, which models the continuous mapping from the CE layer to the output. By automatically learning discontinuity embeddings, the proposed categorical embedding technique avoids the need for explicit domain labeling and allows a single neural network to accurately approximate piecewise-smooth functions, even when the number of discontinuous pieces ranges from tens to hundreds.

The rest of this paper is organized as follows. In Section 2, we introduce the mathematical model of anisotropic elliptic interface problems. In Section 3, we illustrate the network structures of DCSNN alongside the category-embedding structure for piecewise-smooth functions, and apply the proposed neural network to anisotropic elliptic interface problems in Section 4. Section 5 provides a series of numerical experiments to demonstrate the capability and accuracy, and the concluding remarks are shown in Section 6.

2. Anisotropic elliptic interface problems

Let $\Omega \subseteq \mathbb{R}^d$ be a bounded, simply connected domain, together with L disjoint subdomains, $\Omega_1, \Omega_2, \cdots, \Omega_L \subset \Omega$, and $\Omega_i \cap \Omega_j = \emptyset$ for $i \neq j$. Assume that the boundary of each subdomain, Ω_ℓ, denoted by Γ_ℓ, is a $(d-1)$-dimensional closed C^1-hypersurface, and the complementary part of the union of all subdomains is represented by Ω_0 so that $\Omega = \bigcup_{\ell=0}^{L} \Omega_\ell$. An illustrative two-dimensional example with $L = 4$ is shown in Fig. 1. In addition, in the rest of this paper, we assume that u is a d-dimensional piecewise function $u : \Omega \to \mathbb{R}$ which is smooth within each subdomain Ω_ℓ and exhibits jump discontinuities across each interface Γ_ℓ.

The d-dimensional anisotropic elliptic interface problem with nonhomogeneous jump conditions is stated as follows:

$$\begin{aligned} &\nabla \cdot (A(\mathbf{x})\nabla u(\mathbf{x})) - \lambda(\mathbf{x})u(\mathbf{x}) = f(\mathbf{x}) \quad \text{in } \Omega \setminus \bigcup_{\ell=1}^{L} \Gamma_\ell, \\ &[u] = v_\ell(\mathbf{x}), \quad [A\nabla u \cdot \mathbf{n}] = w_\ell(\mathbf{x}) \quad \text{on } \Gamma_\ell, \text{ for } \ell = 1, 2, \cdots, L, \end{aligned} \tag{1}$$

where ∇ and $\nabla\cdot$ are respectively the gradient and divergence operators with respect to the spatial variable $\mathbf{x} = (x_1, x_2, \cdots, x_d)$, $A(\mathbf{x}) \in \mathbb{R}^{d\times d}$ is a positive-definite matrix, and $\lambda(\mathbf{x})$ is a nonnegative scalar function. The objective solution u is typically piecewise-smooth when

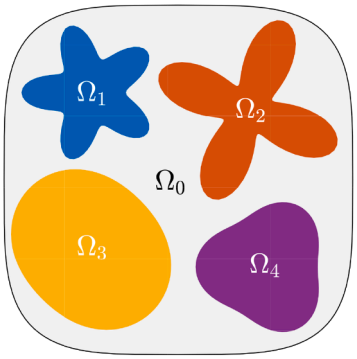


Fig. 1. A two-dimensional domain, $\Omega = \cup_{\ell=0}^{4}\Omega_\ell$, contains four disjoint subdomains.

the source term f, the matrix $A(\mathbf{x})$, and $\lambda(\mathbf{x})$ are piecewise-defined functions which are smooth within each subdomain Ω_ℓ, representing direction-dependent or piecewise properties due to the heterogeneous materials. The bracket $[\cdot]$ denotes the jump quantity for function values approaching from Ω_0 minus the one from the subdomain Ω_ℓ; $\mathbf{n}$ is the unit outward normal vector (pointing from Ω_ℓ toward Ω_0) defined on the interface Γ_ℓ.

To close the system, a boundary condition must be specified along $\partial\Omega$. Throughout this paper, we assume that the solution is imposed by the Dirichlet boundary condition $u(\mathbf{x})|_{\partial\Omega} = g(\mathbf{x})$, while other types of boundary conditions (Neumann or Robin type) can be implemented easily without changing the main ingredient of the proposed method (see the implementation in the following).

3. Neural network architecture for piecewise-smooth functions

As discussed earlier, a standard neural network utilizing smooth activation functions yields inherently smooth representations, making it unable to capture jump discontinuities. To approximate piecewise-smooth functions, we revisit our previous work on DCSNN [36], which demonstrates the ability to model the discontinuous behavior using a single shallow neural network architecture. Building on this foundation, we introduce several classification models, particularly the categorical embedding technique, which automatically learns the embedding discontinuity information of piecewise-smooth functions with a large number of pieces.

3.1. Scalar encoding neural networks

The core idea is to transform a d-dimensional piecewise-continuous function into a continuous function defined in a $(d+1)$-dimensional space. This is done by introducing a mapping that performs classification, assigning each subdomain to a point in a mutually distinct point set. For this purpose, we define a scalar categorical function $z : \mathbb{R}^d \to \mathbb{R}$ as follows:

$$z(\mathbf{x}) = \gamma_\ell \quad \text{if } \mathbf{x} \in \Omega_\ell, \tag{2}$$

where γ_ℓ for $\ell = 0, 1, \cdots, L$ are "predefined" constants. A natural choice is to set $\gamma_\ell = \ell$, whereby γ_ℓ serves as the region index and the label identifies the subdomain to which $\mathbf{x}$ belongs. Alternatively, similar to target embedding categorization [37], one can incorporate input data information into the label, and define γ_ℓ based on function averages over each region: $\gamma_\ell = \bar{\gamma}_\ell = \int_{\Omega_\ell} u(\mathbf{x})\mathrm{d}\mathbf{x}/|\Omega_\ell|$. However, these values may be unavailable if u itself is a function to be found (for instance, when u is the solution of a PDE).

We should point out that the selection of scalars, $\gamma_0, \gamma_1, \cdots, \gamma_L$, implicitly introduces an ordering for the subdomains, which significantly influences the training of subsequent neural networks, especially when dealing with a large number of disjoint subdomains. In addition, seeking the optimal constants a priori is generally impractical. Nevertheless, as long as those label values are distinct, the function $\mathbf{x} \to z(\mathbf{x})$ maps disjoint subdomains to a disconnected point set.

Using the above scalar categorical encoding function $z(\mathbf{x}) \in \mathbb{R}$, we define an augmented function $U : \mathbb{R}^{d+1} \to \mathbb{R}$ that satisfies

$$U(\mathbf{x}, z(\mathbf{x})) = u(\mathbf{x}), \quad \text{if } \mathbf{x} \in \Omega. \tag{3}$$

Notice that U is well defined and continuous on a disconnected subset of $\mathbb{R}^{d+1}$, namely $\mathcal{M} = \{(\mathbf{x}, z(\mathbf{x})) \in \mathbb{R}^{d+1} \mid \mathbf{x} \in \Omega\}$. Since each subdomain is bounded with a smooth boundary, the restriction of U to each connected component of $\mathcal{M}$ is smooth. Classical smooth extension results, such as the Whitney and Seeley extension theorems [38,39], imply that U admits a smooth extension to all of $\mathbb{R}^{d+1}$. We emphasize that our framework does not explicitly construct such extensions; rather, neural networks with smooth activation functions are used as smooth approximators in the augmented space. The remaining task is to construct a neural network to approximate the augmented continuous function U.

Leveraging the rich expressiveness of neural networks, the extension function U can be approximated by a fully-connected one-hidden-layer neural network, $U_{\mathcal{N}}$, which takes the form

$$U_{\mathcal{N}}(\mathbf{x}, z(\mathbf{x}); \Theta) = \sum_{j=1}^{N} c_j \sigma(W_j[\mathbf{x}, z]^\top + b_j), \tag{4}$$

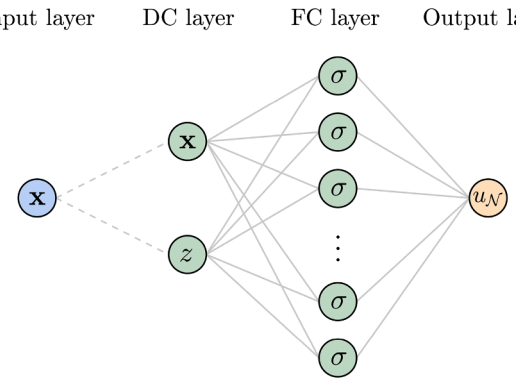


Fig. 2. Schematic diagram of the scalar encoding neural network. The dashed/solid lines denote non-trainable/trainable parameters.

where N is the number of neurons in the hidden layer and σ is the activation function. The notation Θ denotes a vector collecting all the trainable parameters, including the weights, $c_j \in \mathbb{R}$ and $W_j \in \mathbb{R}^{1\times(d+1)}$, and the biases, $b_j \in \mathbb{R}$. In addition, we define a neural function $u_{\mathcal{N}}(\mathbf{x})$ through $U_{\mathcal{N}}$, representing inherently piecewise continuous functions as

$$u_{\mathcal{N}}(\mathbf{x}) = U_{\mathcal{N}}(\mathbf{x}, z(\mathbf{x}); \Theta). \tag{5}$$

With the same input, this succinct notation makes it easy to estimate the error between the target function $u(\mathbf{x})$ and the neural network function $u_{\mathcal{N}}(\mathbf{x})$.

In summary, the scalar encoding neural network $u_{\mathcal{N}}$ consists of two layers: a discontinuity-capturing (DC) layer and a fully connected (FC) layer. The DC layer maps the input variable $\mathbf{x}$ to $(\mathbf{x}, z(\mathbf{x}))$, which is pre-defined without training. On the contrary, the FC layer consists of N neurons with trainable weights and biases. A schematic illustration of the network structure is shown in Fig. 2. As a result, the number of overall trainable parameters is $N_p = (d+3)N$.

3.2. One-hot encoding neural networks

Inspired by one-hot encoding in classification tasks, we propose an alternative representation of the categorical function. Specifically, we define a vector-valued function $\mathbf{z} : \mathbb{R}^d \to \mathbb{R}^{L+1}$ to represent categorical variables in a binary format:

$$\mathbf{z}(\mathbf{x}) = \boldsymbol{\delta}_{\ell+1}, \quad \text{if } \mathbf{x} \in \Omega_\ell, \tag{6}$$

for $\ell = 0, 1, \cdots, L$, where $\boldsymbol{\delta}_{\ell+1}$ is the $(l+1)$-th vector of the canonical basis for $\mathbb{R}^{L+1}$. This mapping offers the advantage that the distance between the augmented features $\mathbf{z}$ of any two subdomains remains constant, avoiding any implicit ordering in the augmentation coordinates. We then construct a $(d+L+1)$-dimensional continuous extension function U that incorporates the vector-valued categorical function $\mathbf{z}$ as follows:

$$U(\mathbf{x}, \mathbf{z}(\mathbf{x})) = u(\mathbf{x}), \quad \text{if } \mathbf{x} \in \Omega. \tag{7}$$

This intermediate map U can again be simply approximated by a shallow network $u_{\mathcal{N}}$ of the form

$$u_{\mathcal{N}}(\mathbf{x}) = U_{\mathcal{N}}(\mathbf{x}, \mathbf{z}(\mathbf{x}); \Theta) = \sum_{j=1}^{N} c_j \sigma(W_j [\mathbf{x}, \mathbf{z}]^\top + b_j), \tag{8}$$

where Θ represents a vector collecting all the trainable parameters. Notably, the usage of the vector-valued functions as augmented inputs has also been explored in [40], where it was employed to classify two subdomains within their network architecture.

Compared to the scalar encoding model (2), adopting the one-hot encoding model seems ideal in the absence of ordinal or nominal labels for each subdomain. However, approximating this continuous function U using a neural network expression with the augmented input $(\mathbf{x}, \mathbf{z}) \in \mathbb{R}^{d+L+1}$ poses computational challenges as the number of subdomains increases (i.e., larger L), leading to a significant increase in the number of trainable parameters. Consequently, the number of unknown parameters of the network expression (8) now becomes $N_p = (d+L+3)N$, making the training process computationally intensive.

3.3. Categorical embedding neural networks

From the two models discussed above, it is evident that multiple options exist for defining the categorical function. The role of this function is to construct a mapping that both lifts each smooth component of the piecewise function into a higher-dimensional space and ensures their separation in that space. Consequently, there are infinitely many possible choices for such a mapping.

In this context, we propose a categorical embedding neural network, in which the objective is to *learn* an optimal categorical function. This function aims to effectively capture the intrinsic properties (or features) within the function profiles of each subdomain (or

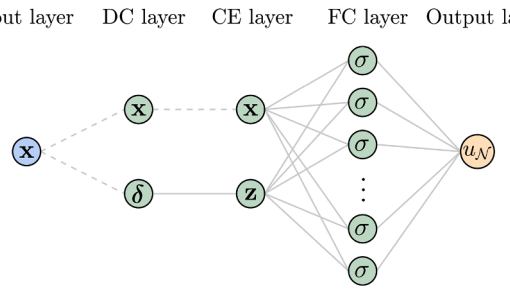


Fig. 3. Schematic diagram of the categorical embedding neural network. The dashed/solid lines denote non-trainable/trainable parameters.

category), and thus map similar categories closer together in a specific low-dimensional space. To this end, we define the categorical function $\mathbf{z} : \mathbb{R}^d \to \mathbb{R}^D$ via a linear map:

$$\mathbf{z}(\mathbf{x}) = E\boldsymbol{\delta}(\mathbf{x}), \tag{9}$$

where $\boldsymbol{\delta}(\mathbf{x}) = \boldsymbol{\delta}_{\ell+1} \in \mathbb{R}^{L+1}$ for $\mathbf{x} \in \Omega_\ell$, and $E \in \mathbb{R}^{D\times(L+1)}$ is the embedding matrix that classifies the $(L+1)$ subdomains into a low-dimensional embedded space of the selected dimension D (with $D \le L+1$). We remark that the categorical embedding function (9) used here is inspired by the same underlying idea as the entity embedding model [41], where high-cardinality categorical variables are embedded into low-dimensional Euclidean spaces.

Again, we look for an extension function $U : \mathbb{R}^{d+D} \to \mathbb{R}$ with the proposed categorical embedding map (9) that satisfies

$$U(\mathbf{x}, \mathbf{z}(\mathbf{x})) = U(\mathbf{x}, E\boldsymbol{\delta}(\mathbf{x})) = u(\mathbf{x}), \quad \text{if } \mathbf{x} \in \Omega. \tag{10}$$

At this stage, the embedding matrix E is treated as an unknown weight matrix and can be learned through standard machine learning optimization techniques. Additionally, we emphasize that the importance of this categorical step lies in mapping low-dimensional, discontinuous functions to high-dimensional, smooth functions.

As before, it involves approximating the extension function U by constructing a categorical embedding neural network $u_{\mathcal{N}}$ through the intermediate map $U_{\mathcal{N}}$ that represents inherently piecewise functions as follows:

$$u_{\mathcal{N}}(\mathbf{x}) = U_{\mathcal{N}}(\mathbf{x}, E\boldsymbol{\delta}(\mathbf{x}); \Theta) = \sum_{j=1}^{N} c_j \sigma(W_j[\mathbf{x}, E\boldsymbol{\delta}(\mathbf{x})]^\top + b_j), \tag{11}$$

where Θ represents a vector that collects all the trainable parameters, including those in the embedding feature matrix E. The number of total trainable parameters, including weights, biases, and embedding matrix E, is $N_p = (d + D + 2)N + (L+1)D$.

The categorical embedding neural network consists of three layers: a DC layer, a categorical embedding (CE) layer, and an FC layer. The DC layer maps the input variable $\mathbf{x}$ to $(\mathbf{x}, \boldsymbol{\delta}(\mathbf{x}))$, which is predefined and does not require training. In contrast, the CE layer learns the embedding (or weight) matrix E via the mapping $\mathbf{z}(\mathbf{x}) = E\boldsymbol{\delta}(\mathbf{x})$. The FC layer consists of N neurons with trainable weights and biases. A schematic representation of the network structure is shown in Fig. 3. In this paper, we focus on the simple structure defined in Eq. (11). While replacing the FC layer with a deep neural network or adding residual connections to the proposed model is straightforward.

We provide several remarks on the proposed model in the following. Firstly, the categorical embedding technique generalizes both scalar encoding and one-hot encoding models depending on the choice of the embedding matrix E. Particularly, setting $E = [\gamma_0, \gamma_1, \cdots, \gamma_L]$ recovers the scalar encoding model (see Eq. (2)), while choosing the identity matrix $E = I_{L+1}$ reverts to the one-hot encoding approach (see Eq. (6)). Secondly, one can also generalize the proposed linear embedding by including nonlinearity. For instance, applying an activation function to the categorical embedding, i.e., setting $\mathbf{z}(\mathbf{x}) = \sigma(E\boldsymbol{\delta}(\mathbf{x}))$, may enhance the classification of the encoding labels in the embedded space. Thirdly, selecting the best low-dimensional embedding, or the optimal reduced dimension D, of the neural network function $u_{\mathcal{N}}$ in Eq. (11) remains a subject of ongoing research. To provide insight into the choice of D, we will present a systematic study of function approximation experiments in Section 5.1. Lastly, the existence of the smooth extension function U on the entire domain is guaranteed by classical smooth extension results such as the Whitney and Seeley extension theorems [38,39], ensuring that such a function can be approximated using neural networks for the above three models.

3.4. Training method

In the training procedure of the network function $u_{\mathcal{N}}(\mathbf{x})$ in Eq. (11), we utilize the commonly used mean-squared error loss function. As in supervised learning tasks, we approximate a piecewise-defined function $u(\mathbf{x})$ using the neural network $u_{\mathcal{N}}(\mathbf{x})$. We choose a set of training data, $\{(\mathbf{x}^i, u^i)\}_{i=1}^M$, where $\mathbf{x}^i \in \Omega$ and $u^i = u(\mathbf{x}^i)$, and specify the embedding function $\mathbf{z}(\mathbf{x})$. The loss function under the supervised learning framework is then defined as

$$\text{Loss}(\Theta) = \frac{1}{M}\sum_{i=1}^{M}\left(u^i - u_{\mathcal{N}}(\mathbf{x}^i)\right)^2. \tag{12}$$

Recall that $u_{\mathcal{N}}(\mathbf{x}^i) = U_{\mathcal{N}}(\mathbf{x}^i, \mathbf{z}(\mathbf{x}^i); \Theta)$, and Θ is a vector collecting all the trainable parameters.

Throughout the numerical experiments in this paper, we employ the Levenberg–Marquardt (LM) method [42], which is particularly effective due to the adopted least-squares loss formulation. The update at step $(k+1)$ in the LM method is expressed as

$$\Theta^{(k+1)} = \Theta^{(k)} + (J^\top J + \mu I)^{-1}\left[J^\top\left(\mathbf{u} - \mathbf{u}_{\mathcal{N}}(\Theta^{(k)})\right)\right], \tag{13}$$

where $\mu > 0$ is a tunable damping parameter; $\mathbf{u}$ and $\mathbf{u}_{\mathcal{N}}(\Theta^{(k)})$ denote the vectors collecting the data u^i and $U_{\mathcal{N}}(\mathbf{x}^i, \mathbf{z}(\mathbf{x}^i); \Theta^{(k)})$ in the loss function (12), respectively. Here, J is the Jacobian matrix defined as $\partial \mathbf{u}_{\mathcal{N}}(\Theta^{(k)})/\partial\Theta$. Obviously, the primary computational expense in the LM update step $\Delta\Theta$ (i.e., the matrix-vector multiplication in the right-hand side of Eq. (13)) arises from solving the regularized least-squares problem with the damping parameter μ. This step is typically performed using Cholesky factorization. However, when $\mu \ll 1$, the condition number of $J^\top J + \mu I$ can become extremely large, leading to instability and poor approximation of the update step. To mitigate this issue, we can compute the parameter update $\Delta\Theta$ using QR factorization by solving the following linear system:

$$\begin{bmatrix} J \\ \sqrt{\mu} I \end{bmatrix} \Delta\Theta = \begin{bmatrix} \mathbf{u} - \mathbf{u}_{\mathcal{N}}(\Theta^{(k)}) \\ \mathbf{0} \end{bmatrix}.$$

Note that the network model with the loss function (12) is conventionally trained using optimizers such as Adam [43] or L-BFGS [44], which are widely adopted in the literature. In subsequent experiments, we will compare the efficiency of various approaches using these optimizers during training.

As discussed above, the optimization in this work is primarily performed using the LM method, which is well suited for nonlinear least-squares problems and often exhibits fast and stable convergence for moderately sized networks [45]. However, LM is known to suffer from scalability limitations due to the need to compute and store the Jacobian matrix and to solve a linear system at each iteration. In particular, for M residuals and N_p trainable parameters, the per-iteration computational cost is typically of order $O(M N_p^2 + N_p^3)$ when using Cholesky- or QR-based solvers. Consequently, both the computational and memory costs increase rapidly as the network size grows. Therefore, while LM is effective for the relatively compact network architectures considered in this work, its applicability to large-scale neural networks remains limited. For larger-scale problems, a hybrid training strategy may be adopted, in which first-order optimizers are first used to obtain a good initialization, followed by LM to accelerate convergence during the later stages of training. We also note that several modified second-order approaches, such as the geodesic-acceleration variant of the LM method [46] used in DR-PINNs [47], have been proposed to improve convergence behavior and scalability for larger neural network models.

4. Categorical embedding neural network for anisotropic elliptic interface problems

In this section, we apply the proposed categorical embedding model (11) as a solution representation for the anisotropic elliptic interface problem (1). The derivation of the jump discontinuities and the derivatives of the categorical embedding function $u_{\mathcal{N}}(\mathbf{x})$ in Eq. (11) is presented in Section 4.1, while Section 4.2 introduces a PINN framework for solving such a problem.

4.1. Discontinuities and derivatives

Values at jump discontinuities. Generally, the function values of a discontinuous function at a breakpoint hold less significance. Of greater importance is the jump value at the breakpoint, which represents the difference between the limiting values approaching from two opposite directions. For instance, denoting the boundary of subdomain Ω_1 as Γ_1, the jump quantity for the function value at $\mathbf{x} \in \Gamma_1$ is given by

$$[u]_{\mathbf{x}} = \lim_{\mathbf{x}^+ \to \mathbf{x}} u(\mathbf{x}^+) - \lim_{\mathbf{x}^- \to \mathbf{x}} u(\mathbf{x}^-), \tag{14}$$

where $\mathbf{x}^+ \in \Omega_0$ and $\mathbf{x}^- \in \Omega_1$ (see Fig. 1 for example). Surprisingly, this information naturally emerges within the intermediate map U from Eq. (10). That is, taking the categorical function $\mathbf{z}$ in Eq. (9), the limiting value approaching from Ω_0 follows

$$\lim_{\mathbf{x}^+ \to \mathbf{x}} u(\mathbf{x}^+) = \lim_{\mathbf{x}^+ \to \mathbf{x}} U(\mathbf{x}^+, \mathbf{z}(\mathbf{x}^+)) = U(\mathbf{x}, E\boldsymbol{\delta}_0), \tag{15}$$

while the other limit is $\lim_{\mathbf{x}^- \to \mathbf{x}} u(\mathbf{x}^-) = U(\mathbf{x}, E\boldsymbol{\delta}_1)$. Thus the jump value at $\mathbf{x} \in \Gamma_1$, which typically requires taking one-sided limits of the function u, can be calculated directly by evaluating $[u]_{\mathbf{x}} = U(\mathbf{x}, E\boldsymbol{\delta}_0) - U(\mathbf{x}, E\boldsymbol{\delta}_1)$. The same manner applies to the jump of the network function $u_{\mathcal{N}}$ as

$$[u_{\mathcal{N}}]_{\mathbf{x}} = U_{\mathcal{N}}(\mathbf{x}, E\boldsymbol{\delta}_0; \Theta) - U_{\mathcal{N}}(\mathbf{x}, E\boldsymbol{\delta}_1; \Theta). \tag{16}$$

The rationale for this outcome is the smoothness of the intermediate map U. Consequently, determining the jump quantity can be done easily by evaluating the intermediate map, eliminating the need to take limits of the function.

Derivative evaluation. Except at breakpoints, the categorical function $\mathbf{z}(\mathbf{x})$ remains constant everywhere; hence, its derivative is zero, implying that the Jacobian matrix $\nabla\mathbf{z} = \mathbf{0}$. To express the gradient of network function $u_{\mathcal{N}}$ in terms of the intermediate map $U_{\mathcal{N}}$, the chain rule gives

$$\nabla u_{\mathcal{N}} = \nabla_{\mathbf{x}} U_{\mathcal{N}} + \nabla\mathbf{z}\nabla_{\mathbf{z}} U_{\mathcal{N}} = \nabla_{\mathbf{x}} U_{\mathcal{N}}, \tag{17}$$

where ∇ is the conventional gradient operator; $\nabla_{\mathbf{x}}$ and $\nabla_{\mathbf{z}}$ denote differentiating only with respect to $\mathbf{x}$ and $\mathbf{z}$, respectively. As a result, the partial derivatives of $u_{\mathcal{N}}$ can be calculated equivalently by taking the partial derivatives of $U_{\mathcal{N}}$ with respect to the original variables $\mathbf{x}$. Higher-order derivatives of $u_{\mathcal{N}}$ are calculated in the same manner.

4.2. PINN framework

We now apply the methodology of physics-informed learning machinery [24] for solving Eq. (1) as follows. To find the network parameters in the expression (11), we convert Eq. (1) to an optimization problem via a loss function. Namely, we first choose the sets of training points in the domain, on the domain boundary, and along all the interfaces, as

$$\{\mathbf{x}^i\}_{i=1}^{M} \subseteq \Omega, \qquad \left\{\mathbf{x}_{\partial\Omega}^j\right\}_{j=1}^{M_b} \subseteq \partial\Omega, \qquad \left\{\mathbf{x}_{\Gamma_\ell}^k\right\}_{k=1}^{M_{\Gamma_\ell}} \subseteq \Gamma_\ell,$$

respectively. The loss function is then defined as

$$\begin{aligned}\text{Loss}(\Theta) = &\frac{1}{M}\sum_{i=1}^{M}\left(\nabla\cdot(A(\mathbf{x}^i)\nabla u_{\mathcal{N}}(\mathbf{x}^i)) - \lambda(\mathbf{x}^i)u_{\mathcal{N}}(\mathbf{x}^i) - f(\mathbf{x}^i)\right)^2 \\ &+\frac{\lambda_b}{M_b}\sum_{j=1}^{M_b}\left(u_{\mathcal{N}}(\mathbf{x}_{\partial\Omega}^j) - g(\mathbf{x}_{\partial\Omega}^j)\right)^2 \\ &+\sum_{\ell=1}^{L}\frac{\lambda_\ell}{M_{\Gamma_\ell}}\left(\sum_{k=1}^{M_{\Gamma_\ell}}\left([u_{\mathcal{N}}] - v_\ell(\mathbf{x}_{\Gamma_\ell}^k)\right)^2 + \left([A\nabla u_{\mathcal{N}}\cdot\mathbf{n}] - w_\ell(\mathbf{x}_{\Gamma_\ell}^k)\right)^2\right).\end{aligned} \tag{18}$$

which consists of the mean squared residual for each equation in the original problem (1), following the same principle as the PINN-type loss [23,24]. We also recall that Θ represents the set of trainable parameters, and the objective is to find Θ that minimizes the loss function. As each term in the loss again takes the form of least-squares errors, we can train the model efficiently using the LM algorithm. In PINNs, balancing multiple loss components is known to affect training performance. Various adaptive weighting strategies have been proposed, including gradient-based weighting [48], neural tangent kernel-based approaches [49], and pointwise residual weighting methods [50]. In the present work, we adopt an unweighted loss formulation $\lambda_b = \lambda_\ell = 1$ for simplicity. Our numerical results indicate that the unweighted formulation already yields stable training and satisfactory accuracy for the problems considered. Since the primary focus of the CE-PINN framework is the embedding architecture for piecewise discontinuous solutions, a systematic investigation of adaptive loss balancing remains an important direction for future work.

As mentioned in Section 4.1, we recall that the categorical function $\mathbf{z}(\mathbf{x})$ is a piecewise constant vector function that has zero derivative over the interior of each subdomain, so that computing derivatives of $u_{\mathcal{N}}$, such as gradient or divergence, can be straightforwardly done using the chain rule without any difficulty. For example, recall the relation (17), we have

$$\nabla u_{\mathcal{N}}(\mathbf{x}) = \nabla_{\mathbf{x}} U_{\mathcal{N}}(\mathbf{x}, \mathbf{z}; \Theta), \quad \mathbf{x} \in \Omega \setminus \bigcup_{\ell=1}^{L}\Gamma_\ell, \tag{19}$$

where $U_{\mathcal{N}}$ is the intermediate map and $\nabla_{\mathbf{x}}$ is the gradient operator with respect to the $\mathbf{x}$ variable only. Therefore, all the derivatives in Eq. (18) are well-defined. As regards the jump conditions in Eq. (18), they can be computed easily through function evaluations of $U_{\mathcal{N}}$ (see Section 4.1). Similarly, the computation for the flux jump condition $[A\nabla u_{\mathcal{N}}\cdot\mathbf{n}]$ can be evaluated in the same manner as $[u_{\mathcal{N}}]$.

Moreover, the derivative terms involved in the loss are commonly computed using automatic differentiation. However, in fact, thanks to the design of the shallow network structure (11), one can easily derive the explicit form of the derivatives, which is much more efficient in practice. For example, the partial derivative with respect to the k-th spatial component is obtained by

$$\frac{\partial u_{\mathcal{N}}}{\partial x_k}(\mathbf{x}) = \frac{\partial U_{\mathcal{N}}}{\partial x_k}(\mathbf{x}, \mathbf{z}; \Theta) = \sum_{j=1}^{N} c_j W_{jk}\sigma'(W_j[\mathbf{x}, \mathbf{z}]^\top + b_j), \tag{20}$$

where W_{jk} is the k-th component of the vector W_j; the prime notation of σ means the derivative of the activation function. We mention that both the higher-order or mixed partial derivatives of the target function and the Jacobian matrix (collecting all partial derivatives with respect to the trainable parameters Θ for each residual loss in Eq. (18)) involved in the LM training iteration can be implemented straightforwardly using the simple formulation (11). We also point out that since a simple structure with a moderate number of neurons is employed in the present network, the computational complexity and learning workload can be significantly reduced.

5. Numerical results

This section presents a series of numerical experiments to demonstrate the effectiveness of the proposed categorical embedding techniques for representing piecewise-smooth functions and for solving anisotropic elliptic interface problems. Benchmark comparisons with several existing PINN-based interface methods are also included. Throughout all experiments, the sigmoid activation function is employed. Since the numerical results may vary slightly due to the random sampling of training points and the initialization of trainable parameters, we report the average L^2 and L^∞ errors over several independent runs. Unless otherwise specified, the LM optimizer is terminated either after 1000 training steps or when the tolerance $\varepsilon = 10^{-15}$ is reached. The damping parameter is initially set to $\mu = 1$, with increment and reduction factors of 2 and 1.2, respectively. All experiments were performed on a desktop equipped with an NVIDIA GeForce RTX 4090 GPU.

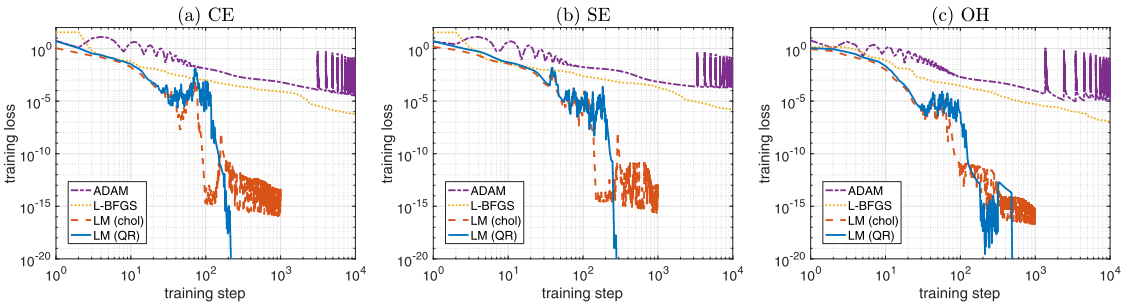


Fig. 4. Training history for (a) categorical embedding, (b) scalar encoding, and (c) one-hot encoding model with different optimizers.

Table 1
Numerical results for approximating multi-piece function (21). CE: categorical embedding model; SE: scalar encoding model; OH: one-hot encoding model.

Method	N_p	L^2 error	L^∞ error
CE ($D = 1$)	255	6.47E−08	2.09E−06
SE ($\gamma_\ell = \ell$)	250	1.10E−07	4.07E−06
OH	450	4.22E−08	1.06E−06

5.1. Examples for the encoding methods

We present two examples to illustrate the capability of the proposed CE model, originally introduced as the categorical embedding technique, in approximating piecewise-smooth functions. Additionally, we compare its performance with scalar encoding (SE) and one-hot encoding (OH) models. In all network models, we use $N = 50$ neurons in the FC layer, maintaining the same number of basis network functions across different approaches.

Example 1. We consider a domain $\Omega \subset \mathbb{R}^2$ that is enclosed by the superellipse, $x_1^4 + x_2^4 = 1$, which encapsulates four subdomains whose boundaries (or interfaces) are described by the polar curves as $r_1(\theta) = 0.3 - 0.1\cos(5\theta), r_2(\theta) = 0.35 - 0.2\sin(4\theta), r_3(\theta) = 0.45 - 0.05\sin(2\theta), r_4(\theta) = 0.35 - 0.05\cos(3\theta)$ with the center located at $(-0.5, 0.5)$, $(0.4, 0.4)$, $(-0.5, -0.4)$, $(0.5, -0.5)$, respectively. The domain is depicted in Fig. 1. The target function u is chosen as

$$u(x_1, x_2) = \begin{cases} \sin(x_1)\sin(x_2) & \text{if } (x_1, x_2) \in \Omega_0, \\ \exp(x_1 - x_2) & \text{if } (x_1, x_2) \in \Omega_1, \\ \cos(x_1 + x_2) & \text{if } (x_1, x_2) \in \Omega_2, \\ 0.5\cosh(x_1 + x_2) & \text{if } (x_1, x_2) \in \Omega_3, \\ \ln(x_1 + x_2 + 3) & \text{if } (x_1, x_2) \in \Omega_4. \end{cases} \tag{21}$$

We minimize the loss function (12) using $M = 1000$ randomly sampled training points, consisting of 880 points inside the domain Ω and 120 points along the domain boundary $\partial\Omega$.

To demonstrate the training efficiency, we compare the performance of various optimizers discussed previously with the three categorical network models. For the scalar encoding model, we particularly set the nominal label values as $\gamma_\ell = \ell$ for $\ell = 0, 1, \cdots, 4$, while for the CE model, we set $D = 1$.

The results are reported in Fig. 4, where all models exhibit a similar trend. Specifically, the gradient-based Adam optimizer (purple dash-dotted line) achieves a loss magnitude of approximately 10^{-5} but gets stuck in a local minimum, even after 10000 training steps. Meanwhile, the quasi-Newton L-BFGS algorithm (yellow dotted line) performs slightly better than Adam but also nearly stagnates, resulting in a slight decrease in the loss value in subsequent steps. In contrast, the LM update strategy using Cholesky decomposition (red dashed line) converges to a local minimum at a value as low as 10^{-15}. As expected, the QR factorization method (blue solid line) achieves even lower loss values, around 10^{-20}, within just a few hundred training steps.

Next, we demonstrate the capability of each network model for approximating the 2D discontinuous function. Using the LM optimizer, the comparison results for all models are summarized in Table 1. For the categorical embedding model, we select the reduced dimension $D = 1$. As anticipated, the one-hot encoding model, which requires the most parameters to train, achieves the most accurate predictions. However, the categorical embedding model, which has approximately half the number of parameters as the one-hot encoding model, yields nearly identical results. Furthermore, although both categorical embedding (with $D = 1$) and scalar encoding models use scalars to categorize each function piece, the categorical embedding model is certainly expected to perform better as it learns the optimal classification map in the embedding space.

Additionally, we present the network profile of the learned categorical embedding function in Fig. 5(a). As shown, the network model accurately captures all jump discontinuities sharply and represents the function well, with the pointwise absolute error, depicted

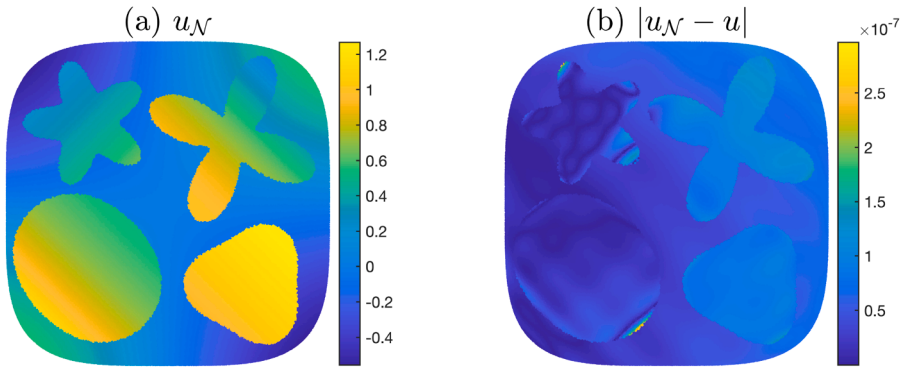


Fig. 5. (a) The trained CE model $u_{\mathcal{N}}$ for the piecewise-defined function in Example 1. (b) The pointwise absolute error $|u_{\mathcal{N}} - u|$. The maximum error is $\|u_{\mathcal{N}} - u\|_\infty = 2.96 \times 10^{-7}$.(For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Table 2
Numerical results for approximating a one-dimensional multi-piece function with different numbers of pieces in Example 2. CE: categorical embedding model; SE: scalar encoding model; OH: one-hot encoding model. "–": the method fails to converge.

Number of pieces	Method	N_p	L^2 error	L^∞ error
5	CE ($D = 1$)	205	5.65E−08	1.43E−07
	CE ($D = 2$)	260	2.76E−08	8.14E−08
	SE ($\gamma_\ell = \ell$)	200	4.02E−08	3.43E−07
	SE ($\gamma_\ell = \bar{\gamma}_\ell$)	200	9.60E−08	2.19E−07
	OH	400	2.79E−08	7.14E−08
10	CE ($D = 1$)	210	4.27E−08	3.96E−07
	CE ($D = 2$)	270	4.78E−08	1.68E−07
	CE ($D = 5$)	450	3.24E−08	1.87E−07
	SE ($\gamma_\ell = \ell$)	200	–	–
	SE ($\gamma_\ell = \bar{\gamma}_\ell$)	200	3.81E−08	5.67E−07
	OH	650	3.26E−08	1.49E−07
50	CE ($D = 1$)	250	4.03E−04	2.94E−03
	CE ($D = 2$)	350	1.71E−06	2.50E−05
	CE ($D = 5$)	650	3.89E−07	5.29E−06
	CE ($D = 10$)	1150	7.43E−08	1.47E−06
	SE ($\gamma_\ell = \bar{\gamma}_\ell$)	200	8.29E−04	8.92E−03
	OH	2650	4.29E−08	6.00E−07
100	CE ($D = 1$)	300	2.27E−03	1.99E−02
	CE ($D = 2$)	450	1.46E−05	1.92E−04
	CE ($D = 5$)	900	1.94E−07	3.83E−06
	CE ($D = 10$)	1650	9.02E−08	2.83E−06
	SE ($\gamma_\ell = \bar{\gamma}_\ell$)	200	4.47E−03	3.62E−02
	OH	5150	5.69E−08	1.43E−06

in Fig. 5(b), being on the order of 10^{-7}. It is observed that significant errors mainly occur near the interfaces between subdomains. This is because we only randomly sample the training points within the domain, but without additional information along the interfaces.

Example 2. In this example, we demonstrate the expressive power of the proposed CE network by approximating a discontinuous function comprising up to 100 segments. We select a one-dimensional domain $\Omega = [0, 2\pi]$, in which, given random variables a_ℓ, b_ℓ, and c_ℓ, the subfunctions $a_\ell \exp(\sin(b_\ell x) + \cos(c_\ell x))$ are defined within each subdomain $\Omega_\ell = (\Gamma_\ell, \Gamma_{\ell+1})$. Here, the interfaces Γ_ℓ are uniformly distributed with $\Gamma_\ell = \ell \frac{2\pi}{L+1}$. Table 2 summarizes the approximation results for the cases with 5, 10, 50, and 100 subdomains. For the loss model, 1000 training points are sampled in the first three cases, while 2000 points are used for the 100-piece case.

Notably, the one-hot encoding model consistently achieves high prediction accuracy across all cases, succeeding even in the 100-piece case with L^2 error as low as 10^{-8}. However, this model requires the largest number of trainable parameters, resulting in a substantial computational workload. By contrast, regardless of the number of segments, the design of the scalar encoding model with given labels (γ_ℓ) requires the same (and least) number of trainable parameters $N_p = 200$. When using a simple nominal label $\gamma_\ell = \ell$, the scalar encoding model performs well only for the 5-piece case but fails in the other cases. To improve the model's capability, it is natural to deepen or widen the FC structure in the scalar encoding model. However, our experiments indicate that this strategy still fails to approximate discontinuous functions with many pieces (not shown here). This issue can be easily cured by setting the mean

Table 3
The average L^2 and L^∞ errors of different methods for the two-subdomain benchmark problem with coefficient ratio $\eta = 0.1$ [47].

Method	L^2 error	L^∞ error
INN [51]	5.37E−04	2.82E−03
nDS-PINN [52]	4.73E−04	3.14E−03
coupling DR-PINN [47]	4.16E−09	3.96E−08
CE-PINN	3.10E−09	2.36E−08

of the target label $\gamma_\ell = \bar{\gamma}_\ell = \int_{\Omega_\ell} u(\mathbf{x})\, d\mathbf{x}/|\Omega_\ell|$, which can be approximated simply via Monte Carlo integration. Despite the decrease in prediction accuracy with an increasing number of segments, this labeling strategy consistently enables successful training across all the cases, emphasizing the importance of incorporating informative categorical labels.

To investigate the effect of the reduced dimension D in the categorical embedding model, we test various values of D across all the cases. For cases with 5 and 10 pieces, setting $D = 1$ is sufficient to achieve highly accurate prediction models with L^2 error as low as 10^{-8}. Increasing D to 2 or 5 provides only a minor improvement in accuracy. However, in the cases with 50 or 100 pieces, a higher-dimensional embedded space may be needed to capture more intrinsic features of the subfunctions. As a result, increasing D generally yields better approximation results. Our experiments indicate that for cases with a large number of pieces, setting the reduced dimension D to about 10% or 20% of the number of pieces yields a categorical embedding model with comparable accuracy to the one-hot encoding model, while requiring significantly fewer trainable parameters.

5.2. Examples for Poisson interface problems

To assess the performance of the proposed method, we first present a PINN-to-PINN comparison for a Poisson interface problem. We then investigate a more challenging interface problem involving multiple disconnected subdomains to illustrate the capability of the proposed embedding framework for complex subdomain configurations.

Example 1. We consider the standard two-subdomain Poisson interface benchmark problem from Example 4.4 of [47], where $\lambda = 0$ and the coefficients are piecewise constant. The exact solution is given by

$$u(x_1, x_2) = \begin{cases} 5\exp(-x_1^2 - x_2^2), & (x_1, x_2) \in \Omega_0, \\ 7 + \sin(2\pi x_1)\sin(2\pi x_2), & (x_1, x_2) \in \Omega_1. \end{cases} \tag{22}$$

Here, $A(\mathbf{x}) = A_0 = 1$ in Ω_0 and $A(\mathbf{x}) = A_1 = 0.1$ in Ω_1, corresponding to the coefficient contrast $\eta = A_1/A_0 = 0.1$. The domain Ω_0 is bounded by the five-fold flower curve $r(\theta) = 0.9 + 0.2\sin(5\theta)$, while Ω_1 is a disk of radius 0.5.

We compare CE-PINN with several representative PINN-based methods, including INN [51], nDS-PINN [52], and coupling DR-PINN [47]. The corresponding network architectures and training settings follow those reported in Table 4.6 of [47]. All methods use 2900 training points, including 2500 interior points, 200 boundary points, and the remaining points for interface jump conditions.

The results are summarized in Table 3. The proposed CE-PINN achieves accuracy comparable to coupling DR-PINN in both L^2 and L^∞ errors, while significantly outperforming INN and nDS-PINN. In particular, CE-PINN attains errors of order 10^{-9} and 10^{-8} (with standard deviation of order 10^{-10} and 10^{-9}) in L^2 and L^∞ norms, respectively. For this experiment, we use a two-hidden-layer network with 30 neurons per layer and embedding dimension $D = 1$, resulting in 1081 trainable parameters, fewer than those used in DR-PINN. The average training wall time for 10000 epochs is approximately 349 seconds.

Example 2. As the advantages of CE-PINN become more pronounced for problems involving many subdomains and complex interface geometries, we further conduct ablation studies on the embedding dimension D and network architectures for a Poisson interface problem with piecewise-constant coefficients. We consider the exact solution

$$u(x_1, x_2) = \begin{cases} \alpha_0 \exp\left(-x_1^2 - x_2^2\right) & \text{if } (x_1, x_2) \in \Omega_0, \\ \alpha_\ell + p_\ell(x_1) q_\ell(x_2) & \text{if } (x_1, x_2) \in \Omega_\ell, \end{cases} \tag{23}$$

defined on the superellipse domain $\Omega = \left\{(x_1, x_2) \in \mathbb{R}^2 \mid x_1^4 + x_2^4 \le 1\right\}$. The domain contains 16 subdomains Ω_ℓ, whose boundaries are described by polar curves $r_\ell(\theta) = a_\ell + b_\ell \cos(k_\ell \theta)$ centered at $(c_{1,\ell}, c_{2,\ell})$, for $\ell = 1, \ldots, 16$. Here, $c_{1,\ell}, c_{2,\ell}$ are chosen from $\{\pm 0.25, \pm 0.65\}$ giving 16 center points.

To demonstrate the robustness of CE-PINN, we fix $A_0 = 1$ and $\alpha_0 = 1$ on Ω_0, while the setting for each subdomain Ω_ℓ is generated using randomly selected parameters and basis functions:

$$a_\ell \in [0.1, 0.15], \quad b_\ell \in [0.02, 0.04], \quad k_\ell \in \{2, \ldots, 9\}, \quad \alpha_\ell \in \{2, \ldots, 10\},$$

and

$$A_\ell \in [0.1, 1], \quad p_\ell(x),\, q_\ell(x) \in \{\cos(2\pi x), \sin(2\pi x)\}.$$

Table 4 presents the ablation study with respect to the embedding dimension and network architecture. The first column lists the network architectures. The second column reports the embedding dimension $D = 1, 2, 4, 8$ together with the corresponding number of

Table 4
Ablation studies of CE-PINN on the embedding dimension D and network architecture for the two-dimensional Poisson interface problem with 16 subdomains.

Network	$(D,\ N_p)$	L^2 error	L^∞ error	time (sec.)
$[(2+D), 110, 1]$	(1, 567)	2.91E−05	1.84E−04	137
	(2, 694)	1.74E−06	6.98E−06	180
	(4, 948)	2.71E−07	2.00E−06	258
	(8, 1456)	3.79E−07	1.58E−06	354
$[(2+D), 20, 20, 1]$	(1, 537)	1.51E−04	8.69E−04	147
	(2, 574)	1.34E−05	7.03E−05	151
	(4, 648)	1.15E−05	4.79E−05	165
	(8, 796)	3.08E−06	1.72E−05	221
$[(2+D), 30, 30, 1]$	(1, 1097)	1.78E−06	4.26E−06	277
	(2, 1144)	3.13E−07	1.49E−06	288
	(4, 1238)	1.87E−07	8.83E−07	310
	(8, 1426)	4.58E−08	1.97E−07	396

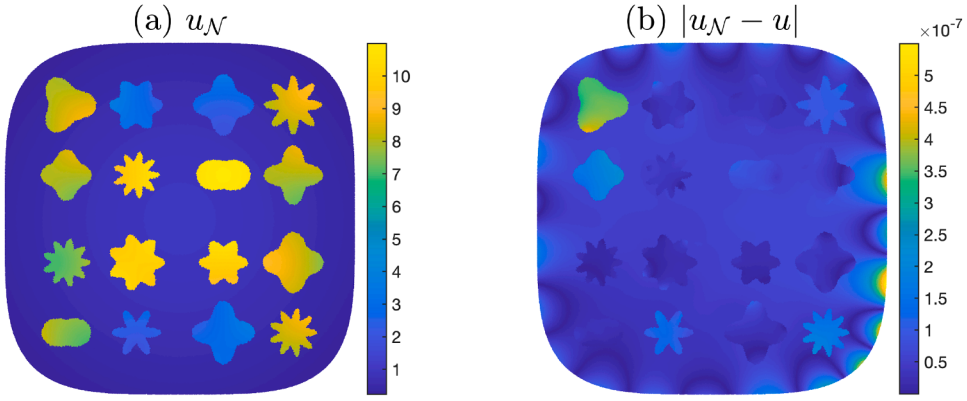


Fig. 6. (a) The trained CE model $u_{\mathcal{N}}$ for the case with $D = 8$ and network architecture $[(2 + D), 30, 30, 1]$. (b) The pointwise absolute error $|u_{\mathcal{N}} - u|$. The maximum error is $\|u_{\mathcal{N}} - u\|_\infty = 5.50 \times 10^{-7}$.

trainable parameters N_p. The remaining columns present the L^2 and L^∞ errors, and wall-clock time, respectively. For all experiments, a total of 8000 training points are used, including 6000 interior points for the PDE residual, 400 boundary points, and 1600 interface points (100 points per interface). For achieving stable and accurate results, we set the initial damping parameter to $\mu = 10^2$ and terminate the LM optimizer after 2000 training steps.

The results show that increasing the embedding dimension D generally improves the approximation accuracy, although the improvement gradually saturates once the embedding becomes sufficiently expressive. Increasing the network width or depth further improves the accuracy, but at the expense of higher wall-clock time. In Fig. 6(a), we show the scatter plot of a representative CE-PINN prediction $u_{\mathcal{N}}$ obtained with $D = 8$ and network architecture $[(2 + D), 30, 30, 1]$, while the corresponding pointwise error, of magnitude $O(10^{-7})$, is presented in Fig. 6(b). These results demonstrate that CE-PINN remains robust under moderate changes in the embedding dimension and network architecture, and that the learned embedding representation is particularly effective for interface problems involving many disconnected subdomains.

5.3. Examples for anisotropic elliptic interface problems

In this section, we present several examples of solving anisotropic elliptic interface problems in one to three dimensions. To evaluate the performance of the proposed method, for each case, we derive the terms $f(\mathbf{x})$, $g(\mathbf{x})$, $v_\ell(\mathbf{x})$, and $w_\ell(\mathbf{x})$ from the exact solution $u(\mathbf{x})$, the coefficient matrix $A(\mathbf{x})$, and the scalar function $\lambda(\mathbf{x})$. With the knowledge of these terms, we then train the model to minimize the loss function (18). In Examples 1 to 4, we deploy $N = 50$ neurons in the FC layer, while $N = 100$ neurons for Example 5. We also recall that the scalar encoding and one-hot encoding models are applied in the PINN-type loss (18) simply by fixing $E = [\gamma_0, \gamma_1, \cdots, \gamma_L]^\top$ and $E = I_{L+1}$, respectively.

Example 1.

In the first example, we demonstrate the capability of the proposed CE models for solving one-dimensional anisotropic problems with numerous jump discontinuities. Following the same setup as in Example 2 of Section 5.1, namely, we uniformly partition the domain $\Omega = [0, 2\pi]$ into subdomains $\Omega_\ell = (\Gamma_\ell, \Gamma_{\ell+1})$ with $\Gamma_\ell = \ell \frac{2\pi}{L+1}$. In each subdomain, we set the exact solution

Table 5
Numerical results for solving the one-dimensional anisotropic elliptic interface problem with different numbers of pieces in Example 1. CE: categorical embedding model; SE: scalar encoding model; OH: one-hot encoding model. "–": the method fails to converge.

Number of pieces	Method	N_p	L^2 error	L^∞ error
5	CE ($D = 1$)	205	4.37E−07	1.78E−06
	CE ($D = 2$)	260	6.13E−08	1.47E−07
	SE ($\gamma_\ell = \ell$)	200	7.63E−07	4.01E−06
	SE ($\gamma_\ell = \bar{\gamma}_\ell$)	200	6.40E−07	2.71E−06
	OH	400	2.57E−08	7.69E−08
10	CE ($D = 1$)	210	1.68E−06	4.68E−06
	CE ($D = 2$)	270	2.32E−07	6.43E−07
	SE ($\gamma_\ell = \ell$)	200	–	–
	SE ($\gamma_\ell = \bar{\gamma}_\ell$)	200	1.36E−03	3.25E−03
	OH	650	7.00E−08	2.20E−07
50	CE ($D = 5$)	650	2.38E−05	1.44E−04
	CE ($D = 10$)	1150	2.26E−05	9.29E−05
	SE ($\gamma_\ell = \bar{\gamma}_\ell$)	200	–	–
	OH	2650	3.47E−08	8.04E−07
100	CE ($D = 5$)	900	5.77E−05	3.29E−04
	CE ($D = 10$)	1650	4.97E−05	1.94E−04
	SE ($\gamma_\ell = \bar{\gamma}_\ell$)	200	–	–
	OH	5150	8.08E−08	4.23E−07

$u(x) = a_\ell \exp(\sin(b_\ell x) + \cos(c_\ell x))$, the anisotropic coefficient $A(x) = (d_\ell x)^2$, and $\lambda(x) = \sin^2(e_\ell x)$, with randomly predefined variables $(a_\ell, b_\ell, c_\ell, d_\ell, e_\ell)$. Notice that here we choose exactly the same random variables a_ℓ, b_ℓ and c_ℓ as in the function approximation case by fixing the random seed.

Table 5 shows the results for cases with 5, 10, 50, and 100 pieces, exhibiting trends similar to those observed in the function approximation tests in Section 5.1. As expected, the one-hot encoding categorization model performs effectively across all cases, achieving accuracy with L^2 errors on the order of 10^{-8}. However, this comes at the cost of a trade-off between prediction accuracy and training cost due to the large number of trainable parameters. On the other hand, the scalar encoding with nominal labeling $\gamma_\ell = \ell$ succeeds only in the 5-piece case. As encountered in the function approximation experiments, this can be addressed by assigning a more informative mean value to γ_ℓ. Here, we use the mean of the right-hand side function f by setting $\bar{f}_\ell = \int_{\Omega_\ell} f(x)\,\mathrm{d}x/|\Omega_\ell|$, and normalize those mean values to assign $\gamma_\ell = \bar{\gamma}_\ell = \bar{f}_\ell / \max_{1\le\ell\le L+1} |\bar{f}_\ell|$ in the embedding matrix since the $\bar{f}_\ell$ values range from 10^{-3} to 10^3 in this experiment. This allows the scalar encoding model to solve cases with 5 and 10 pieces, but remains insufficient for cases with more than 50 pieces, leaving an open question about selecting appropriate labels γ_ℓ.

Setting the reduced dimension to $D = 1$ or 2 in the categorical embedding model performs effectively for cases with 5 and 10 pieces, achieving accuracy comparable to that of the one-hot encoding model. However, unlike in the function approximation context, a low-dimensional embedded space ($D = 1$ or 2) in 50- and 100-piece cases may be inadequate to capture the intrinsic features of the network solution at the PDE level using PINN learning machinery. As observed, increasing the reduced dimension D up to 5 and 10 significantly enhances the model's capability, allowing it to learn the complex network solutions required for the 50- and 100-piece cases.

Remark 1. The key distinction between SE and CE models lies in the representation of subdomain information. In SE framework, each subdomain is encoded by a fixed scalar label $z(\mathbf{x}) = \gamma_\ell \in \mathbb{R}$, which provides a relatively simple representation of discontinuities. In contrast, CE framework replaces this scalar encoding with a learnable vector representation through the embedding matrix $E \in \mathbb{R}^{D\times(L+1)}$, thereby introducing a higher-dimensional and trainable representation of the subdomain structure. This modification allows the network to learn more expressive feature representations across interfaces, rather than relying on manually prescribed scalar identifiers. Such flexibility becomes particularly useful in problems involving a large number of subdomains, where scalar labels may be insufficient to capture complex interactions between subdomains. In these settings, the learned embedding features often lead to improved approximation accuracy and more stable training behavior. For simpler configurations, such as problems with only a few subdomains, the improvement may be less pronounced, which is consistent with our numerical observations.

Example 2. Next, we turn to solving a two-dimensional problem with anisotropic variable coefficients, which often serves as a benchmark for various numerical methods; see [11,12,53]. The designated domain is a regular square $\Omega = [-1, 1]^2$ containing a heart-shaped interface $\Gamma_1 = \{(r(\theta)\cos\theta - 0.25, r(\theta)\sin\theta) | r(\theta) = (1 + \cos\theta)/3, \theta \in [0, 2\pi)\}$, which partitions the domain into two subdomains, $\Omega = \Omega_0 \cup \Omega_1$. The exact solution is given by

$$u(x_1, x_2) = \begin{cases} x_1^2 + x_2^2 & \text{if } (x_1, x_2) \in \Omega_0, \\ \exp(x_1)\cos(x_2) & \text{if } (x_1, x_2) \in \Omega_1. \end{cases}$$

Table 6
Numerical results for solving the two-dimensional anisotropic interface problem in Example 2. CE: categorical embedding model; SE: scalar encoding model; OH: one-hot encoding model; FVM: finite volume method [11].

Method	N_p	L^2 error	L^∞ error
CE ($D = 1$)	252	5.97E−08	5.43E−07
SE ($\gamma_\ell = \ell$)	250	6.62E−08	6.23E−07
OH	300	1.47E−08	1.34E−07
FVM [11]	16384	1.80E−04	–

Define

$$A_1(x_1, x_2) = \begin{bmatrix} x_1^2 + x_2^2 + 1 & x_1^2 + x_2^2 \\ x_1^2 + x_2^2 & x_1^2 + x_2^2 + 2 \end{bmatrix},$$

and $\lambda_1(x_1, x_2) = \exp(x_1)(x_1^2 + x_2^2 + 3)\sin(x_2)$, the anisotropic coefficient A and the scalar function λ are piecewise spatial dependent functions set by

$$A(x_1, x_2) = \begin{cases} 1000 A_1(x_1, x_2) & \text{if } (x_1, x_2) \in \Omega_0, \\ A_1(x_1, x_2) & \text{if } (x_1, x_2) \in \Omega_1, \end{cases}$$

$$\lambda(x_1, x_2) = \begin{cases} 1000 \lambda_1(x_1, x_2) & \text{if } (x_1, x_2) \in \Omega_0, \\ \lambda_1(x_1, x_2) & \text{if } (x_1, x_2) \in \Omega_1, \end{cases}$$

so that the contrasts for both functions are 1000.

In this test, we train the loss model using randomly selected training points with $(M, M_b, M_{\Gamma_1}) = (324, 72, 72)$. The results, compared with those from the finite volume method (FVM) [11], are presented in Table 6. Notice that for FVM, the total number of degrees of freedom (or unknowns) equals the number of discretization grid points, m^2. As shown in Table 6, FVM uses a grid resolution of $m = 128$, resulting in $N_p = 16384$ unknowns, while the neural network models require only a few hundred parameters. As noted, with a relatively small training dataset (a few hundred points) and a moderate number of trainable parameters (also in the few hundred range), all neural network models achieve higher accuracy than FVM. Given that the solution involves only two subdomain solutions, all models are capable of achieving similar levels of prediction accuracy.

From a computational perspective, classical FVM for a two-dimensional elliptic problem leads to a sparse linear system with $n = m^2$ unknowns and near-linear complexity in n when efficient solvers are employed. In contrast, the neural network-based approach involves a nonlinear optimization over N_p parameters using M_{tot} collocation points. When the LM method is applied, each iteration requires forming a Jacobian matrix J of size $M_{tot} \times N_p$ and solving a dense linear system associated with $J^\top J$, leading to significantly higher per-iteration computational and memory requirements compared to FVM. Consequently, while LM often provides fast and stable local convergence for moderate values of N_p and M_{tot}, its scalability is limited for large-scale networks. In the present work, the proposed CE-PINN employs relatively compact architectures, for which LM remains computationally feasible and effective.

The categorical embedding solution with reduced dimension $D = 1$ is depicted in Fig. 7(a). As shown, the model captures the jump discontinuity sharply along the interface, and the absolute error, depicted in Fig. 7(b), exhibits a pointwise absolute error as low as 10^{-8}, demonstrating the high prediction accuracy of our proposed model. It is worth noting that, in this example, a cusp occurs at $(-0.25, 0)$; our discontinuity-capturing models can handle such interfaces without difficulty. In contrast, some existing numerical methods may encounter challenges with these singular points.

In the proposed CE-PINN framework, the interface locations are explicitly prescribed, enabling the direct enforcement of interface jump conditions via dedicated collocation points on the interfaces. In practice, sufficiently many training points are sampled both within the bulk subdomains and along the interfaces to ensure adequate enforcement of the interface conditions during training.

To examine the sensitivity to collocation point distributions near the interfaces, we conducted additional numerical tests by varying the sampling density around the interfaces. Our numerical results indicate that the proposed method is relatively robust with respect to the distribution of sampling points near the interfaces (not shown here). We also note that residual-based adaptive sampling [54] provides an effective strategy, especially for problems in which the interface locations are unknown or implicitly defined. In the present setting, however, the interface geometry is known a priori and directly sampled, so the additional benefit of adaptive refinement near the interfaces appears to be relatively limited.

Example 3. In this example, we consider a benchmark for a 2D variable-coefficient elliptic interface problem; see Example 4 in [55]. With a square domain, $\Omega = [-1, 1]^2$, the interface is described as the longitude and latitude of a "chessboard"-like domain (see the left panel of Fig. 8) given by the zero level set of $\phi(x_1, x_2) = (\sin(5\pi x_1) - x_2)(-\sin(5\pi x_2) - x_1)$. Thus we define the separated domain

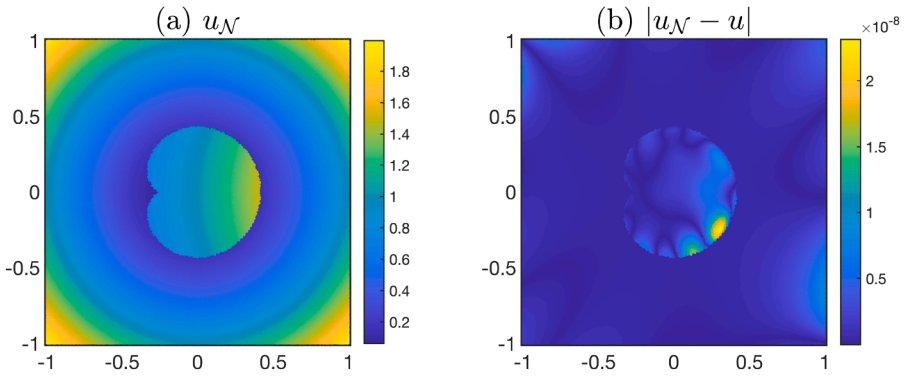


Fig. 7. (a) The trained CE model $u_{\mathcal{N}}$ for the piecewise-defined solution of Example 2. (b) The pointwise absolute error $|u_{\mathcal{N}} - u|$. The maximum error is $\|u_{\mathcal{N}} - u\|_\infty = 2.31 \times 10^{-8}$.

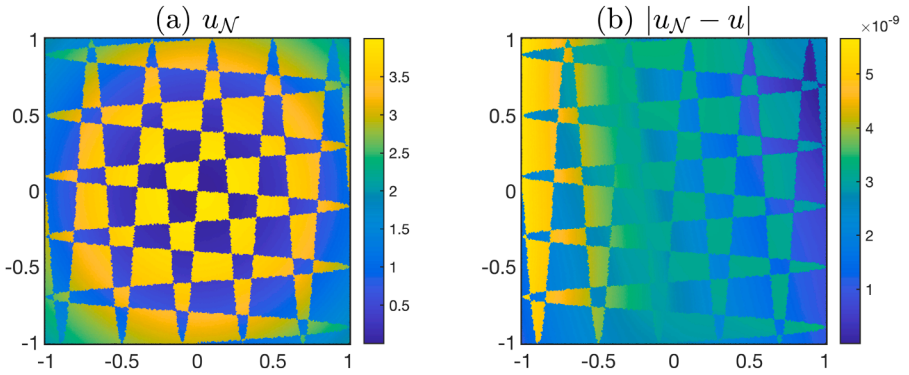


Fig. 8. (a) The trained categorical embedding solution $u_{\mathcal{N}}$ with $D = 1$ of Example 3. (b) The pointwise absolute error $|u_{\mathcal{N}} - u|$. The maximum error is $\|u_{\mathcal{N}} - u\|_\infty = 5.65 \times 10^{-9}$.

$\Omega_0 = \{(x_1, x_2) | \phi(x_1, x_2) > 0\}$ and $\Omega_1 = \{(x_1, x_2) | \phi(x_1, x_2) < 0\}$. The diffusion coefficient is given by

$$A(x_1, x_2) = \begin{cases} x_1 x_2 + 2 & \text{if } (x_1, x_2) \in \Omega_0, \\ x_1^2 - x_2^2 + 3 & \text{if } (x_1, x_2) \in \Omega_1. \end{cases}$$

We set $\lambda = 0$ and the exact solution

$$u(x_1, x_2) = \begin{cases} 4 - x_1^2 - x_2^2 & \text{if } (x_1, x_2) \in \Omega_0, \\ x_1^2 + x_2^2 & \text{if } (x_1, x_2) \in \Omega_1. \end{cases}$$

We follow the same setup as in Example 2. The categorical embedding prediction solution with $D = 1$ along with its pointwise absolute error is displayed in Fig. 8. The present model achieves a very accurate result with the L^∞ error, $\|u_{\mathcal{N}} - u\|_\infty = 5.65 \times 10^{-9}$.

We summarize the accuracy between the present network models, and that of the finite element method (FEM) [55] in Table 7. Indeed, the prediction accuracy of all models clearly outperforms that of FEM; each network model with a few hundred trainable parameters readily achieves high solution expressivity, with an L^2 error of order 10^{-9}. It is worth noting that, although this problem can be dealt appropriately with traditional numerical methods such as FVM or FEM, these methods require identifying regular and irregular grid points (or cell triangulations) as a preliminary step, making their implementation somewhat tedious. On the contrary, it is straightforward to implement the present network models simply using the categorization map $\mathbf{z}$. Moreover, we randomly sampled $m_{\Gamma_1} = 72$ points to represent the interface, and the optimization algorithm's implementation requires no additional effort compared to Examples 1 and 2, demonstrating the robustness of the present method regardless of the complexity of embedded interface geometries.

Example 4. In this example, we aim to highlight the capability of the present method by tackling two-dimensional problems with multiple subdomains enclosed in an irregular domain. Here, the domain and subdomain geometries are shown in Fig. 1, while the detailed formulation can be found in Example 1 of Section 5.1. The solution profile is given in Eq. (21); the coefficient matrix A and scalar function λ in Ω_0 are respectively set by

$$A_0(x_1, x_2) = \begin{bmatrix} (x_1 + x_2)^2 + 1 & -x_1^2 + x_2^2 \\ -x_1^2 + x_2^2 & (x_1 - x_2)^2 + 1 \end{bmatrix} \quad \text{and} \quad \lambda_0(x_1, x_2) = \exp(x_1 - x_2),$$

Table 7
Numerical results for solving the two-dimensional anisotropic interface problem in Example 3. CE: categorical embedding model; SE: scalar encoding model; OH: one-hot encoding model; FEM: finite element method [55].

Method	N_p	L^2 error	L^∞ error
CE ($D = 1$)	252	4.39E−09	9.59E−09
SE ($\gamma_\ell = \ell$)	250	5.48E−09	1.25E−08
OH	300	4.13E−09	7.89E−09
FEM [55]	102400		2.60E−05

Table 8
Numerical results for solving the two-dimensional anisotropic interface problem in Example 4. CE: categorical embedding model; SE: scalar encoding model; OH: one-hot encoding model.

Method	N_p	L^2 error	L^∞ error
CE ($D = 1$)	255	2.33E−09	2.28E−08
SE ($\gamma_\ell = \ell$)	250	3.96E−09	3.54E−08
OH	450	2.95E−09	2.09E−08

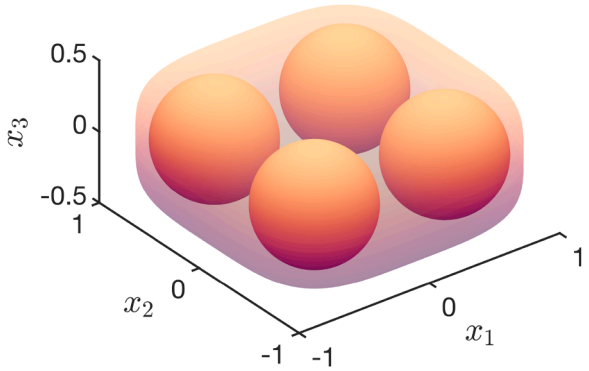


Fig. 9. The super-quadric domain $x_1^4 + x_2^4 + 16x_3^4 = 1$ with four embedded spheres of radius 0.4 located at $(-0.45, 0.45, 0)$, $(0.45, 0.45, 0)$, $(-0.45, -0.45, 0)$, and $(0.45, -0.45, 0)$.

while in the other subdomains, we set $A_\ell(x_1, x_2) = \beta_\ell A_0(x_1, x_2)$ and $\lambda_\ell(x_1, x_2) = \beta_\ell \lambda_0(x_1, x_2)$, where $\beta_1 = 10^{-1}$, $\beta_2 = 10^{-2}$, $\beta_3 = 10^1$, and $\beta_4 = 10^2$ (so the largest ratio in this case is 10000).

Following the same training setup as in Example 2, each solution model achieves high prediction accuracy, with L^2 errors as low as 10^{-9}, as shown in Table 8. Notably, at the PDE-solving level, the categorical embedding model requires only about half as many trainable parameters as the one-hot encoding model. Additionally, owing to the mesh-free nature of the neural network method, implementing the model is straightforward, as demonstrated in this test with the superellipse. In contrast, grid-based methods require substantially greater implementation effort to solve problems on irregular domains with multiple subdomains.

Example 5. In the last example, we illustrate the robustness of our method for solving a three-dimensional anisotropic problem with multiple interfaces. We set the domain Ω with a super-quadric boundary $x_1^4 + x_2^4 + 16x_3^4 = 1$, in which there are four subdomains, Ω_1, Ω_2, Ω_3, and Ω_4, encapsulated by four spheres of radius 0.4 with their center respectively located at $(-0.45, 0.45, 0)$, $(0.45, 0.45, 0)$, $(-0.45, -0.45, 0)$, and $(0.45, -0.45, 0)$. See the domain depiction in Fig. 9.

Denoting $\mathbf{x} = (x_1, x_2, x_3)$, we set the solution as

$$u(\mathbf{x}) = \begin{cases} \exp(x_1 + x_2 + x_3) & \text{if } \mathbf{x} \in \Omega_0, \\ \sin x_1 \sin x_2 \sin x_3 & \text{if } \mathbf{x} \in \Omega_1, \\ \cos x_1 \cos x_2 \cos x_3 & \text{if } \mathbf{x} \in \Omega_2, \\ \sinh x_1 \sinh x_2 \sinh x_3 & \text{if } \mathbf{x} \in \Omega_3, \\ \cosh x_1 \cosh x_2 \cosh x_3 & \text{if } \mathbf{x} \in \Omega_4, \end{cases} \tag{24}$$

Table 9
Numerical results for solving the three-dimensional anisotropic interface problem in Example 5. CE: categorical embedding model; SE: scalar encoding model; OH: one-hot encoding model.

Method	N_p	L^2 error	L^∞ error
CE ($D = 1$)	605	3.42E−08	2.64E−07
SE ($\gamma_\ell = \ell$)	600	2.87E−07	3.45E−06
OH	1000	4.25E−08	5.51E−07

and choose $A_0(\mathbf{x}) = R\Lambda R^T$ and $\lambda_0(\mathbf{x}) = \exp(x_1 - x_2 - x_3)$, where

$$R = \begin{bmatrix} 2/3 & 1/3 & 2/3 \\ -2/3 & 2/3 & 1/3 \\ 1/3 & 2/3 & -2/3 \end{bmatrix} \quad \text{and} \quad \Lambda = \begin{bmatrix} \|\mathbf{x}\|^2 + 1 & 0 & 0 \\ 0 & \|\mathbf{x}\|^2 + 2 & 0 \\ 0 & 0 & \|\mathbf{x}\|^2 + 3 \end{bmatrix}.$$

We further define $A_\ell = \beta_\ell A_0$ and $\lambda_\ell = \beta_\ell \lambda_0$, where $\beta_1 = 0.1$, $\beta_2 = 0.05$, $\beta_3 = 10$, and $\beta_4 = 50$, resulting in a maximum contrast ratio of 1000. In Table 9, we train each model using $(M, M_b, M_\Gamma) = (324, 144, 144)$. In this setup, the categorical embedding network with $D = 1$ and the one-hot encoding model exhibit similar performance, achieving approximately one order of magnitude higher prediction accuracy than the scalar encoding model. Notably, the categorical embedding model learns the categorization map using a single-dimensional representation. This results in roughly half as many trainable parameters as the one-hot encoding model, significantly reducing computational effort.

6. Conclusion

In this work, we propose a single neural network architecture, a categorical embedding neural network, for representing piecewise-smooth functions. The architecture consists of three hidden layers: (i) a discontinuity-capturing layer, which maps domain segments to disconnected sets in a higher-dimensional space; (ii) a categorical embedding layer, which reduces the high-dimensional information into low-dimensional features; (iii) a fully connected layer, which models the continuous mapping. This design enables accurate approximation of piecewise-smooth functions, even when a function contains a large number of pieces. Moreover, the proposed network possesses two notable features: it allows direct calculation of jump values at interfaces, and it ensures well-defined derivatives at all points away from interfaces and domain boundaries, thereby facilitating derivative calculations.

We further apply the proposed categorical embedding neural network to solve anisotropic elliptic interface problems, which are traditionally regarded as highly challenging tasks. Follow the PINN framework, the network is trained using the LM optimizer by minimizing the mean squared error loss of the system. With such a simple design and shallow network architecture, the model achieves accuracy and efficiency comparable to established grid-based numerical methods. Importantly, the approach is entirely mesh-free: once training points are appropriately selected, the loss formulation and optimization process remain unchanged across different domains and subdomain geometries.

Finally, our results demonstrate that machine learning-based methods can achieve accuracy comparable to conventional scientific computing approaches, while offering additional advantages. These include their mesh-free nature, ease of implementation, natural compatibility with GPU acceleration, and flexibility for extension to high-dimensional problems. Such properties suggest strong potential for practical applications in computational physics and engineering, where both accuracy and efficiency are critical. Although rigorous approximation and convergence analysis for neural network methods applied to interface problems remains largely open, recent work for linear second-order elliptic interface problems has established convergence results for PINNs under a domain decomposition setting. In particular, under suitable assumptions and carefully designed loss formulations, the neural network solutions are shown to converge to the true solution in Sobolev norms as the number of training samples increases [56]. Extending such analysis to more general network architectures, adaptive sampling strategies, and complex interface geometries remains an important direction for future research.

CRediT authorship contribution statement

Wei-Fan Hu: Writing – review & editing, Writing – original draft, Visualization, Methodology, Conceptualization; **Te-Sheng Lin:** Writing – review & editing, Writing – original draft, Methodology, Conceptualization; **Yu-Hau Tseng:** Writing – review & editing, Writing – original draft, Conceptualization; **Ming-Chih Lai:** Writing – review & editing, Writing – original draft, Conceptualization.

Declaration of competing interest

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

Code availability

The source code used in this work is available from the authors upon request.

Acknowledgment

W.-F. Hu, T.-S. Lin, Y.-H. Tseng, and M.-C. Lai acknowledge the supports by National Science and Technology Council, Taiwan, under research grants 114-2628-M-008-002-MY4, 113-2115-M-390-007-MY2, and respectively. W.-F. Hu and T.-S. Lin also acknowledge the supports by National Center for Theoretical Sciences, Taiwan.

Data availability

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

References

[1] F. Wang, A. Dong, W.E. Buhro, Solution-liquid-solid synthesis, properties, and applications of one-dimensional colloidal semiconductor nanorods and nanowires, Chem. Rev. 116 (2016) 10888–10933.
[2] C. Wang, H. Dong, L. Jiang, W. Hu, Organic semiconductor crystals, Chem. Soc. Rev. 47 (2018) 422–500.
[3] E.Y. Tsymbal, I. Zutic, Spin Transport and Magnetism, CRC Press, 2012.
[4] L. Kondic, M.J. Shelley, P. Palffy-Muhoray, Non-Newtonian Hele-Shaw flow and the Saffman-Taylor instability, Phys. Rev. Lett. 80 (1998) 1433.
[5] P. Fast, L. Kondic, M.J. Shelley, P. Palffy-Muhoray, Pattern formation in non-Newtonian Hele-Shaw flow, Phys. Fluids 13 (2001) 1191–1212.
[6] Y. Jeon, High order immersed hybridized difference methods for elliptic interface problems, J. Numer. Math. 32 (2023) 139–156.
[7] H. Zhou, W. Ying, A correction function-based kernel-free boundary integral method for elliptic PDEs with implicitly defined interfaces, J. Comput. Phys. 496 (2024) 112545.
[8] K. Lipnikov, M. Shashkov, D. Svyatskiy, Y. Vassilevski, Monotone finite volume schemes for diffusion equations on unstructured triangular and shape-regular polygonal meshes, J. Comput. Phys. 227 (2007) 492–512.
[9] F. Zhao, X. Lai, G. Yuan, Z. Sheng, A new interpolation for auxiliary unknowns of the monotone finite volume scheme for 3D diffusion equations, Commun. Comput. Phys. 27 (2020) 1201–1233.
[10] C. Lu, Z. Yang, J. Bai, Y. Cao, X. He, Three-dimensional immersed finite-element method for anisotropic magnetostatic/electrostatic interface problems with nonhomogeneous flux jump, Int. J. Numer. Methods Eng. 121 (2020) 2107–2127.
[11] K. Pan, X. Wu, Y. Xu, G. Yuan, An exact-interface-fitted mesh generator and linearity-preserving finite volume scheme for anisotropic elliptic interface problems, J. Comput. Phys. 463 (2022) 111293.
[12] Y. Xing, H. Zheng, A high order generalized finite difference method for solving the anisotropic elliptic interface problem in static and moving systems, Comput. Math. Appl. 166 (2024) 1–23.
[13] B. Dong, X. Feng, Z. Li, An L^∞ second order Cartesian method for 3D anisotropic interface problems, J. Comp. Math. 40 (2022) 882–912.
[14] H. Ji, Z. Li, An immersed finite element method for anisotropic elliptic interface problems with nonhomogeneous jump conditions, J. Sci. Comput. 106 (2026) 1–29.
[15] L. Li, C. Yang, APFOS-NET: Asymptotic preserving scheme for anisotropic elliptic equations with deep neural network, J. Comput. Phys. 453 (2022) 110958.
[16] S. Pakravan, P.A. Mistani, M.A. Aragon-Calvo, F. Gibou, Solving inverse-PDE problems with physics-aware neural networks, J. Comput. Phys. 440 (2021) 110414.
[17] Z. Geng, D. Johnson, R. Fedkiw, Coercing machine learning to output physically accurate results, J. Comput. Phys. 406 (2020) 109099.
[18] G. Strang, G. Fix, An Analysis of the Finite Element Method, Prentice-Hall, 1973.
[19] A.R. Barron, Universal approximation bounds for superpositions of a sigmoidal function, IEEE Trans. Inf. Theory 39 (2002) 930–945.
[20] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Contr. Signals, Syst. 2 (1989) 303–314.
[21] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (1989) 359–366.
[22] A. Pinkus, Approximation theory of the MLP model in neural networks, Acta Numer. 8 (1999) 143–195.
[23] M.W.M.G. Dissanayake, N. Phan-Thien, Neural-network-based approximations for solving partial differential equations, Commun. Numer. Methods Eng. 10 (1994) 195–201.
[24] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[25] W. E, B. Yu, The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. 6 (2018) 1–12.
[26] L. Lu, P. Jin, G. Pang, Z. Zhang, G.E. Karniadakis, Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, Nat. Mach. Intell. 3 (2021) 218–229.
[27] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations (2020). arXiv:2010.08895
[28] B. Llanas, S. Lantaron, F.J. Sainz, Constructive approximation of discontinuous functions by neural networks, Neural Process. Lett. 27 (2008) 209–226.
[29] R.R. Selmic, F.L. Lewis, Neural-network approximation of piecewise continuous functions: application to friction compensation, IEEE Trans. Neural Netw. 13 (2002) 745–751.
[30] M. Forti, P. Nistri, Global convergence of neural networks with discontinuous neuron activations, IEEE Trans. Circuits Syst. I 50 (2003) 1421–1435.
[31] C. Wen, X. Ma, A max-piecewise-linear neural network for function approximation, Neurocomputing 71 (2008) 843–852.
[32] Q. Liu, J. Wang, A one-layer recurrent neural network with a discontinuous hard-limiting activation function for quadratic programming, IEEE Trans. Neural Netw. 19 (2008) 558–570.
[33] C. He, X. Hu, L. Mu, A mesh-free method using piecewise deep neural network for elliptic interface problems, J. Comput. Phys. 412 (2022) 114358.
[34] M.-C. Lai, C.-C. Chang, W.-S. Lin, W.-F. Hu, T.-S. Lin, A shallow Ritz method for elliptic problems with singular sources, J. Comput. Phys. 469 (2022) 111547.
[35] P. Mistani, S. Pakravan, R. Ilango, F. Gibou, JAX-DIPS: neural bootstrapping of finite discretization methods and application to elliptic problems with discontinuities, J. Comput. Phys. 493 (2023) 112480.
[36] W.-F. Hu, T.-S. Lin, M.-C. Lai, A discontinuity capturing shallow neural network for elliptic interface problems, J. Comput. Phys. 469 (2022) 111576.
[37] D. Barreca, A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems, SIGKDD Explor. 3 (2001) 27–32.
[38] H. Whitney, Analytic extensions of differentiable functions defined in closed sets, Trans. Amer. Math. Soc. 36 (1934) 63–89.
[39] R.T. Seeley, Extension of C^∞ functions defined in a half space, Proc. Amer. Math. Soc. 15 (1964) 625–626.
[40] M. Hou, Y. Chen, S. Cao, Y. Chen, J. Ying, HRW: hybrid residual and weak form loss for solving elliptic interface problems with neural network, Numer. Math. Theor. Meth. Appl. 16 (2023) 883–913.
[41] C. Guo, F. Berkhahn, Entity embeddings of categorical variables (2016). arXiv:1604.06737
[42] D. Marquardt, An algorithm for least-squares estimation of nonlinear parameters, SIAM J. Appl. Math. 11 (1963) 431–441.
[43] B. Hanin, M. Sellke, Adam: A method for stochastic optimization (2018). arXiv:1710.11278

[44] D. Liu, J. Nocedal, On the limited memory BFGS method for large scale optimization, Math. Program. 45 (1989) 503–528.
[45] M.T. Hagan, M.B. Menhaj, Training feedforward networks with the Marquardt algorithm, IEEE Trans. Neural Netw. 5 (1994) 989–993.
[46] M.K. Transtrum, J.P. Sethna, Improvements to the Levenberg–Marquardt algorithm for nonlinear least-squares minimization, 2012, arXiv:1201.5885
[47] H. Fan, Z. Tan, Novel and general discontinuity-removing PINNs for elliptic interface problems, J. Comput. Phys. 529 (2025) 113861.
[48] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient flow pathologies in physics-informed neural networks, SIAM J. Sci. Comput. 43 (2021) A3055–A3081.
[49] S. Wang, X. Yu, P. Perdikaris, When and why PINNs fail to train: a neural tangent kernel perspective, J. Comput. Phys. 449 (2022) 110768.
[50] L.D. McClenny, U.M. Braga-Neto, Self-adaptive physics-informed neural networks, J. Comput. Phys. 474 (2023) 111722.
[51] S. Wu, B. Lu, INN: interfaced neural networks as an accessible meshless approach for solving interface PDE problems, J. Comput. Phys. 470 (2022) 111588.
[52] Y. Yao, J. Guo, T. Gu, A deep learning method for multi-material diffusion problems based on physics-informed neural networks, Comput. Methods Appl. Mech. Eng. 417 (2023) 116395.
[53] B. Dong, X. Feng, Z. Li, An FE-FD method for anisotropic elliptic interface problems, SIAM J. Sci. Comput. 42 (2020) B1041–B1066.
[54] C. Wu, M. Zhu, Q. Tan, Y. Kartha, L. Lu, A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks, Comput. Methods Appl. Mech. Eng. 403 (2023) 115671.
[55] S. Hou, W. Wang, L. Wang, Numerical method for solving matrix coefficient elliptic equation with sharp-edged interfaces, J. Comput. Phys. 229 (2010) 7162–7179.
[56] S. Wu, A. Zhu, Y. Tang, B. Lu, Convergence of physics-informed neural networks applied to linear second-order elliptic interface problems, Comm. Comput. Phys. 33 (2023) 596–627.