Journal of Computational and Applied Mathematics 445 (2024) 115830

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

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

Local randomized neural networks with discontinuous Galerkin methods for partial differential equations

Jingbo Sun [a,1], Suchuan Dong [b,2], Fei Wang [a,1,*]

[a] School of Mathematics and Statistics, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, PR China
[b] Center for Computational and Applied Mathematics, Department of Mathematics, Purdue University, West Lafayette, IN, USA

ARTICLE INFO

MSC:
65N30
41A46

Keywords:
Randomized neural networks
Discontinuous Galerkin method
Partial differential equations
Least-squares method
Space–time approach

ABSTRACT

Randomized Neural Networks (RNNs) are a variety of neural networks in which the hidden-layer parameters are fixed to randomly assigned values, and the output-layer parameters are obtained by solving a linear system through least squares. This improves the efficiency without degrading the accuracy of the neural network. In this paper, we combine the idea of the Local RNN (LRNN) and the Discontinuous Galerkin (DG) approach for solving partial differential equations. RNNs are used to approximate the solution on the subdomains, and the DG formulation is used to glue them together. Taking the Poisson problem as a model, we propose three numerical schemes and provide convergence analysis. Then we extend the ideas to time-dependent problems. Taking the heat equation as a model, three space–time LRNN with DG formulations are proposed. Finally, we present numerical tests to demonstrate the performance of the methods developed herein. We evaluate the performance of the proposed methods by comparing them with the finite element method and the conventional DG method. The LRNN-DG methods can achieve higher accuracy with the same degrees of freedom, and can solve time-dependent problems more precisely and efficiently. This indicates that this new approach has great potential for solving partial differential equations.

1. Introduction

Artificial Neural Networks (ANNs) have been successfully applied to solve problems of segmentation, classification, pattern recognition, automatic control, etc. In recent years, many research works based on neural networks have been proposed for solving Partial Differential Equations (PDEs) due to the excellent approximation capability of Neural Networks (NNs). Some of these contributions are based on the strong form of PDEs. Physical Informed Neural Networks (PINNs) [1] and the Deep Galerkin Method [2] are two representative methods among them. Specifically, PINNs train the neural network by minimizing the mean squared error loss consisting of information about the PDE, boundary conditions, and/or initial conditions on certain collocation points. The loss function of the Deep Galerkin Method measures the residual of the PDE in the sense of the integral. Based on PINNs, a number of new models [3–10] have been proposed aiming to improve the performance, and some other studies [11–15] focus on the application of this technique for different kinds of problems.

* Corresponding author.
E-mail addresses: jingbosun@stu.xjtu.edu.cn (J. Sun), sdong@purdue.edu (S. Dong), feiwang.xjtu@xjtu.edu.cn (F. Wang).

[1] The work of this author was partially supported by the National Natural Science Foundation of China (Grant No. 12171383) and Shaanxi Fundamental Science Research Project for Mathematics and Physics (Grant No. 22JSY027).
[2] The work of this author was partially supported by the US National Science Foundation (DMS-2012415).

https://doi.org/10.1016/j.cam.2024.115830
Received 5 May 2023; Received in revised form 5 December 2023
Available online 10 February 2024

Some problems have solutions with low regularity, which cannot be described by PDEs. Therefore, some investigations on neural networks use loss functions constructed based on weak formulations, such as the Deep Ritz method [16], Deep Nitsche method [17], Weak Adversarial Networks [18], and other methods [19,20]. One issue with these neural network-based methods, whether in strong or weak forms, is their limited accuracy and time-consuming nature. Although neural networks have a strong capability for function approximation, they are challenging to train to reach the global optimal state due to the lack of efficient optimization methods for the training process. Additionally, the computational cost of network training is high, hindering practical applications. In terms of accuracy and efficiency, these neural network-based methods generally cannot compete with traditional methods such as the finite element method (FEM), finite difference method, and finite volume method.

Randomized neural networks (RNNs) have been proposed as an alternative approach to fully parameterized neural network models [21–24]. In RNNs, the parameters of the links between the hidden layers are randomly chosen and then fixed during training, while the parameters for the links between the last hidden layer and output layer are determined using a least-squares method. The extreme learning machine (ELM) [25] is an example of a randomized neural network that has been successfully applied to various problems [26–32], including the solution of differential equations [33–37]. The feasibility analysis of ELM was proved in [38], which demonstrates that the generalization capability of ELM is similar to that of fully parameterized neural networks when suitable activation functions and initialization strategies are properly selected for the fixed parameters. To solve partial differential equations, Dong and Li proposed the local extreme learning machine and domain decomposition (locELM-DD) method in [39], which combines the ideas of local ELM and domain decomposition to improve accuracy and efficiency. However, locELM-DD is based on the strong form of PDE problems and may not be suitable for problems that require a weak formulation. In [40], the deep Petrov–Galerkin method is proposed based on ELM and the Petrov–Galerkin formulation for solving partial differential equations, and numerical examples show that this approach is accurate and efficient with respect to degrees of freedom (DoF).

In this work, we focus on the weak formulations of partial differential equations (PDEs) and their approximations by local neural networks. We combine local randomized neural networks with the discontinuous Galerkin (DG) approach and seek to exploit the DG framework to glue the local neural networks together. Specifically, we use the Poisson equation and the heat equation as model problems to develop three schemes and show how to implement them. The first scheme is the LRNN-DG (local randomized neural networks with discontinuous Galerkin) method, which uses the output fields of the last hidden layer as the local basis functions for the DG formulation on each subdomain. It then solves the final system of linear equations using either a linear solver or a least-squares method. The other two schemes are the LRNN-C^0DG (local randomized neural networks with C^0 discontinuous Galerkin) and LRNN-C^1DG (local randomized neural networks with C^1 discontinuous Galerkin) methods. These methods enforce continuity conditions for the function and its gradients across sub-domain boundaries. We provide a convergence analysis of these methods under certain appropriate assumptions. For the time-dependent problem, we use the heat equation as a model and propose three space–time LRNN-DG type formulations. The space–time approach is very natural for neural networks, and we do not need to compute the numerical solution with time iteration. Finally, we present numerical examples to show that the proposed methods are able to compete with traditional methods in some aspects. First, when the number of degrees of freedom is fixed and small, the accuracies of the LRNN/DG type methods developed herein are better than the finite element method (FEM) and the usual DG methods. Second, when the time–space approach is adopted, a notable advantage is that the error accumulation can be avoided, and one can obtain the numerical solution at any time instant without interpolation.

The remainder of this paper is structured as follows. In Section 2, we introduce the concept of randomized neural networks and propose three LRNN-DG formulations to solve the Poisson equation. In Section 3, we present a convergence analysis of the methods by making certain assumptions. In Section 4, we provide three space–time LRNN-DG methods to solve the heat equation. In Section 5, we present some numerical examples to demonstrate the performance of the proposed methods. Finally, we summarize our findings in the last section.

2. Local randomized neural networks with DG methods

In this section, we first describe randomized neural networks, then we introduce the local randomized neural networks with discontinuous Galerkin formulations for solving the Poisson equation.

2.1. Randomized neural networks

The general deep neural networks can be represented as compositions of many hidden layers and an output layer. A hidden layer is defined as a composition of a linear transformation and an activation function:

$$N(\boldsymbol{x}) = \rho(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}),$$

where $\boldsymbol{x} \in \Omega \subset \mathbb{R}^d$, $N(\boldsymbol{x}) \in \mathbb{R}^{\tilde{d}}$, $\boldsymbol{W} \in \mathbb{R}^{\tilde{d}\times d}$ is the matrix of weights, $\boldsymbol{b} \in \mathbb{R}^{\tilde{d}}$ is the bias, and ρ is a nonlinear activation function. The first layer is usually called the input layer, and the number of layers is the depth of the neural network. The output layer is a linear transformation

$$N^o(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b},$$

where $\boldsymbol{x} \in \mathbb{R}^{\tilde{d}}$, $N^o(\boldsymbol{x}) \in \mathbb{R}^{n_o}$, $\boldsymbol{W} \in \mathbb{R}^{n_o\times\tilde{d}}$ is the weight, and $\boldsymbol{b} \in \mathbb{R}^{n_o}$ is the bias. Here, n_o is the dimension of output data.

Then a fully connected neural network can be represented by

$$\mathcal{U}(\boldsymbol{x}) = \boldsymbol{W}^{(L+1)}(N^{(L)} \circ \cdots N^{(2)} \circ N^{(1)}(\boldsymbol{x})) + \boldsymbol{b}^{(L+1)},$$

where L is the depth of the neural network, $\boldsymbol{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ and $\boldsymbol{b} \in \mathbb{R}^{n_l}$ are the parameters, and n_l is the width of the lth layer of the neural network. Given the depth of the network and the width of each layer, we denote the set of NN functions by

$$\mathcal{M}(\theta, L) = \{\mathcal{U}(\boldsymbol{x}) = \boldsymbol{W}^{(L+1)}(N^{(L)} \circ \cdots \circ N^{(1)}(\boldsymbol{x})) + \boldsymbol{b}^{(L+1)} : \boldsymbol{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}, \boldsymbol{b}^{(l)} \in \mathbb{R}^{n_l}, l = 1, \ldots, L+1\},$$

where $\theta = \{(\boldsymbol{W}^{(l)}, \boldsymbol{b}^{(l)})\}_{l=1}^{L+1}$.

Next, let us introduce randomized neural networks. While the structure of randomized neural networks is the same as that of fully connected neural networks, there is a key difference. In fully connected neural networks, all parameters are trained. In randomized neural networks, however, the output-layer parameters are adjustable while the hidden-layer parameters are randomly assigned and fixed. We focus on single-hidden-layer neural networks with the dimension of the output layer being one, that is, $\boldsymbol{W}^{(2)} \in \mathbb{R}^{1 \times n_1}$, and $\boldsymbol{b}^{(2)}$ is set to zero. In the context of randomized neural networks, we define the function space

$$\mathcal{M}_{RNN}(K) = \left\{ \mathcal{U}(\alpha, \theta, \boldsymbol{x}) = \sum_{j=1}^{M} \alpha_j^K \phi^K(\theta_j, \boldsymbol{x}) : \boldsymbol{x} \in K \right\}, \tag{2.1}$$

where $K \subset \Omega$ is the domain, $M = n_1$ is the width of the last hidden layer, θ represents the parameters of the hidden layers, α denotes the parameters of the output layer, and ϕ represents the nonlinear function that produces the output of the last hidden layer. For simplicity, we will use $\phi_j^K(\boldsymbol{x})$ instead of $\phi^K(\theta_j, \boldsymbol{x})$ for the remainder of this paper.

2.2. LRNN-DG method

In [39], the authors demonstrated the success of locELM, which combines the concepts of randomized neural networks and domain decomposition, in solving partial differential equations. This method has proven to be competitive with traditional methods like FEM and has shown strong potential for solving PDEs numerically. However, locELM is based on the strong form of PDEs, which may not be suitable for problems that require weak formulations. The main contribution of this paper is to combine local randomized neural networks with the DG methods to solve PDEs in weak form. In this approach, the output fields of the last hidden layers of the local neural networks are utilized to construct local basis functions for numerical solutions, which are then connected using DG formulation.

Let us introduce the local randomized neural networks with the discontinuous Galerkin formulation. Here, we take the Poisson equation as a model problem,

$$-\Delta u = f \quad \text{in } \Omega, \tag{2.2a}$$

$$u = g \quad \text{on } \partial\Omega, \tag{2.2b}$$

where f is a given source term, $\partial\Omega$ is the boundary of Ω, and g is a function defined on $\partial\Omega$. The weak formulation of the above problem is: Find $u \in H_g^1(\Omega)$ such that

$$a(u, v) = \int_\Omega f\, v \,\mathrm{d}x \quad \forall v \in H_0^1(\Omega). \tag{2.3}$$

Here, $H_g^1(\Omega) = \{v \in H^1(\Omega) : v = g \text{ on } \partial\Omega\}$ and

$$a(u, v) = \int_\Omega \nabla u \cdot \nabla v \,\mathrm{d}x.$$

Like the setting in the DG method, we partition the domain into some subdomains, and approximate the solution on each subdomain by using a local neural network. First, we give some notation. Let $\{\mathcal{T}_h\}$ be the decomposition of $\bar{\Omega}$, where $h = \max_{K \in \mathcal{T}_h}\{\mathrm{diam}(K)\}$. For each $\mathcal{T}_h$, N_e denotes the number of elements in $\mathcal{T}_h$, that is, $|\mathcal{T}_h| = N_e$. Let $\mathcal{E}_h$ be the union of the boundaries of all the elements $K \in \mathcal{T}_h$, $\mathcal{E}_h^i$ is the set of all interior edges, and $\mathcal{E}_h^\partial = \mathcal{E}_h \backslash \mathcal{E}_h^i$. Let K^+ and K^- be two neighboring elements sharing a common edge e. Denote by $\boldsymbol{n}^\pm = \boldsymbol{n}|_{\partial K^\pm}$ the unit outward normal vectors on $\partial K^\pm$. For a scalar function v and a vector-valued function $\boldsymbol{q}$, let $v^\pm = v|_{\partial K^\pm}$ and $\boldsymbol{q}^\pm = \boldsymbol{q}|_{\partial K^\pm}$. We define the averages $\{\cdot\}$ and the jumps $[\![\cdot]\!]$, $[\cdot]$ on $e \in \mathcal{E}_h^i$ by

$$\{v\} = \frac{1}{2}(v^+ + v^-), \quad [\![v]\!] = v^+\boldsymbol{n}^+ + v^-\boldsymbol{n}^-,$$
$$\{\boldsymbol{q}\} = \frac{1}{2}(\boldsymbol{q}^+ + \boldsymbol{q}^-), \quad [\boldsymbol{q}] = \boldsymbol{q}^+ \cdot \boldsymbol{n}^+ + \boldsymbol{q}^- \cdot \boldsymbol{n}^-.$$

If $e \in \mathcal{E}_h^\partial$, we set

$$[\![v]\!] = v\boldsymbol{n}, \quad \{\boldsymbol{q}\} = \boldsymbol{q},$$

where $\boldsymbol{n}$ is the unit outward normal vector on $\partial\Omega$. In the analysis, we need the following identities:

$$\int_K \nabla v \cdot \boldsymbol{q} \,\mathrm{d}x = -\int_K v\,(\nabla \cdot \boldsymbol{q})\,\mathrm{d}x + \int_{\partial K} v\,\boldsymbol{q} \cdot \boldsymbol{n}_K \,\mathrm{d}s, \tag{2.4}$$

$$\sum_{K \in \mathcal{T}_h} \int_{\partial K} v\boldsymbol{q} \cdot \boldsymbol{n}_K \,\mathrm{d}s = \int_{\mathcal{E}_h} [\![v]\!] \cdot \{\boldsymbol{q}\} \,\mathrm{d}s + \int_{\mathcal{E}_h^i} \{v\}[\boldsymbol{q}] \,\mathrm{d}s. \tag{2.5}$$

We introduce the following DG space based on local randomized neural networks associated with the partition $\mathcal{T}_h$:

$$V_h = \{v_h \in L^2(\Omega) : \ v_h|_K \in \mathcal{M}_{RNN}(K) \quad \forall K \in \mathcal{T}_h\},$$

where $\mathcal{M}_{RNN}(K)$ denotes the function space of the randomized neural networks given in (2.1). So for each $v_h \in V_h$, $v_h|_K = \sum_{j=1}^{M} v_j^K \phi_j^K(\boldsymbol{x})$.

We make the following assumption.

Assumption 2.1. For any $K \in \mathcal{T}_h$, assume that the functions $\{\phi_j^K(\boldsymbol{x}) : j = 1, 2, \dots, M\}$ of last hidden layers in subdomain K are linearly independent.

For example, let $\phi_j^K(\boldsymbol{x}) = \sin(\boldsymbol{W}_j \boldsymbol{x} + \boldsymbol{b}_j)$, then $\{\sin(\boldsymbol{W}_j \boldsymbol{x} + \boldsymbol{b}_j), j = 1, 2, \dots, M\}$ is a set of linearly independent functions if proper values of weights $\boldsymbol{W}_j$ and bias $\boldsymbol{b}_j$ are chosen. Of course, this assumption can be satisfied for other activation functions, like $\phi_j^K(\boldsymbol{x}) = \tanh(\boldsymbol{W}_j \boldsymbol{x} + \boldsymbol{b}_j)$.

The local randomized neural networks with DG (LRNN-DG) method for solving the Poisson problem is: Find $u_h \in V_h$ such that

$$a_h(u_h, v_h) = l(v_h) \quad \forall v_h \in V_h, \tag{2.6}$$

where

$$a_h(u, v) = \int_\Omega \nabla_h u \cdot \nabla_h v \mathrm{d}x - \int_{\mathcal{E}_h} \{\nabla_h u\} \cdot [\![v]\!] \mathrm{d}s - \int_{\mathcal{E}_h} [\![u]\!] \cdot \{\nabla_h v\} \mathrm{d}s + \int_{\mathcal{E}_h} \eta [\![u]\!] \cdot [\![v]\!] \mathrm{d}s, \tag{2.7}$$

$$l(v_h) = \int_\Omega f v_h \mathrm{d}x - \int_{\mathcal{E}_h^\partial} g \boldsymbol{n} \cdot \nabla_h v_h \mathrm{d}s + \int_{\mathcal{E}_h^\partial} \eta g v_h \mathrm{d}s. \tag{2.8}$$

Here, $\nabla_h v_h$ denotes the broken gradient of v_h with respect to the decomposition $\mathcal{T}_h$, i.e., $\nabla_h v_h = \nabla v_h|_K$, and $\int_{\mathcal{E}_h} \eta [\![u]\!] \cdot [\![v]\!] \mathrm{d}s$ is the penalty term, where the function η equals a constant $\eta_e (h_e)^{-1}$ on each $e \in \mathcal{E}_h$, with η_e being a positive number. In this paper, we focus on the interior penalty DG (IPDG) scheme, although other DG schemes studied in [41] can also be considered, provided that the bilinear form (2.7) and the linear form (2.8) are modified accordingly.

The outputs of the last hidden layer, i.e., $\{\phi_j^K(\boldsymbol{x}) : K \in \mathcal{T}_h, \ j = 1, 2, \dots, M\}$, can be regarded as the local basis functions of the LRNN-DG. We can obtain the global stiffness matrices $\mathbb{A}$ and the right-hand side L using these local basis functions. It should be noted that the penalty parameter in the IPDG scheme (2.6) needs to be chosen appropriately for the given problem. From (2.6), we obtain the system of equations,

$$\mathbb{A} U = L, \tag{2.9}$$

where $\mathbb{A}$ is a $N_e M \times N_e M$ matrix, L is a $N_e M \times 1$ vector, and there are $N_e M$ unknown variables $U = \{u_j^K : K \in \mathcal{T}_h, j = 1, 2, \dots, M\}$.

2.3. Some properties of the LRNN-DG method

The following lemma shows the consistency of the DG scheme, a similar argument can be found in [41] and other references on DG methods. For completeness, we give brief proof as well.

Lemma 2.2 (*Consistency*). *The LRNN-DG scheme is consistent, i.e., for the solution $u \in H^2(\Omega)$ of problem (2.3), we have*

$$a_h(u, v_h) = l(v_h) \quad \forall v_h \in V_h. \tag{2.10}$$

Proof. We know that $u \in H^2(\Omega)$ implies $[\![u]\!] = 0$, $[\nabla u] = 0$ on $\mathcal{E}_h^i$ and $u = g$ on $\mathcal{E}_h^\partial$. Then, by the identities (2.4), (2.2) and (2.5), we have

$$\begin{aligned} a_h(u, v_h) &= \int_\Omega \nabla u \cdot \nabla_h v_h \mathrm{d}x - \int_{\mathcal{E}_h} \{\nabla_h u\} \cdot [\![v_h]\!] \mathrm{d}s - \int_{\mathcal{E}_h^\partial} g \boldsymbol{n} \cdot \nabla_h v_h \mathrm{d}s + \int_{\mathcal{E}_h^\partial} \eta g v_h \mathrm{d}s \\ &= -\int_\Omega \Delta u v_h \mathrm{d}x + \sum_{K \in \mathcal{T}_h} \int_K \nabla u \cdot \boldsymbol{n}_K v_h \mathrm{d}s - \int_{\mathcal{E}_h} \{\nabla_h u\} \cdot [\![v_h]\!] \mathrm{d}s - \int_{\mathcal{E}_h^\partial} g \boldsymbol{n} \cdot \nabla_h v_h \mathrm{d}s + \int_{\mathcal{E}_h^\partial} \eta g v_h \mathrm{d}s \quad \square \\ &= l(v_h). \end{aligned}$$

From the LRNN-DG scheme (2.6) and Lemma 2.2, we have

$$a_h(u - u_h, v_h) = 0 \quad \forall v_h \in V_h. \tag{2.11}$$

Next, let $V(h) = V_h + H^2(\Omega)$, then we define some seminorms and norms by the following relations:

$$\begin{aligned} |v|_{1,h}^2 &= \sum_{K \in \mathcal{T}_h} |v|_{1,K}^2, \quad |v|_{1,*}^2 = \sum_{e \in \mathcal{E}_h} h_e^{-1} \|[\![v]\!]\|_{0,e}^2, \\ \|v\|_w^2 &= |v|_{1,h}^2 + |v|_{1,*}^2, \quad \|v\|_*^2 = \|v\|_w^2 + \sum_{K \in \mathcal{T}_h} h_K^2 |v|_{2,K}^2. \end{aligned} \tag{2.12}$$

The norms $\|v\|_w^2$ and $\|v\|_*^2$ are well-defined because

$$\|v\|_0 \le C\|v\|_w \le C\|v\|_* \quad \forall v \in V(h). \tag{2.13}$$

Here, and in the rest of the paper, C denotes a constant that is independent of h and M.

Then we have the boundedness and stability of the bilinear form a_h by standard argument (see [41] and the references therein).

Lemma 2.3 (*Boundedness*). *$a_h(u,v)$ satisfies*

$$a_h(u,v) \le C_b\|u\|_*\|v\|_* \quad \forall\, v \in V(h). \tag{2.14}$$

Lemma 2.4 (*Stability*). *Set $\eta_0 = \inf_e \eta_e > 0$, if η_0 is large enough, then a_h satisfies*

$$a_h(v,v) \ge C_s\|v\|_*^2 \quad \forall\, v \in V_h. \tag{2.15}$$

By Assumption 2.1, Lemmas 2.3 and 2.4, we know that problem (2.6) is well-posed and $\mathbb{A}$ is symmetric positive definite (SPD). Therefore, many solvers for the SPD system can be used to solve (2.9). The randomized neural network has a certain possibility that the functions $\{\phi_j^K(\boldsymbol{x}) : j = 1,2,\dots,M\}$ are not linearly independent, which means that $\mathbb{A}$ is singular, and we need to solve the linear system (2.9) by the least-squares approach. Then, the parameters U in the neural networks' output layers can be obtained by a least-squares method.

2.4. LRNN-C^0DG method

The LRNN-DG method presented in the previous subsection is based on the IPDG scheme. It is known that the performance of the IPDG method depends on the choice of the penalty parameter η. It can be cumbersome to determine an appropriate value for the penalty parameter. Of course, we can use other DG formulations, such as local DG, to avoid the difficulty of choosing a proper penalty parameter. However, by taking advantage of the least squares method, we can enforce the C^0-continuous condition on each $e \in \mathcal{E}_h^i$ and the Dirichlet boundary condition on $\mathcal{E}_h^\partial$ to overcome this issue.

We add additional equations to enforce the solution to satisfy the boundary condition (2.2b), that is, we choose some collocation points on the boundary edge, $P_h^g = \{\boldsymbol{x}_j^e \in e : e \in \mathcal{E}_h^\partial, j = 1,2,\dots,N_g\}$ and $|P_h^g| = N_g$, such that

$$u_h(\boldsymbol{x}_j^e) = g(\boldsymbol{x}_j^e) \quad \forall \boldsymbol{x}_j^e \in P_h^g. \tag{2.16}$$

In addition, we also need to make sure that the numerical solution u_h satisfies certain C^0-continuity conditions across the interior edges $e \in \mathcal{E}_h^i$. We choose some collocation points on the interior edges, $P_h^i = \{\boldsymbol{x}_j^e \in e : e \in \mathcal{E}_h^i, j = 1,2,\dots,N_{in}\}$ and $|P_h^i| = N_{in}$, on these points, we set

$$[\![u_h(\boldsymbol{x}_j^e)]\!] = 0 \quad \forall \boldsymbol{x}_j^e \in P_h^i. \tag{2.17}$$

Then we obtain a system of equations with respect to (2.16) and (2.17),

$$\mathbb{A}_2 U = L_2, \tag{2.18}$$

where $\mathbb{A}_2$ is a $(N_g + N_{in}) \times N_e M$ matrix, U is the $N_e M \times 1$ unknown vector, and L_2 is a $(N_g + N_{in}) \times 1$ vector.

Condition (2.17) makes $[\![u_h]\!] \approx 0$, so the LRNN-C^0DG scheme is to find $u_h \in V_h$ such that

$$\begin{aligned} a_h^0(u_h, v_h) &= l^0(v_h) && \forall v_h \in V_h, \\ [\![u_h(\boldsymbol{x}_j^e)]\!] &= 0 && \forall \boldsymbol{x}_j^e \in P_h^i, \\ u_h(\boldsymbol{x}_j^e) &= g(\boldsymbol{x}_j^e) && \forall \boldsymbol{x}_j^e \in P_h^g, \end{aligned} \tag{2.19}$$

where

$$a_h^0(u_h, v_h) = \int_\Omega \nabla_h u_h \cdot \nabla_h v_h \mathrm{d}x - \int_{\mathcal{E}_h} \{\nabla_h u_h\} \cdot [\![v_h]\!] \mathrm{d}s - \int_{\mathcal{E}_h} \{\nabla_h v_h\} \cdot [\![u_h]\!] \mathrm{d}s, \tag{2.20}$$

$$l^0(v_h) = \int_\Omega f v_h \mathrm{d}x - \int_{\mathcal{E}_h^\partial} g\boldsymbol{n} \cdot \nabla_h v_h \mathrm{d}s. \tag{2.21}$$

Here, we keep the term $\int_{\mathcal{E}_h} \{\nabla_h v_h\} \cdot [\![u_h]\!] \mathrm{d}s$ for the symmetry of the bilinear form a_h^0. Note that this scheme is free of penalty parameters. Finally, from (2.19), we get a linear system,

$$\begin{bmatrix} \mathbb{A}_1 \\ \mathbb{A}_2 \end{bmatrix} U = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix}, \tag{2.22}$$

where $\mathbb{A}_1$ is a $N_e M \times N_e M$ matrix, L_1 is a $N_e M \times 1$ vector. We look for the least-squares solution for this linear system.

Remark 2.5. From numerical examples, we see that the scheme (2.19) has a good performance. But note that the following scheme which destroys the symmetry still works well.

$$\widetilde{a_h^0}(u_h, v_h) = \int_\Omega f v_h \mathrm{d}x \quad \forall\, v_h \in V_h, \tag{2.23}$$

where

$$\widetilde{a_h^0}(u_h, v_h) = \int_\Omega \nabla_h u_h \cdot \nabla_h v_h \mathrm{d}x - \int_{\mathcal{E}_h} \{\nabla_h u_h\} \cdot [\![v_h]\!] \mathrm{d}s. \tag{2.24}$$

2.5. LRNN-C^1DG method

In the previous subsection, we simplified the DG scheme by enforcing the continuity of u_h on internal edges, i.e., by setting $[\![u_h]\!] = 0$ on $e \in \mathcal{E}_h^i$. Can we take this one step further? Let us introduce the LRNN-C^1DG method in this subsection.

In each subdomain K, $-\Delta u = f$, so we have the FEM formulation:

$$\int_K \nabla u \cdot \nabla v \mathrm{d}x - \int_{\partial K} \nabla u \cdot \boldsymbol{n}_K v \mathrm{d}s = \int_K f v \mathrm{d}x \quad \forall K \in \mathcal{T}_h, \tag{2.25}$$

where $\boldsymbol{n}_K$ is the unit outer normal vector on ∂K. However, (2.25) with Dirichlet boundary condition (2.2b) is not equivalent to the Poisson problem because the local problems lack connections with each other. From the domain decomposition method [42], we know that we need the continuity of u and flux, i.e., we require $[\![u_h]\!] = 0$ and $[\nabla u_h] = 0$ on each $e \in \mathcal{E}_h^i$.

We need to ensure that local representations of the solution satisfy C^1-continuity conditions across the subdomain boundaries due to consistency. We select some points on the internal edges P_h^i as described in Section 2.4. At these points, using the same set-up as the LRNN-C^0DG method, we obtain the LRNN-C^1DG method: find $u_h \in V_h$ such that

$$\begin{aligned} a_h^K(u_h, v_h) &= \int_K f v_h \mathrm{d}x \qquad \forall v_h \in V_h \quad \forall\ K \in \mathcal{T}_h, \\ [\![u_h(\boldsymbol{x}_j^e)]\!] &= 0 \qquad\qquad\qquad\quad \forall\, \boldsymbol{x}_j^e \in P_h^i, \\ [\nabla u_h(\boldsymbol{x}_j^e)] &= 0 \qquad\qquad\qquad\quad \forall\ \boldsymbol{x}_j^e \in P_h^i, \\ u_h(\boldsymbol{x}_j^e) &= g(\boldsymbol{x}_j^e) \qquad\qquad\qquad\quad \forall\, \boldsymbol{x}_j^e \in P_h^g, \end{aligned} \tag{2.26}$$

where

$$a_h^K(u_h, v_h) = \int_K \nabla_h u_h \cdot \nabla_h v_h \mathrm{d}x - \int_{\partial K} \nabla_h u_h \cdot \boldsymbol{n}_K v_h \mathrm{d}s. \tag{2.27}$$

Finally, we obtain the following linear system,

$$\begin{bmatrix} \mathbb{A}_1 \\ \mathbb{A}_2 \end{bmatrix} U = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix},$$

where $\mathbb{A}_1$ is a $N_e M \times N_e M$ matrix, L_1 is a $N_e M \times 1$ vector. $\mathbb{A}_2$ is a $(2N_{in} + N_g) \times N_e M$ matrix, U is a $N_e M \times 1$ vector of unknown variables, L_2 is a $(2N_{in} + N_g) \times 1$ vector. We look for the least-squares solution to this system. After the weights of the output layer in each local neural network are obtained by the linear least-squares computation, we can get all the values of the problem (2.2) in the domain Ω.

3. Convergence of the LRNN with DG methods

3.1. Convergence of the LRNN-DG method

We now turn to the error analysis of the LRNN-DG method. In [21], the authors prove that a randomized neural network with one hidden layer can approximate any continuous function on a compact domain, as long as the hidden layer size is large enough. According to [38], if the exact solution is a smooth function, the generalization capability of neural networks is not degraded by ELM with proper activation functions and random initialization strategies. In addition, [43,44] have shown that neural networks can approximate the solution well with appropriate depth and width. Based on these findings, we make the following assumption: let $u_\sigma \in V_h$ be a suitable approximation of the exact solution u.

Assumption 3.1. Given a decomposition $\mathcal{T}_h$ with $|\mathcal{T}_h| = N_e$ and V_h is the associated DG space of LRNN. For any small positive number ϵ, there exists a positive integer M_ϵ such that if $M > M_\epsilon$, we have a function $u_\sigma \in V_h$ satisfying

$$\|u - u_\sigma\|_{0,K} \le C h_K N_e^{-1/2} \epsilon, \quad |u - u_\sigma|_{1,K} \le C N_e^{-1/2} \epsilon, \quad |u - u_\sigma|_{2,K} \le C h_K^{-1} N_e^{-1/2} \epsilon.$$

Here, M is the number of the basis of $\mathcal{M}_{RNN}(K)$, and C denotes a constant number that is independent of h and M.

Remark 3.2. For any function $u \in H^{p+1}(K)$, we know that there exists a polynomial function $u_I \in P_p(K)$ such that

$$\begin{aligned} &\|u - u_I\|_{0,K} \le C h_K^{p+1} |u|_{p+1,K}, \quad |u - u_I|_{1,K} \le C h_K^{p} |u|_{p+1,K}, \\ &|u - u_I|_{2,K} \le C h_K^{p-1} |u|_{p+1,K}. \end{aligned}$$

Similarly, we make Assumption 3.1 in light of good approximation properties of neural networks.

From the above assumption and the trace inequality, we have

$$\begin{aligned}\|u-u_\sigma\|_*^2 &= \sum_{K\in\mathcal{T}_h}|u-u_\sigma|_{1,K}^2+\sum_{K\in\mathcal{T}_h}h_K^2|u-u_\sigma|_{2,K}^2+\sum_{e\in\mathcal{E}_h}h_e^{-1}\|[\![u-u_\sigma]\!]\|_{0,e}^2\\ &\le C\left(\sum_{K\in\mathcal{T}_h}|u-u_\sigma|_{1,K}^2+\sum_{K\in\mathcal{T}_h}h_K^2|u-u_\sigma|_{2,K}^2+\sum_{K\in\mathcal{T}_h}h_K^{-2}\|u-u_\sigma\|_{0,K}^2\right)\\ &\le C\epsilon. \end{aligned} \tag{3.1}$$

For the LRNN-DG method, we have the following Ceá-type inequality.

Theorem 3.3. *Let u and u_h be solutions of the problem (2.3) and the LRNN-DG scheme (2.6), we obtain*

$$\|u-u_h\|_* \le (1+C_b/C_s)\inf_{v_h\in V_h}\|u-v_h\|_*. \tag{3.2}$$

Proof. For any $v_h\in V_h$, by the boundedness (2.14) and stability (2.15) of the bilinear form a_h, as well as (2.11), we have

$$\begin{aligned}C_s\|v_h-u_h\|_*^2 &\le a_h(v_h-u_h,v_h-u_h)\\ &= a_h(v_h-u,v_h-u_h)+a_h(u-u_h,v_h-u_h)\\ &\le C_b\|v_h-u\|_*\|v_h-u_h\|_*,\end{aligned}$$

then we get

$$\|v_h-u_h\|_* \le C_b/C_s\|u-v_h\|_*. \tag{3.3}$$

Finally, by triangle inequality, we obtain

$$\|u-u_h\|_* \le \|u-v_h\|_*+\|v_h-u_h\|_* \le (1+C_b/C_s)\|u-v_h\|_*, \tag{3.4}$$

which completes the proof of the theorem. ■

From the Ceá-type inequality and (3.1), let $v_h=u_\rho$ in (3.2), we can obtain the convergence of the LRNN-DG scheme (2.6).

Corollary 3.4. *Let u and u_h be solutions of the problems (2.3) and (2.6), respectively. If Assumption 3.1 holds, then for any small positive number ϵ, there exists a positive integer M_ϵ such that if $M>M_\epsilon$, then*

$$\|u-u_h\|_* \le C\epsilon. \tag{3.5}$$

3.2. Convergence of the LRNN-C^0DG method

In this subsection, we denote the solution of the LRNN-C^0DG scheme (2.19) as $\widetilde{u_h}$. By enforcing the conditions (2.16) and (2.17), we can ensure that $\widetilde{u_h}-g\approx 0$ on boundary edges and $[\![\widetilde{u_h}]\!]\approx 0$ on interior edges. In particular, increasing the number of points $\boldsymbol{x}_j^e$ on each edge e results in smaller values of $\widetilde{u_h}-g$ and $[\![\widetilde{u_h}]\!]$. Therefore, we make the following assumption.

Assumption 3.5. Given a decomposition $\mathcal{T}_h$ with $|\mathcal{T}_h|=N_e$ and V_h is the associated DG space of LRNN. For any small positive number ϵ, on every edge $e\in\mathcal{E}_h$, there exist N_ϵ^e such that if $N^e>N_\epsilon^e$, then

$$\|[\![\widetilde{u_h}]\!]\|_{0,e}\le Ch_e^{1/2}\epsilon \quad\text{and}\quad \|\widetilde{u_h}-g\|_{0,e}\le Ch_e^{1/2}\epsilon.$$

Here, N^e is the number of points $\boldsymbol{x}_j^e$ on the edge e and C denotes a constant number that is independent of h and M.

Next, we prove the convergence of the LRNN-C^0DG scheme (2.19).

Theorem 3.6. *Let u and $\widetilde{u_h}$ be solutions of the problem (2.3) and the LRNN-C^0DG scheme (2.19), respectively. If Assumption 3.1 and Assumption 3.5 hold, for any small positive number ϵ, there exist positive integers M_ϵ, N_ϵ^e such that if $M>M_\epsilon$, $N^e>N_\epsilon^e$, then*

$$\|u-\widetilde{u_h}\|_* \le C\epsilon. \tag{3.6}$$

Proof. From the LRNN-C^0DG scheme (2.19) and the LRNN-DG scheme (2.6), we know that

$$\begin{aligned}a_h^0(\widetilde{u_h},v_h) &= l^0(v_h)\quad \forall\, v_h\in V_h,\\ a_h(u_h,v_h) &= l(v_h)\quad \forall\, v_h\in V_h,\end{aligned} \tag{3.7}$$

so

$$a_h(\widetilde{u_h},v_h)-\int_{\mathcal{E}_h}\eta[\![\widetilde{u_h}]\!]\cdot[\![v_h]\!]\mathrm{d}s = l(v_h)-\int_{\mathcal{E}_h^\partial}\eta g v_h\mathrm{d}s. \tag{3.8}$$

And by the consistency (2.10), we have

$$a_h(\widetilde{u_h}-u,v_h)=\int_{\mathcal{E}_h}\eta[\![\widetilde{u_h}]\!]\cdot[\![v_h]\!]\mathrm{d}s-\int_{\mathcal{E}_h^\partial}\eta g v_h\mathrm{d}s. \tag{3.9}$$

From the stability and boundedness of a_h and (2.11), we get

$$\begin{aligned}C_s\|\widetilde{u_h}-u_h\|_*^2&\le a_h(\widetilde{u_h}-u_h,\widetilde{u_h}-u_h)\\&=a_h(\widetilde{u_h}-u,\widetilde{u_h}-u_h)+a_h(u-u_h,\widetilde{u_h}-u_h)\\&=\int_{\mathcal{E}_h}\eta[\![\widetilde{u_h}]\!]\cdot[\![\widetilde{u_h}-u_h]\!]\mathrm{d}s-\int_{\mathcal{E}_h^\partial}\eta g(\widetilde{u_h}-u_h)\mathrm{d}s\\&\le\eta_{\max}\left(\sum_{e\in\mathcal{E}_h^i}h_e^{-1}\|[\![\widetilde{u_h}]\!]\|_{0,e}^2+\sum_{e\in\mathcal{E}_h^\partial}h_e^{-1}\|\widetilde{u_h}-g\|_{0,e}^2\right)^{\frac12}|\widetilde{u_h}-u_h|_{1,*}.\end{aligned}$$

Therefore, by Assumption 3.5, we have

$$\|\widetilde{u_h}-u_h\|_*\le\frac{\eta_{\max}}{C_s}\left(\sum_{e\in\mathcal{E}_h^i}h_e^{-1}\|[\![\widetilde{u_h}]\!]\|_{0,e}^2+\sum_{e\in\mathcal{E}_h^\partial}h_e^{-1}\|\widetilde{u_h}-g\|_{0,e}^2\right)^{\frac12}\le C\epsilon.$$

Finally, by triangle inequality and Corollary 3.4, we obtain

$$\|u-\widetilde{u_h}\|_*\le\|u-u_h\|_*+\|u_h-\widetilde{u_h}\|_*\le C\epsilon.\quad\square$$

3.3. Convergence of the LRNN-C^1DG method

In this subsection, we denote the solution of the LRNN-C^1DG scheme (2.26) by $\overline{u_h}$. Similar to the LRNN-C^0DG method, by enforcing the condition $[\nabla\overline{u_h}(\boldsymbol{x}_j^e)]=0$ for each point $\boldsymbol{x}_j^e$ on e, we can ensure that $[\nabla\overline{u_h}(\boldsymbol{x}_j^e)]\approx0$. Therefore, we make the following assumption.

Assumption 3.7. Given a decomposition $\mathcal{T}_h$ with $|\mathcal{T}_h|=N_e$ and V_h is the associated DG space of LRNN. For any small positive number ϵ, on every edge $e\in\mathcal{E}_h$, there exists N_ϵ^e such that if $N^e>N_\epsilon^e$, then

$$\|[\![\overline{u_h}]\!]\|_{0,e}\le Ch_e^{1/2}\epsilon,\quad\|\overline{u_h}-g\|_{0,e}\le Ch_e^{1/2}\epsilon\quad\text{and}\quad\|[\nabla\overline{u_h}(\boldsymbol{x}_j^e)]\|_{0,e}\le Ch_e^{-1/2}\epsilon.$$

Here, N^e is the number of the points $\boldsymbol{x}_j^e$ on edge e and C denotes a constant number that is independent of h and M.

Remark 3.8. To demonstrate the validity of Assumption 3.5 and Assumption 3.7, in Section 5, we provide numerical evidence which shows that the quantities $\|[\![u_h]\!]\|_{0,e}$, $\|[\nabla u_h]\|_{0,e}$ on interior edges and $\|u_h-g\|_{0,e}$ on boundary edges decrease as the number of collection points increases.

Finally, we show the convergence of the LRNN-C^1DG scheme.

Theorem 3.9. *Let u and $\overline{u_h}$ be solutions of the problem (2.3) and the LRNN-C^1DG scheme (2.26), respectively. If Assumption 3.1 and Assumption 3.7 hold, for any small positive number ϵ, there exist positive integers M_ϵ, N_ϵ^e such that if $M>M_\epsilon$, $N^e>N_\epsilon^e$, then*

$$\|u-\overline{u_h}\|_*\le C\epsilon. \tag{3.10}$$

Proof. We know that in each subdomain,

$$\int_K\nabla_h\overline{u_h}\cdot\nabla_h v_h\mathrm{d}x-\int_{\partial K}\nabla_h\overline{u_h}\cdot\boldsymbol{n}_K v_h\mathrm{d}s=\int_K f v_h\mathrm{d}x\quad\forall K\in\mathcal{T}_h. \tag{3.11}$$

Then we add all the elements to obtain

$$a_h^1(\overline{u_h},v_h)=\int_\Omega f v_h\mathrm{d}x\quad\forall v_h\in V_h, \tag{3.12}$$

where

$$\begin{aligned}a_h^1(\overline{u_h},v_h)&=\int_\Omega\nabla_h\overline{u_h}\cdot\nabla_h v_h\mathrm{d}x-\sum_{K\in\mathcal{T}_h}\int_{\partial K}\nabla_h\overline{u_h}\cdot\boldsymbol{n}_K v_h\mathrm{d}s\\&=\int_\Omega\nabla_h\overline{u_h}\cdot\nabla_h v_h\mathrm{d}x-\int_{\mathcal{E}_h}\{\nabla_h\overline{u_h}\}\cdot[\![v_h]\!]\mathrm{d}s-\int_{\mathcal{E}_h^i}\{v_h\}\cdot[\nabla_h\overline{u_h}]\mathrm{d}s.\end{aligned} \tag{3.13}$$

So

$$\begin{aligned} &a_h(\overline{u_h}, v_h) - \int_{\mathcal{E}_h^i} \{v_h\} \cdot [\nabla_h \overline{u_h}] \mathrm{d}s + \int_{\mathcal{E}_h} [\![\overline{u_h}]\!] \cdot \{\nabla_h v_h\} \mathrm{d}s - \int_{\mathcal{E}_h} \eta [\![\overline{u_h}]\!] \cdot [\![v_h]\!] \mathrm{d}s \\ =&l(v_h) + \int_{\mathcal{E}_h^\partial} g\boldsymbol{n} \cdot \nabla_h v_h \mathrm{d}s - \int_{\mathcal{E}_h^\partial} \eta g v_h \mathrm{d}s. \end{aligned} \tag{3.14}$$

And we know $a_h(u, v_h) = l(v_h)$, so we have

$$\begin{aligned} a_h(\overline{u_h} - u, v_h) = &\int_{\mathcal{E}_h^i} \{v_h\} \cdot [\nabla_h \overline{u_h}] \mathrm{d}s - \int_{\mathcal{E}_h} [\![\overline{u_h}]\!] \cdot \{\nabla_h v_h\} \mathrm{d}s \\ &+ \int_{\mathcal{E}_h} \eta [\![\overline{u_h}]\!] \cdot [\![v_h]\!] \mathrm{d}s + \int_{\mathcal{E}_h^\partial} g\boldsymbol{n} \cdot \nabla_h v_h \mathrm{d}s - \int_{\mathcal{E}_h^\partial} \eta g v_h \mathrm{d}s. \end{aligned} \tag{3.15}$$

By the stability and boundedness of a_h and (2.11), we get

$$\begin{aligned} &C_s \|\overline{u_h} - u_h\|_*^2 \le a_h(\overline{u_h} - u_h, \overline{u_h} - u_h) = a_h(\overline{u_h} - u, \overline{u_h} - u_h) + a_h(u - u_h, \overline{u_h} - u_h) \\ =&\int_{\mathcal{E}_h^i} \{\overline{u_h} - u_h\} \cdot [\nabla_h \overline{u_h}] \mathrm{d}s - \int_{\mathcal{E}_h} [\![\overline{u_h}]\!] \cdot \{\nabla_h (\overline{u_h} - u_h)\} \mathrm{d}s + \int_{\mathcal{E}_h} \eta [\![\overline{u_h}]\!] \cdot [\![\overline{u_h} - u_h]\!] \mathrm{d}s \\ &+ \int_{\mathcal{E}_h^\partial} g\boldsymbol{n} \cdot \nabla_h (\overline{u_h} - u_h) \mathrm{d}s - \int_{\mathcal{E}_h^\partial} \eta g (\overline{u_h} - u_h) \mathrm{d}s \\ \le &C \left(\sum_{e \in \mathcal{E}_h^i} h_e^{-1} \|[\![\overline{u_h}]\!]\|_{0,e}^2 + \sum_{e \in \mathcal{E}_h^i} h_e \|[\nabla_h \overline{u_h}]\|_{0,e}^2 + \sum_{e \in \mathcal{E}_h^\partial} h_e^{-1} \|\overline{u_h} - g\|_{0,e}^2 \right)^{\frac{1}{2}} \|\overline{u_h} - u_h\|_*, \end{aligned}$$

so

$$\begin{aligned} \|\overline{u_h} - u_h\|_* &\le \frac{C}{C_s} \left(\sum_{e \in \mathcal{E}_h^i} h_e^{-1} \|[\![\overline{u_h}]\!]\|_{0,e}^2 + \sum_{e \in \mathcal{E}_h^i} h_e \|[\nabla_h \overline{u_h}]\|_{0,e}^2 + \sum_{e \in \mathcal{E}_h^\partial} h_e^{-1} \|\overline{u_h} - g\|_{0,e}^2 \right)^{\frac{1}{2}} \\ &\le C\epsilon. \end{aligned} \tag{3.16}$$

Finally,

$$\|u - \overline{u_h}\|_* \le \|u - u_h\|_* + \|u_h - \overline{u_h}\|_* \le C\epsilon. \quad \blacksquare \tag{3.17}$$

4. Space–time LRNN with DG methods for heat equation

In this section, we investigate the use of local randomized neural networks with DG methods for solving a typical time-dependent PDE, namely the heat equation. Unlike traditional methods that solve the problem by iterating over time steps, we adopt a space–time approach where temporal and spatial variables are treated equally and simultaneously. This approach allows us to avoid the accumulation of errors.

Consider

$$u_t(t, \boldsymbol{x}) - \lambda \Delta u(t, \boldsymbol{x}) = f(t, \boldsymbol{x}) \quad \text{in } \Sigma, \tag{4.1a}$$

$$u(0, \boldsymbol{x}) = u_0(\boldsymbol{x}) \quad \text{in } \Omega, \tag{4.1b}$$

$$u(t, \boldsymbol{x}) = g(t, \boldsymbol{x}) \quad \text{on } I \times \partial\Omega, \tag{4.1c}$$

where $\Omega \subset \mathbb{R}^d$ is a bounded space domain, $I = (0,\ T)$ is the time interval, $\Sigma = I \times \Omega$ is the space–time domain, u is the unknown solution to be solved, f is the given source term, u_0 is the initial condition and g is a function defined on $I \times \partial\Omega$. For convenience, we set the coefficient $\lambda = 1$.

Here we present the notation of the space–time approach. We partition the space–time domain Σ into some subdomains and approximate the solution on each subdomain by using a local neural network. First, we give the decomposition of the time interval $\mathcal{D}_\tau = \{I_i = (t_{i-1}, t_i), 0 = t_0 < t_1 < \cdots < t_{N_t} = T\}$, where $\tau = \max_{I_i \in \mathcal{D}_\tau} \{\text{length}(I_i)\}$ and N_t denotes the number of subintervals along the temporal direction. Let $\mathcal{P}_\tau = \{t_i, i = 0, \ldots, N_t\}$ be the union of the boundary points of all the intervals $I_i = (t_{i-1}, t_i) \in \mathcal{D}_\tau$, and $\mathcal{P}_\tau^i = \mathcal{P}_\tau \backslash \{t_0, t_{N_t}\}$ be the set of all interior points. Let $\{\mathcal{T}_h\}$ be the decomposition of $\bar{\Omega}$, where $h = \max_{K \in \mathcal{T}_h} \{\text{diam}(K)\}$. $\mathcal{E}_h$, $\mathcal{E}_h^i$ and $\mathcal{E}_h^\partial$ have the same definitions stated in Section 2.2. Let $\{\mathcal{D}_\tau \times \mathcal{T}_h\}$ denote the decomposition of the space–time domain $\bar{\Sigma}$. For $\mathcal{T}_h$, N_s denotes the number of elements in $\mathcal{T}_h$, that is, $N_e = |\mathcal{D}_\tau \times \mathcal{T}_h| = N_t N_s$. Let σ_h^+ and σ_h^- be two neighboring elements sharing a common spatial face f. Denote by $\boldsymbol{n}^\pm = \boldsymbol{n}|_{\partial K^\pm}$ the unit outward normal vectors on $\partial K^\pm$. For a scalar-valued function v and a vector-valued function $\boldsymbol{q}$, let $v^\pm = v|_{\partial\sigma^\pm}$ and $\boldsymbol{q}^\pm = \boldsymbol{q}|_{\partial\sigma^\pm}$. We define the averages $\{\cdot\}$ and the jumps $[\![\cdot]\!]$, $[\cdot]$ on $f \in (\mathcal{D}_\tau \times \mathcal{E}_h^i)$ by

$$\{v\} = \frac{1}{2}(v^+ + v^-), \quad [\![v]\!] = v^+ \boldsymbol{n}^+ + v^- \boldsymbol{n}^-,$$

$$\{\boldsymbol{q}\} = \frac{1}{2}(\boldsymbol{q}^+ + \boldsymbol{q}^-), \quad [\boldsymbol{q}] = \boldsymbol{q}^+ \cdot \boldsymbol{n}^+ + \boldsymbol{q}^- \cdot \boldsymbol{n}^-.$$

If $f \in (\mathcal{D}_\tau \times \mathcal{E}_h^\partial)$, we set

$$[\![v]\!] = v\boldsymbol{n}, \quad \{\boldsymbol{q}\} = \boldsymbol{q},$$

where $\boldsymbol{n}$ is the unit outward normal vector on $\partial\Omega$.

Moreover, if σ_τ^+ and σ_τ^- are two neighboring elements sharing a common temporal face f, we have $w(t_i^\pm, \boldsymbol{x}) = w(t_i, \boldsymbol{x})|_{\partial\sigma_\tau^\pm}$ for a scalar function w. The average $\{\cdot\}$ and the jump $[\cdot]$ on $f_\tau \in (\mathcal{P}_\tau^i \times \mathcal{T}_h)$ are defined by

$$\{w(t_i, \boldsymbol{x})\} = \frac{1}{2}\left(w(t_i^+, \boldsymbol{x}) + w(t_i^-, \boldsymbol{x})\right), \quad [w(t_i, \boldsymbol{x})] = w(t_i^-, \boldsymbol{x}) - w(t_i^+, \boldsymbol{x}).$$

If $f_\tau \in (\mathcal{P}_\tau^\partial \times \mathcal{T}_h)$, we set

$$[w(t_0, \boldsymbol{x})] = -w(t_0, \boldsymbol{x}), \quad [w(t_{N_t}, \boldsymbol{x})] = w(t_{N_t}, \boldsymbol{x}), \quad \{w(t, \boldsymbol{x})\} = w(t, \boldsymbol{x}).$$

4.1. Space–time LRNN-DG method

We introduce the following DG space based on the local randomized neural network associated with the partition $\mathcal{D}_\tau \times \mathcal{T}_h$:

$$V_h^\tau = \{v_h^\tau \in L^2(\Sigma) : \ v_h^\tau|_{I_i \times K} \in \mathcal{M}_{RNN}(I_i \times K) \quad \forall\, I_i \in \mathcal{D}_\tau \ \forall\, K \in \mathcal{T}_h\},$$
$$\boldsymbol{Q}_h^\tau = \{\boldsymbol{q}_h^\tau \in [L^2(\Sigma)]^d : \ \boldsymbol{q}_h^\tau|_{I_i \times K} \in [\mathcal{M}_{RNN}(I_i \times K)]^d \quad \forall\, I_i \in \mathcal{D}_\tau \ \forall\, K \in \mathcal{T}_h\}.$$

We rewrite the heat equation as the first-order system,

$$\boldsymbol{p} = \nabla u \quad \text{in } \Sigma,$$
$$\frac{\partial u}{\partial t} - \nabla \cdot \boldsymbol{p} = f \quad \text{in } \Sigma.$$

In the above equations, multiply the test functions $\boldsymbol{q}$ and v respectively on subdomain $\sigma = I_i \times K$, then we get by integration by parts,

$$\int_\sigma \boldsymbol{p} \cdot \boldsymbol{q}\mathrm{d}x\mathrm{d}t = -\int_\sigma u\nabla \cdot \boldsymbol{q}\mathrm{d}x\mathrm{d}t + \int_{I_i \times \partial K} u\boldsymbol{q} \cdot \boldsymbol{n}\mathrm{d}s\mathrm{d}t,$$
$$-\int_\sigma u\frac{\partial v}{\partial t}\mathrm{d}x\mathrm{d}t + \int_\sigma \boldsymbol{p} \cdot \nabla v\mathrm{d}x\mathrm{d}t + \int_K (uv)|_{t_{i-1}}^{t_i}\mathrm{d}x = \int_\sigma fv\mathrm{d}x\mathrm{d}t + \int_{I_i \times \partial K} \boldsymbol{p} \cdot \boldsymbol{n}v\mathrm{d}s\mathrm{d}t.$$

We append subscript h on ∇, append subscript τ on ∂ and append subscript h and τ on u, v, $\boldsymbol{p}$ and $\boldsymbol{q}$. Besides, we use numerical traces $\widehat{u_h^\tau}$ and $\widehat{\boldsymbol{p}_h^\tau}$ to approximate u and $\boldsymbol{p}$ in spatial cross-section $f \in \mathcal{D}_\tau \times \mathcal{E}_h$ and use numerical traces $\widetilde{u_h^\tau}$ to approximate u in temporal cross-section $f \in \mathcal{P}_\tau \times \mathcal{T}_h$,

$$\int_\sigma \boldsymbol{p}_h^\tau \cdot \boldsymbol{q}_h^\tau dxdt = -\int_\sigma u_h^\tau \nabla_h \cdot \boldsymbol{q}_h^\tau \mathrm{d}x\mathrm{d}t + \int_{I_i \times \partial K} \widehat{u_h^\tau}\boldsymbol{q}_h^\tau \cdot \boldsymbol{n}\mathrm{d}s\mathrm{d}t,$$
$$-\int_\sigma u_h^\tau \frac{\partial_\tau v_h^\tau}{\partial_\tau t}\mathrm{d}x\mathrm{d}t + \int_\sigma \boldsymbol{p}_h^\tau \cdot \nabla_h v_h^\tau \mathrm{d}x\mathrm{d}t + \int_K (\widetilde{u_h^\tau} v_h^\tau)|_{t_{i-1}}^{t_i}\mathrm{d}x = \int_\sigma fv_h^\tau \mathrm{d}x\mathrm{d}t + \int_{I_i \times \partial K} \widehat{\boldsymbol{p}_h^\tau} \cdot \boldsymbol{n}v_h^\tau \mathrm{d}s\mathrm{d}t.$$

Then we add over all the elements, use integration by parts and (2.5)

$$\begin{aligned}
\int_\Sigma \boldsymbol{p}_h^\tau \cdot \boldsymbol{q}_h^\tau dxdt &= \int_\Sigma \nabla_h u_h^\tau \cdot \boldsymbol{q}_h^\tau \mathrm{d}x\mathrm{d}t + \int_{\mathcal{D}_\tau \times \mathcal{E}_h} [\![\widehat{u_h^\tau} - u_h^\tau]\!] \cdot \{\boldsymbol{q}_h^\tau\}\mathrm{d}s\mathrm{d}t + \int_{\mathcal{D}_\tau \times \mathcal{E}_h^i} [\boldsymbol{q}_h^\tau] \cdot \{\widehat{u_h^\tau} - u_h^\tau\}\mathrm{d}s\mathrm{d}t, \\
&\int_\Sigma \frac{\partial_\tau u_h^\tau}{\partial_\tau t} v_h^\tau \mathrm{d}x\mathrm{d}t + \int_\Sigma \boldsymbol{p}_h^\tau \cdot \nabla_h v_h^\tau \mathrm{d}x\mathrm{d}t + \sum_{i=0}^{N_t} \int_{\mathcal{T}_h} \left[\widetilde{u_h^\tau}(t_i, \boldsymbol{x}) - u_h^\tau(t_i, \boldsymbol{x})\right] \cdot \{v_h^\tau(t_i, \boldsymbol{x})\}\mathrm{d}x \\
&- \int_\Sigma fv_h^\tau \mathrm{d}x\mathrm{d}t + \sum_{i=1}^{N_t - 1} \int_{\mathcal{T}_h} \left[v_h^\tau(t_i, \boldsymbol{x})\right] \cdot \{\widetilde{u_h^\tau}(t_i, \boldsymbol{x}) - u_h^\tau(t_i, \boldsymbol{x})\}\mathrm{d}x \\
&= \int_{\mathcal{D}_\tau \times \mathcal{E}_h} [\![v_h^\tau]\!] \cdot \{\widehat{\boldsymbol{p}_h^\tau}\}\mathrm{d}s\mathrm{d}t + \int_{\mathcal{D}_\tau \times \mathcal{E}_h^i} [\widehat{\boldsymbol{p}_h^\tau}] \cdot \{v_h^\tau\}\mathrm{d}s\mathrm{d}t.
\end{aligned}$$

Here, we take

$$\begin{aligned}
\widehat{u_h^\tau} &= \{u_h^\tau\} \quad on\ f \in \mathcal{D}_\tau \times \mathcal{E}_h^i, \\
\widehat{u_h^\tau} &= g \quad on\ f \in \mathcal{D}_\tau \times \mathcal{E}_h^\partial, \\
\widetilde{u_h^\tau} &= \{u_h^\tau\} - \eta\left[u_h^\tau\right] \quad on\ f \in \mathcal{P}_\tau^i \times \mathcal{T}_h, \\
\widetilde{u_h^\tau} &= u_0 \quad on\ f \in \{t_0\} \times \mathcal{T}_h, \\
\widetilde{u_h^\tau} &= u_h^\tau \quad on\ f \in \{t_{N_t}\} \times \mathcal{T}_h, \\
\widehat{\boldsymbol{p}_h^\tau} &= \{\nabla_h u_h^\tau\} - \eta[\![u_h^\tau]\!] \quad on\ f \in \mathcal{D}_\tau \times \mathcal{E}_h^i,
\end{aligned}$$

$$\widehat{\boldsymbol{p}_h^\tau} = \nabla_h u_h^\tau - \eta(u-g)\boldsymbol{n} \quad on\ f \in \mathcal{D}_\tau \times \mathcal{E}_h^\partial,$$

where $\eta = \eta_f (h_f)^{-1}$, and η_f can be different by the choice of the face f. And we choose $\boldsymbol{q}_h^\tau = \nabla_h v_h^\tau$, then the space–time LRNN-DG scheme for solving the heat problem is: Find $u_h^\tau \in V_h^\tau$ such that

$$B_{h\tau}(u_h^\tau, v_h^\tau) = l(v_h^\tau) \quad \forall v_h^\tau \in V_h^\tau, \tag{4.8}$$

where

$$\begin{aligned} B_{h\tau}(u_h^\tau, v_h^\tau) = & \int_\Sigma \frac{\partial_\tau u_h^\tau}{\partial_\tau t} v_h^\tau \mathrm{d}x\mathrm{d}t + \int_\Sigma \nabla_h u_h^\tau \cdot \nabla_h v_h^\tau \mathrm{d}x\mathrm{d}t \\ & - \sum_{i=0}^{N_t-1} \int_{\mathcal{T}_h} \left[u_h^\tau(t_i, \boldsymbol{x})\right] \cdot \{v_h^\tau(t_i, \boldsymbol{x})\} \mathrm{d}x - \sum_{i=1}^{N_t-1} \int_{\mathcal{T}_h} \eta \left[u_h^\tau(t_i, \boldsymbol{x})\right] \cdot \left[v_h^\tau(t_i, \boldsymbol{x})\right] \mathrm{d}x \\ & - \int_{D_\tau \times \mathcal{E}_h} \left([\![u_h^\tau]\!] \cdot \{\nabla_h v_h^\tau\} + [\![v_h^\tau]\!] \cdot \{\nabla_h u_h^\tau\} - \eta [\![u_h^\tau]\!] \cdot [\![v_h^\tau]\!] \right) \mathrm{d}s\mathrm{d}t, \end{aligned} \tag{4.9}$$

$$l(v_h^\tau) = \int_\Sigma f v_h^\tau dx dt - \int_{D_\tau \times \mathcal{E}_h^\partial} (g\boldsymbol{n} \cdot \nabla_h v_h^\tau - \eta g v_h^\tau) \mathrm{d}t\mathrm{d}s + \int_{\mathcal{T}_h} u_0(\boldsymbol{x}) v_h^\tau(t_0, \boldsymbol{x}) \mathrm{d}x. \tag{4.10}$$

From the above scheme, we can get a linear system of equations

$$\mathbb{A}U = L, \tag{4.11}$$

where $\mathbb{A}$ is a $N_e M \times N_e M$ matrix, L is a $N_e M \times 1$ vector, and there are $N_e M$ unknown variables $U = \{u_j^{I_i \times K} : I_i \in \mathcal{D}_\tau, K \in \mathcal{T}_h, j = 1, 2, \dots, M\}$. Here, the width of the last hidden layer is M. We look for the least-squares solution for this system. Therefore, the parameters U in the neural networks' output layers are obtained by the linear least-squares computation.

Remark 4.1. Because the variables x and t are treated as inputs of the randomized neural networks, the LRNN-DG scheme (4.8) is based on a space–time approach. Using this approach, we can solve this time-dependent problem in a single least-squares computation, which is more efficient than traditional iterative approaches.

We give the following lemma to show the consistency of the space–time LRNN-DG method.

Lemma 4.2. *The space–time LRNN-DG scheme is consistent, i.e., for the solution $u \in C^0(I; H^2(\Omega))$ of the heat Eq. (4.1), we have*

$$B_{h\tau}(u, v_h^\tau) = l(v_h^\tau) \quad \forall v_h^\tau \in V_h^\tau. \tag{4.12}$$

Proof. We know that $u \in C^0(I; H^2(\Omega))$ implies $[\![u]\!] = 0$, $[\nabla u] = 0$ on $\mathcal{E}_h^i$, $u = g$ on $\mathcal{E}_h^\partial$, $\left[u(t_i, \boldsymbol{x})\right] = 0$ for $i = 1, 2, \dots, N_t - 1$ and $u(t_0, \boldsymbol{x}) = u_0(\boldsymbol{x})$. Then, by the identities (2.4) and (2.5), we have

$$\begin{aligned} B_{h\tau}(u, v_h^\tau) = & \int_\Sigma \frac{\partial u}{\partial t} v_h^\tau \mathrm{d}x\mathrm{d}t + \int_\Sigma \nabla u \cdot \nabla_h v_h^\tau \mathrm{d}x\mathrm{d}t - \int_{D_\tau \times \mathcal{E}_h} [\![v_h^\tau]\!] \cdot \{\nabla u\} \mathrm{d}s\mathrm{d}t \\ & - \int_{D_\tau \times \mathcal{E}_h^\partial} g\boldsymbol{n} \cdot \nabla_h v_h^\tau \mathrm{d}t\mathrm{d}s + \int_{D_\tau \times \mathcal{E}_h^\partial} \eta g v_h^\tau \mathrm{d}t\mathrm{d}s + \int_{\mathcal{T}_h} u_0(\boldsymbol{x}) v_h^\tau(t_0, \boldsymbol{x}) \mathrm{d}x \\ = & \int_\Sigma (\frac{\partial u}{\partial t} - \Delta u) v_h^\tau \mathrm{d}x\mathrm{d}t + \int_{D_\tau \times \mathcal{E}_h} \nabla u \cdot \boldsymbol{n}_K v_h^\tau \mathrm{d}t\mathrm{d}s - \int_{D_\tau \times \mathcal{E}_h} [\![v_h^\tau]\!] \cdot \{\nabla u\} \mathrm{d}s\mathrm{d}t \quad \square \\ & - \int_{D_\tau \times \mathcal{E}_h^\partial} g\boldsymbol{n} \cdot \nabla_h v_h^\tau \mathrm{d}t\mathrm{d}s + \int_{D_\tau \times \mathcal{E}_h^\partial} \eta g v_h^\tau \mathrm{d}t\mathrm{d}s + \int_{\mathcal{T}_h} u_0(\boldsymbol{x}) v_h^\tau(t_0, \boldsymbol{x}) \mathrm{d}x \\ = & l(v_h). \end{aligned}$$

4.2. Space–time LRNN-C^0DG method

In this subsection, we introduce the space–time LRNN-C^0DG method to address the difficulty of choosing a proper value of the penalty parameter in the space–time LRNN-DG.

The setting of local randomized neural networks and the partition is the same as described above. In this approach, we enforce the C^0-continuous condition on $f \in \left(\mathcal{D}_\tau \times \mathcal{E}_h^i\right) \cup \left(P_\tau^i \times \mathcal{T}_h\right)$, the initial condition on $f \in \{t_0\} \times \mathcal{T}_h$ and Dirichlet boundary condition on $f \in \mathcal{D}_\tau \times \mathcal{E}_h^\partial$ to solve this problem. We add the additional equations to enforce the solution to satisfy the boundary condition (4.1c). Specifically, we choose some points on boundary faces, denoted by $P_h^g = \{(t_j^f, \boldsymbol{x}_j^f) \in f : f \in \mathcal{D}_\tau \times \mathcal{E}_h^\partial, j = 1, 2, \dots, N_h^g\}$ and $|P_h^g| = N_h^g$, and require that

$$u_h^\tau(t_j^f, \boldsymbol{x}_j^f) = g(t_j^f, \boldsymbol{x}_j^f) \quad \forall (t_j^f, \boldsymbol{x}_j^f) \in P_h^g. \tag{4.13}$$

We add additional equations to ensure that the solution satisfies the initial condition (4.1b). In particular, we choose points at the initial time t_0, denoted by $P_\tau^{t_0} = \{(t_0, \boldsymbol{x}_j^f) \in f : f \in \{t_0\} \times \mathcal{T}_h, j = 1, 2, \dots, N_\tau^{t_0}\}$ with $|P_\tau^{t_0}| = N_\tau^{t_0}$, and enforce that

$$u_h^\tau(t_0, \boldsymbol{x}_j^f) = u_0(\boldsymbol{x}_j^f) \quad \forall (t_0, \boldsymbol{x}_j^f) \in P_\tau^{t_0}. \tag{4.14}$$

We also ensure that the numerical solution u_h satisfies certain C^0-continuity conditions along the spatial and temporal directions across the interior faces $f \in \left(\mathcal{D}_\tau \times \mathcal{E}_h^i\right) \cup \left(\mathcal{P}_\tau^i \times \mathcal{T}_h\right)$. To achieve this, we pick points on the boundary faces denoted by $P_h^{Si} = \{(t_j^f, \boldsymbol{x}_j^f) \in f : f \in \left(\mathcal{D}_\tau \times \mathcal{E}_h^i\right), j = 1, 2, \ldots, N_h^i\}$ with $|P_h^{Si}| = N_h^i$ and $P_\tau^{Ti} = \{(t_j^f, \boldsymbol{x}_j^f) \in f : f \in \left(\mathcal{P}_\tau^i \times \mathcal{T}_h\right), j = 1, 2, \ldots, N_\tau^i\}$ with $|P_\tau^{Ti}| = N_\tau^i$, and enforce that:

$$[\![u_h^\tau(t_j^f, \boldsymbol{x}_j^f)]\!] = 0 \quad \forall (t_j^f, \boldsymbol{x}_j^f) \in P_h^{Si}. \tag{4.15}$$

$$\left[u_h^\tau(t_j^f, \boldsymbol{x}_j^f)\right] = 0 \quad \forall (t_j^f, \boldsymbol{x}_j^f) \in P_\tau^{Ti}. \tag{4.16}$$

The system of equations with respect to (4.13)–(4.16) confirms the boundary conditions, initial conditions and continuity conditions of interior edges, so we have

$$\mathbb{A}_2 U = L_2,$$

where $\mathbb{A}_2$ is a $(N_h^g + N_\tau^{t_0} + N_h^i + N_\tau^i) \times N_e M$ matrix, U is a $N_e M \times 1$ unknown vector, L_2 is a $(N_h^g + N_\tau^{t_0} + N_h^i + N_\tau^i) \times 1$ vector. This system of equations reduces the jump of the solution to nearly zero, so the LRNN-C^0DG scheme aims to find $u_h^\tau \in V_h^\tau$ such that

$$\begin{aligned}
B_{h\tau}^0(u_h^\tau, v_h^\tau) &= l^0(v_h^\tau) && \forall\, v_h^\tau \in V_h^\tau,\\
[\![u_h^\tau(t_j^f, \boldsymbol{x}_j^f)]\!] &= 0 && \forall (t_j^f, \boldsymbol{x}_j^f) \in P_h^{Si},\\
\left[u_h^\tau(t_j^f, \boldsymbol{x}_j^f)\right] &= 0 && \forall (t_j^f, \boldsymbol{x}_j^f) \in P_\tau^{Ti},\\
u_h^\tau(t_j^f, \boldsymbol{x}_j^f) &= g(t_j^f, \boldsymbol{x}_j^f) && \forall (t_j^f, \boldsymbol{x}_j^f) \in P_h^g,\\
u_h^\tau(t_0, \boldsymbol{x}_j^f) &= u_0(\boldsymbol{x}_j^f) && \forall (t_0, \boldsymbol{x}_j^f) \in P_\tau^{t_0},
\end{aligned} \tag{4.17}$$

where

$$\begin{aligned}
B_{h\tau}^0(u_h^\tau, v_h^\tau) &= \int_\Sigma \frac{\partial_\tau u_h^\tau}{\partial_\tau t} v_h^\tau \mathrm{d}x\mathrm{d}t + \int_\Sigma \nabla_h u_h^\tau \cdot \nabla_h v_h^\tau \mathrm{d}x\mathrm{d}t\\
&\quad - \int_{\mathcal{D}_\tau \times \mathcal{E}_h} ([\![u_h^\tau]\!] \cdot \{\nabla_h v_h^\tau\} + [\![v_h^\tau]\!] \cdot \{\nabla_h u_h^\tau\}) \mathrm{d}s\mathrm{d}t,\\
l^0(v_h^\tau) &= \int_\Sigma f v_h^\tau dx dt - \int_{\mathcal{D}_\tau \times \mathcal{E}_h^\partial} g\boldsymbol{n} \cdot \nabla_h v_h^\tau \mathrm{d}t\mathrm{d}s.
\end{aligned} \tag{4.18}$$

Then we can get global stiffness matrix $\mathbb{A}_1$ and the right-hand side L_1 of (4.8), where $\mathbb{A}_1$ is a $N_e M \times N_e M$ matrix, L_1 is a $N_e M \times 1$ vector. Combine $\mathbb{A}_1$, $\mathbb{A}_2$, L_1 and L_2, we get a linear system

$$\begin{bmatrix} \mathbb{A}_1 \\ \mathbb{A}_2 \end{bmatrix} U = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix}. \tag{4.19}$$

We seek the least-squares solution to this system, and once we have obtained the weights of the output layer in each local neural network via linear least-squares computation, we can obtain all the values of the problem (4.1) in the domain Σ. Notably, this scheme does not rely on the penalty parameter.

Remark 4.3. Based on numerical experiments, we observe that the scheme (4.17) performs well. Surprisingly, even when we remove the term $-\int_{D\tau \times \mathcal{E}h} [\![u_h^\tau]\!] \cdot \{\nabla_h v_h^\tau\} \mathrm{d}s\mathrm{d}t$ in (4.18), the modified scheme still yields good numerical results:

$$\widetilde{B_{h\tau}^0}(u_h^\tau, v_h^\tau) = \int_\Sigma f v_h^\tau \mathrm{d}x\mathrm{d}s \quad \forall\, v_h^\tau \in V_h^\tau, \tag{4.20}$$

where

$$\widetilde{B_{h\tau}^0}(u_h^\tau, v_h^\tau) = \int_\Sigma \frac{\partial_\tau u_h^\tau}{\partial_\tau t} v_h^\tau \mathrm{d}x\mathrm{d}t + \int_\Sigma \nabla_h u_h^\tau \cdot \nabla_h v_h^\tau \mathrm{d}x\mathrm{d}t - \int_{\mathcal{D}_\tau \times \mathcal{E}_h} [\![v_h^\tau]\!] \cdot \{\nabla_h u_h^\tau\} \mathrm{d}s\mathrm{d}t. \tag{4.21}$$

4.3. Space–time LRNN-C^1DG method

We briefly outline the space–time LRNN-C^1DG method for solving the heat equation, building upon the ideas presented in Section 2.5.

We introduce a system of equations to enforce that the solution u_h^τ satisfies the boundary condition, the initial condition, and C^0-continuity conditions along both spatial and temporal directions, similar to (4.13)–(4.16). Additionally, we require that the numerical solution u_h^τ satisfies C^1-continuity conditions across the interior faces $f \in (\mathcal{D}_\tau \times \mathcal{E}_h^i)$ along the spatial direction:

$$[\nabla_h u_h^\tau(\boldsymbol{x}_j^f)] = 0 \quad \forall \boldsymbol{x}_j^f \in P_h^{Si}. \tag{4.22}$$

So we have the new problem that finding $u_h \in V_h^\tau$ such that

$$B_{h\tau}^\sigma(u_h^\tau, v_h^\tau) = \int_\sigma f v_h^\tau \mathrm{d}t\mathrm{d}x \quad \forall\, \sigma = \left(I_i \times K\right) \in \left(\mathcal{D}_\tau \times \mathcal{T}_h\right), \tag{4.23}$$

Table 1
Errors of the LRNN-DG method for 1-d Helmholtz equation in Example 5.1.

h	2^{-4}		2^{-5}		2^{-6}	
Norm / DoF_K	L^2	H^1	L^2	H^1	L^2	H^1
20	4.14E−03	2.80E+00	1.17E−04	1.95E−01	1.06E−05	2.53E−02
40	3.46E−05	7.20E−02	2.19E−06	5.44E−03	3.65E−08	1.76E−04
80	1.70E−06	3.00E−03	4.03E−09	1.60E−05	3.32E−10	2.54E−06
160	2.46E−07	3.86E−04	6.12E−10	2.49E−06	1.72E−10	1.31E−06
320	5.23E−08	1.03E−04	2.74E−10	1.11E−06	9.70E−11	8.26E−07
640	1.02E−08	2.56E−05	2.49E−10	7.97E−07	9.62E−11	6.85E−07

where

$$B^{\sigma}_{h\tau}(u^{\tau}_h, v^{\tau}_h) = -\int_{\sigma} u^{\tau}_h \frac{\partial v^{\tau}_h}{\partial t}\mathrm{d}t\mathrm{d}x + \int_K u^{\tau}_h v^{\tau}_h|^{t_i}_{t_{i-1}}\mathrm{d}x + \int_{\sigma}\nabla u^{\tau}_h\cdot\nabla v^{\tau}_h\mathrm{d}t\mathrm{d}x - \int_{I_i\times\partial K}\nabla u^{\tau}_h\cdot\boldsymbol{n}v^{\tau}_h\mathrm{d}s\mathrm{d}t. \tag{4.24}$$

Then we can get the global stiffness matrix $\mathbb{A}_1$ and the right-hand side L_1, where $\mathbb{A}_1$ is a $N_eM\times N_eM$ matrix, L_1 is a $N_eM\times 1$ vector, such that $\mathbb{A}_1U = L_1$.

Let u^{τ}_h satisfy the conditions (4.13)–(4.16), C^1-continuity condition (4.22). Then we can obtain a system of equations

$$\mathbb{A}_2U = L_2, \tag{4.25}$$

where $\mathbb{A}_2$ is a $(N^g_h + N^{t_0}_{\tau} + 2N^i_h + N^i_{\tau})\times N_eM$ matrix, U is a $N_eM\times 1$ matrix of unknown variables, L_2 is a $(N^g_h + N^{t_0}_{\tau} + 2N^i_h + N^i_{\tau})\times 1$ vector. Combine $\mathbb{A}_1$, $\mathbb{A}_2$, L_1 and L_2, we obtain

$$\begin{bmatrix}\mathbb{A}_1\\ \mathbb{A}_2\end{bmatrix}U = \begin{bmatrix}L_1\\ L_2\end{bmatrix}. \tag{4.26}$$

We look for the least-squares solution to this system. After the weights of the output layer in each local neural network are obtained by the linear least-squares computation, we can get all the values of the problem (4.1) in the domain Σ.

Remark 4.4. We have presented a space–time formulation for solving the heat equation based on LRNN-DG in the above. Convergence analysis of this method for dynamic problems, however, remains elusive at this point. The main difficulty lies in proving the stability of the space–time formulation. This outstanding problem requires further research and will be addressed in a future publication.

5. Numerical examples

In this section, we present several test problems to demonstrate the performance of the methods developed herein.

For implementing the neural network, we use the PyTorch library in Python, as stated in Section 2.1. Each local neural network, for each sub-domain, consists of a single hidden layer with pre-assigned and fixed parameters that are uniform random values generated from $[-w_0, w_0]$, where w_0 is a constant. Note that w_0 affects the shape of the basis functions, so it has an influence on the accuracy of the proposed methods, see more discussion in [45]. The overall neural network is composed of all the local neural networks, which are coupled with one another through the DG formulation or the C^0/C^1 conditions. The integrals in the weak formulations are computed using Gaussian quadrature, and we employ different numbers of quadrature points for integrals in different examples. For the following experiments, we use the Tanh function as the activation function. Other activation functions may be suitable for different problems. For solving the linear system of equations about the output-layer coefficients, we use the linear least-squares routine from LAPACK, available through wrapper functions in the scipy package in Python. The DoF_K or DoF_{σ} in all tables below denote the degrees of freedom on each subdomain, and the DoF in the figures refers to the total degrees of freedom.

Example 5.1 (*One-Dimensional Helmholtz Equation*)**.** The first test problem is a one-dimensional Helmholtz equation on the domain $\Omega = [0, 1]$,

$$\begin{aligned} -u_{xx} + \lambda u &= f(x),\\ u(0) &= g_1(x),\\ u(1) &= g_2(x), \end{aligned}$$

where the $\lambda = 10$ and $f(x)$ is a prescribed source term, $g_1(x)$ and $g_2(x)$ are boundary conditions, with the manufactured exact solution

$$u(x) = \frac{1}{2}\left(x^2+1\right)\mathrm{e}^{\cos\left(81x^3+8\pi-24\right)}.$$

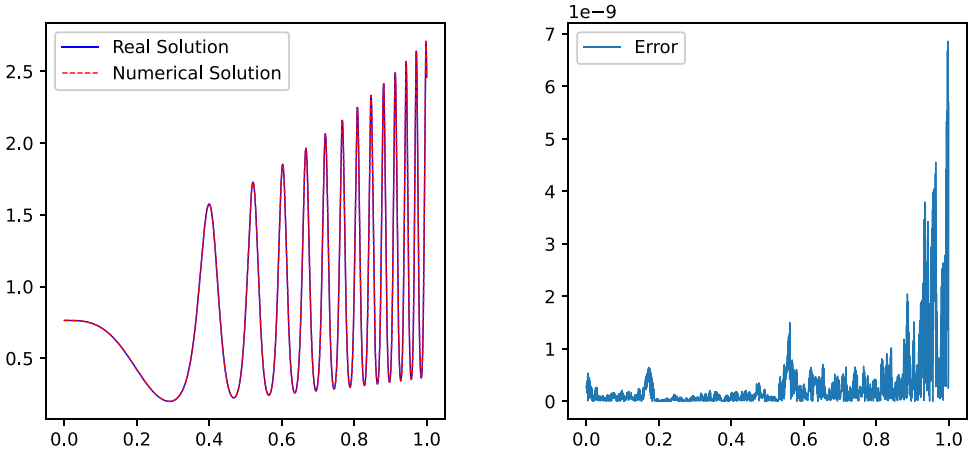
We partition the interval Ω into non-overlapping uniform subintervals of size h and choose the source term f such that the solution satisfies the boundary value problem given above. The numerical errors in the L^2 norm and the H^1 seminorm for different numbers of degrees of freedom are shown in Table 1, Table 2, and Table 3.

Table 2
Errors of the LRNN-C^0DG method for 1-d Helmholtz equation in Example 5.1.

h	2^{-4}		2^{-5}		2^{-6}	
Norm / DoF_K	L^2	H^1	L^2	H^1	L^2	H^1
20	1.14E−02	5.82E+00	3.45E−04	4.02E−01	2.88E−05	8.18E−02
40	9.24E−05	1.34E−01	2.07E−06	6.11E−03	6.64E−07	2.98E−03
80	1.40E−06	2.46E−03	4.12E−09	1.30E−05	5.19E−10	4.90E−06
160	7.41E−08	1.10E−04	3.79E−10	1.62E−06	1.24E−10	1.04E−06
320	1.48E−08	1.91E−05	1.32E−10	8.29E−07	5.47E−11	6.32E−07
640	3.56E−09	1.15E−05	1.33E−10	9.66E−07	5.11E−11	6.74E−07

Table 3
Errors of the LRNN-C^1DG method for 1-d Helmholtz equation in Example 5.1.

h	2^{-4}		2^{-5}		2^{-6}	
Norm / DoF_K	L^2	H^1	L^2	H^1	L^2	H^1
20	3.10E−03	2.19E+00	9.46E−05	1.40E−01	8.86E−05	2.68E−01
40	1.02E−04	1.19E−01	8.89E−06	1.35E−02	7.07E−08	2.95E−04
80	1.13E−06	1.17E−03	2.22E−09	8.07E−06	1.14E−09	8.31E−06
160	1.14E−07	1.48E−04	4.35E−10	1.21E−06	8.81E−11	6.80E−07
320	5.79E−09	6.76E−06	1.56E−10	6.74E−07	5.87E−11	4.62E−07
640	2.87E−09	9.48E−06	8.06E−11	5.42E−07	4.11E−11	4.96E−07



(a) Real and numerical solution (b) Absolute error

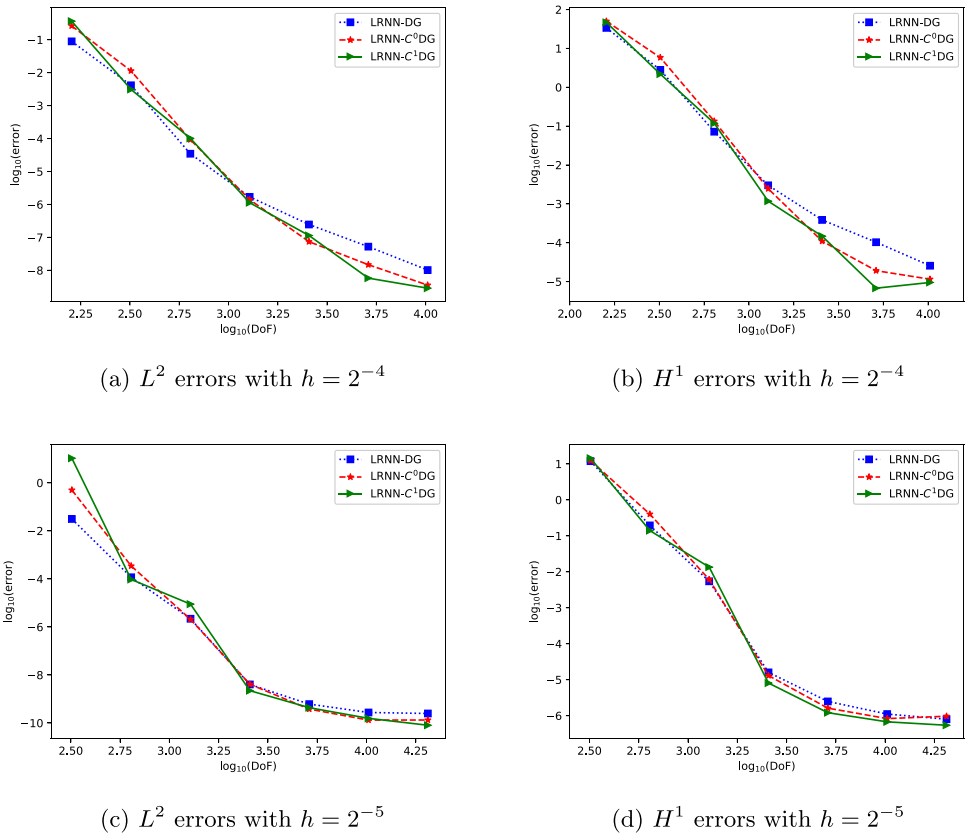
Fig. 1. Solution and error obtained from LRNN-DG methods in Example 5.1 with $h = 2^{-5}$ and $\mathrm{DoF}_K = 160$.

Table 1 presents the LRNN-DG errors in terms of the number of degrees of freedom per subinterval (DoF_K) and the subinterval size h. In these tests, we set the penalty parameter η to 4, the parameter w_0 of the uniform distribution to 5.0, and use 70 quadrature points in each subinterval. It can be observed that for a fixed h, the errors initially decrease rapidly as the degrees of freedom per subinterval increase, and then the reduction slows down. For a fixed DoF_K, there is a general decrease in errors as h decreases. To further illustrate the performance of our methods, we show the real solution, numerical solution, and error in Fig. 1 for a mesh size $h = 2^{-5}$ and $\mathrm{DoF}_K = 160$. It can be seen that the LRNN-DG method can approximate the real solution well, whether the frequency is high or low.

Table 2 shows the corresponding errors of LRNN-C^0DG in terms of the number of degrees of freedom per interval and the size of the element, while Table 3 displays the errors of LRNN-C^1DG in terms of DoF_K and h. For LRNN-C^0DG, the parameter w_0 is set to 6 and the number of quadrature points is 70. For LRNN-C^1DG, the parameter w_0 is set to 5.7 and the number of quadrature points is 70. The trends of errors with respect to the size h and the DoF_K are similar to those observed for LRNN-DG.

Fig. 2 compares the errors of the three methods for different sizes h and different norms. Generally, we can observe that the performance of the three methods is similar, with LRNN-C^1DG performing better than LRNN-C^0DG and LRNN-C^0DG performing better than LRNN-DG. However, it is important to note that the last two methods involve more equations due to continuity restrictions on edges, so it is not surprising that they have better performance.

Additionally, in Example 5.1, we investigate the performance of the LRNN-DG method with reduced DoF_K as the mesh size h decreases. Table 4 presents the errors obtained with fewer local basis functions and smaller grid sizes. In this test, we set



(a) L^2 errors with $h = 2^{-4}$ (b) H^1 errors with $h = 2^{-4}$

(c) L^2 errors with $h = 2^{-5}$ (d) H^1 errors with $h = 2^{-5}$

Fig. 2. Comparison of the errors obtained by the three methods in Example 5.1.

Table 4
Errors of the LRNN-DG method for smaller mesh size h in Example 5.1.

DoF_K	10		20	
Norm / h	L^2	H^1	L^2	H^1
2^{-3}	3.09E+01	3.10E+02	3.09E+01	3.07E+02
2^{-4}	3.14E−01	4.12E+01	3.04E−01	3.55E+01
2^{-5}	1.43E−02	9.63E+00	8.53E−03	5.93E+00
2^{-6}	5.77E−04	7.49E−01	7.84E−04	9.38E−01
2^{-7}	3.32E−06	8.43E−03	1.96E−06	5.13E−03
2^{-8}	6.11E−08	2.68E−04	1.32E−08	6.72E−05
2^{-9}	1.20E−08	9.99E−05	2.08E−09	2.16E−05

$\eta = 1$, $w_0 = 0.4$, and use 14 quadrature points in each dimension. Table 4 demonstrates that the LRNN-DG method can accurately approximate the solution even with relatively few basis functions, as long as the mesh size is small enough.

Example 5.2 (*Two-Dimensional Poisson Equation*). Consider the Poisson Eq. (2.2) on $\Omega = [0,1]^2$ and the exact solution $u = e^{x+y}\sin(3\pi x + 0.5\pi)\cos(\pi y + 0.2\pi)$.

In this example, we partition the domain Ω into non-overlapping uniform square elements with edge length h. The number of quadrature points used in each direction is 14. The numerical errors in the L^2 norm and the H^1 seminorm for different DoF_K and h are shown in Table 5, Table 6, and Table 7.

Table 5 presents the errors of LRNN-DG in terms of degrees of freedom on each element and the size of the element. In this set of tests, the weight/bias coefficients in the hidden layer of each local network are initialized with uniform random values generated in the range $[-1.2, 1.2]$, and the penalty parameter is set to $\eta_e = 1$. The L^2 norm and H^1 seminorm errors initially decrease rapidly with increasing DoF_K and then more slowly for fixed h. We also observe that reducing the size h leads to a decrease in errors for a fixed DoF_K.

Tables 6 and 7 show the errors of LRNN-C^{0}DG and LRNN-C^{1}DG, respectively, in terms of the number of degrees of freedom on each element and the size of the element. In Table 6, the parameter w_0 is set to 1.2, and the number of collocation points is 14.

Table 5
Errors of the LRNN-DG method for 2-d Poisson equation in Example 5.2.

h	2^{-1}		2^{-2}		2^{-3}	
Norm / DoF_K	L^2	H^1	L^2	H^1	L^2	H^1
10	3.40E+00	3.95E+01	3.04E+00	7.42E+01	7.64E−01	3.17E+01
20	3.62E−01	6.82E+00	1.26E+00	4.99E+01	5.70E−02	4.91E+00
40	1.49E−02	5.46E−01	2.04E−03	1.43E−01	1.47E−03	2.18E−01
80	1.13E−04	6.31E−03	1.39E−05	1.53E−03	2.64E−06	6.01E−04
160	3.07E−06	2.19E−04	1.63E−07	2.31E−05	3.11E−08	8.92E−06
320	6.60E−07	6.08E−05	2.16E−07	3.51E−05	2.94E−08	1.13E−05

Table 6
Errors of the LRNN-C^0DG method for 2-d Poisson equation in Example 5.2.

h	2^{-1}		2^{-2}		2^{-3}	
Norm / DoF_K	L^2	H^1	L^2	H^1	L^2	H^1
10	9.62E−01	1.27E+01	2.81E−01	6.88E+00	3.29E−01	5.13E+00
20	1.46E−01	2.55E+00	3.03E−02	1.35E+00	8.57E−03	7.13E−01
40	1.40E−02	3.97E−01	2.24E−03	1.43E−01	4.57E−04	5.89E−02
80	3.39E−04	1.53E−02	1.55E−05	1.54E−03	4.64E−06	9.39E−04
160	1.00E−06	6.90E−05	4.21E−08	5.80E−06	1.23E−08	3.16E−06
320	4.11E−07	2.80E−05	1.24E−07	2.08E−05	1.98E−08	7.20E−06

Table 7
Errors of the LRNN-C^1DG method for 2-d Poisson equation in Example 5.2.

h	2^{-1}		2^{-2}		2^{-3}	
Norm / DoF_K	L^2	H^1	L^2	H^1	L^2	H^1
10	1.31E+00	1.33E+01	1.04E+00	8.92E+00	8.96E−01	8.59E+00
20	2.80E−01	2.97E+00	4.42E−02	1.15E+00	1.97E−02	8.08E−01
40	2.28E−02	4.99E−01	1.86E−03	8.19E−02	5.24E−04	4.45E−02
80	2.45E−04	5.60E−03	1.05E−05	7.59E−04	2.99E−06	4.26E−04
160	1.03E−06	4.46E−05	7.57E−08	4.72E−06	7.91E−09	1.46E−06
320	3.07E−07	1.09E−05	1.78E−08	1.66E−06	7.96E−09	9.62E−07

Table 8
Condition number of stiffness matrices computed by LRNN-DG in Example 5.2.

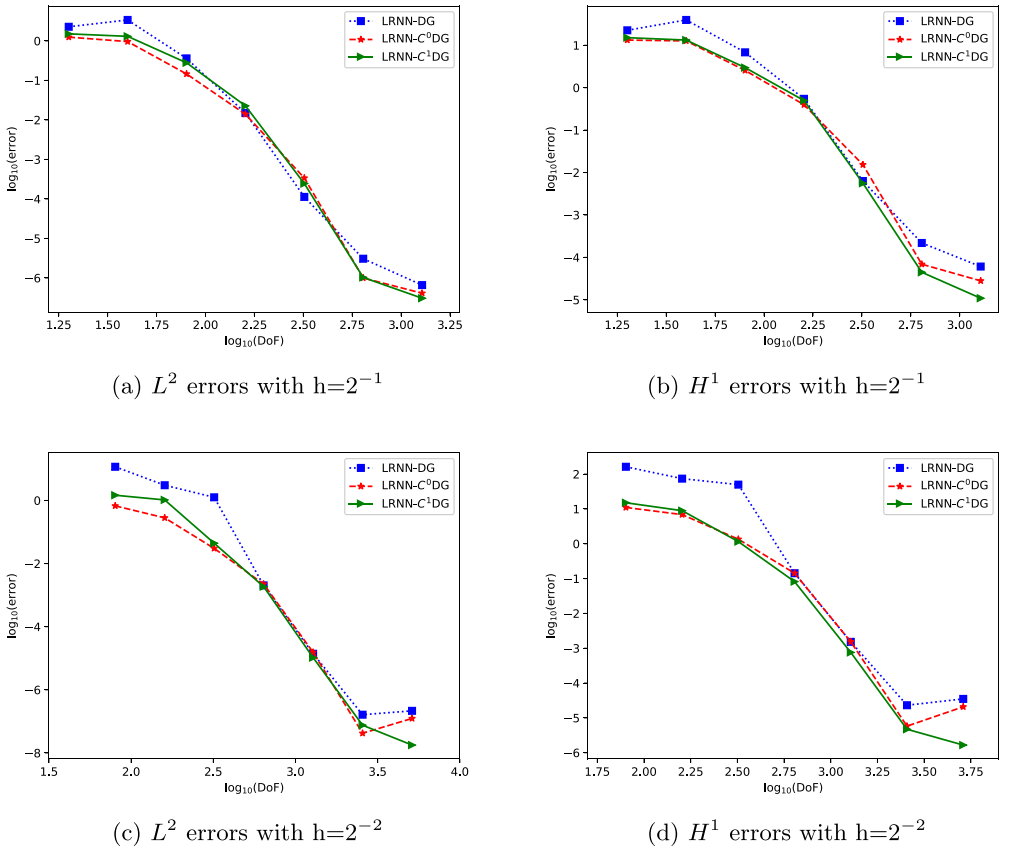
h / DoF_K	2^{-1}	2^{-2}	2^{-3}
5	5.57E+02	7.81E+04	2.65E+05
10	1.12E+04	2.77E+06	4.97E+06
20	8.72E+06	6.60E+08	5.95E+08
40	2.36E+10	2.13E+12	2.56E+12
80	1.01E+16	3.17E+16	1.46E+17
160	8.47E+16	3.97E+17	2.04E+18
320	2.13E+17	1.11E+18	3.41E+18

Table 9
Norm of $[\![u_h]\!]$, $[\nabla u_h]$, and $u_h - g$ computed by LRNN-C^1DG in Example 5.2.

N_e	Vertical Edge			Horizontal edge		
	$\|[\![u_h]\!]\|_{0,e}$	$\|[\nabla u_h]\|_{0,e}$	$\|u_h - g\|_{0,e}$	$\|[\![u_h]\!]\|_{0,e}$	$\|[\nabla u_h]\|_{0,e}$	$\|u_h - g\|_{0,e}$
5	2.46E−05	3.77E−04	4.24E−06	6.01E−05	3.87E−04	1.87E−05
10	1.91E−08	1.68E−06	5.14E−09	3.23E−08	1.27E−06	7.67E−09
20	1.07E−11	1.24E−10	5.43E−11	6.80E−11	4.12E−10	1.46E−10
40	2.08E−12	5.32E−12	5.66E−12	1.08E−12	1.48E−11	4.72E−12
80	2.57E−12	3.29E−12	5.90E−12	1.67E−12	1.51E−12	6.40E−13

In Table 7, the parameter w_0 is set to 1.0, and the number of collocation points is 14. The trends observed for LRNN-C^0DG and LRNN-C^1DG are similar to those of the LRNN-DG method.

Fig. 3 compares the errors of the three methods for different sizes h and different norms. Overall, the performance of the three schemes is similar, with LRNN-C^1DG outperforming LRNN-C^0DG and LRNN-C^0DG outperforming LRNN-DG.



(a) L^2 errors with h=2^{-1} (b) H^1 errors with h=2^{-1}

(c) L^2 errors with h=2^{-2} (d) H^1 errors with h=2^{-2}

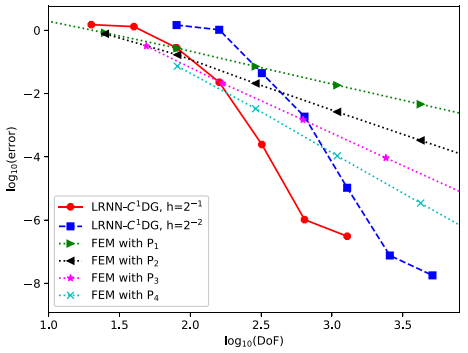
Fig. 3. Errors obtained from three different methods in Example 5.2.

We also compare the proposed methods with the finite element method and the discontinuous Galerkin method programmed with FEniCS [46] for this problem. Fig. 4 compares the performance of the proposed LRNN-C^1DG and LRNN-DG methods with the FEM and DG methods using piecewise P_k polynomial functions. The results show that the proposed methods achieve significantly smaller errors than the traditional methods with the same degrees of freedom. However, as the number of basis functions increases, the errors of the LRNN methods tend to stagnate. This could be attributed to the fact that the basis functions $\{\phi_j^K(\boldsymbol{x}) : j = 1, 2, \ldots, M\}$ may become linearly dependent as M increases, resulting in a rank deficient linear system. To further support this observation, we provide Table 8 which shows the condition number of the global stiffness matrices of the LRNN-DG method in Example 5.2 with different numbers of basis functions and mesh sizes. As shown in the table, the condition number increases with the increase of DoF_K and the decrease of h. As shown in Table 4 of Example 5.1, one way to reduce errors is to increase the number of subdomains, i.e., use elements with a smaller size h. Another approach is to design better neural networks that can provide improved basis functions. This is an aspect we will further explore in future work.

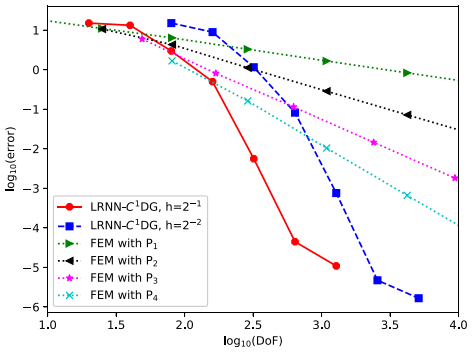
Fig. 5 illustrates the distribution of point-wise absolute errors computed using the three methods with an element size of $h = 2^{-2}$ and $\text{Dof}_K = 320$. It can be observed that the absolute error of LRNN-DG is larger but smoother, while the absolute error of LRNN-C^0DG and LRNN-C^1DG is smaller but with a larger variation.

Table 9 provides numerical evidence to support the reasonableness of Assumption 3.5 and Assumption 3.7. We perform a test to examine how the jump of the numerical solution u_h and its gradient ∇u_h vary as the number of collocation points increases using the LRNN-C^1DG method. We consider a vertical interior edge and a horizontal interior edge and compute the $L^2(e)$-norm of $[\![u_h]\!]$ and $[\nabla u_h]$ on these edges. We also consider a vertical boundary edge and a horizontal boundary edge and compute the $L^2(e)$-norm of $u_h - g$ on these boundary edges. In these tests, $\text{Dof}_K = 320$, and the size of element is $h = 2^{-2}$. Table 9 lists these errors corresponding to a set of collocation points. We observe that $\|[\![u_h]\!]\|_{0,e}$, $\|[\nabla u_h]\|_{0,e}$ and $\|u_h - g\|_{0,e}$ decrease rapidly with increasing number of collection points N_e, reaching a level close to machine zero as N_e becomes large.

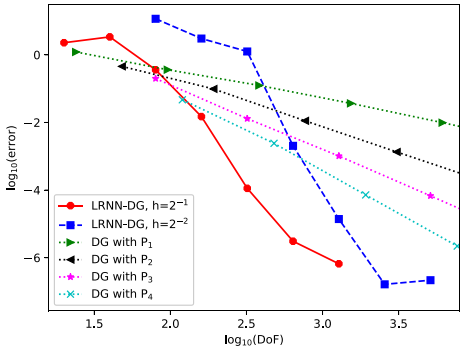
Example 5.3 (*2-D Poisson Equation with Corner Singularities*)**.** In this experiment, we use the LRNN-DG methods to solve the Poisson problem (2.2) with a non-smooth solution around a reentrant corner, which has been considered recently in [47]. The problem is defined on the L-shaped domain $\Omega = \overline{OABCDEO}$, as shown in Fig. 6. The exact solution is given by $u(x, y) = r^{\frac{2}{3}k} \sin(\frac{2}{3}k\theta)$, where (r, θ) denotes the polar coordinates, and $k \geq 1$ is a prescribed integer. We employ a source term $f = 0$ and set the boundary condition g according to the exact solution.



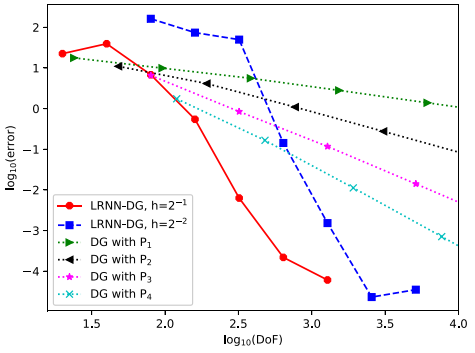
(a) L^2 errors by FEM and LRNN-C^1DG



(b) H^1 errors by FEM and LRNN-C^1DG



(c) L^2 errors by DG and LRNN-DG



(d) H^1 errors by DG and LRNN-DG

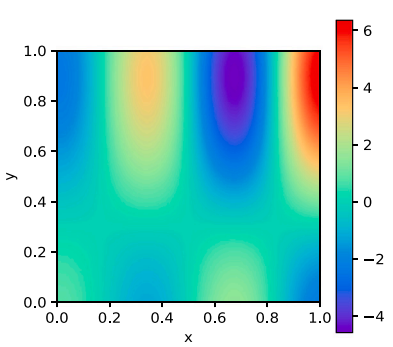
Fig. 4. Errors obtained by FEM, DG and LRNN with DG methods in Example 5.2.

Table 10
Errors of the LRNN-DG method for 2-d Poisson equation with the non-smooth solution in Example 5.3.

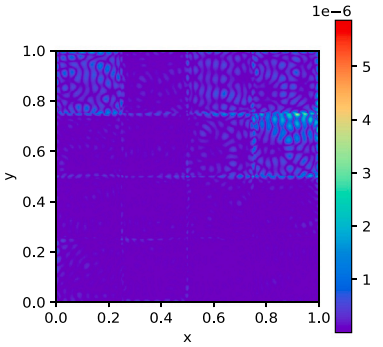
k	1		3		5	
Norm / h, DoF$_K$	L^2	H^1	L^2	H^1	L^2	H^1
1, 160	3.14E−04	3.25E−02	1.86E−07	6.51E−06	1.32E−06	4.89E−05
1/2, 160	1.00E−04	2.08E−02	4.78E−08	3.44E−06	1.50E−07	1.10E−05
1/4, 160	3.21E−05	1.30E−02	1.93E−08	2.88E−06	3.89E−08	5.84E−06
1/4, 40	4.65E−04	3.45E−02	3.22E−02	2.63E+00	4.00E−03	3.26E−01
1/4, 80	8.20E−05	1.72E−02	1.29E−06	1.52E−04	2.08E−06	2.48E−04
1/4, 160	3.21E−05	1.30E−02	1.93E−08	2.88E−06	3.89E−08	5.84E−06
1/4, 320	3.55E−05	1.14E−02	1.28E−08	2.03E−06	2.38E−08	3.83E−06

The exact solution is smooth in Ω only when k is a multiple of 3. Otherwise, the $\lceil \frac{2}{3}k \rceil$- th derivative of the solution is singular at the reentrant corner and the solution is non-smooth. Moreover, the solution becomes smoother as the integer k increases. We apply the LRNN-DG method and LRNN-C^1DG method to solve this problem with $k = 1, 3, 5$. We divide the square domains $\overline{DEOG}$, $\overline{GOFC}$ and $\overline{OABF}$ into smaller square subdomains $\{K\}$ uniformly, and h is the size of each subdomain. Then we show the performances of the LRNN-DG method and LRNN-C^1DG method with different degrees of freedom and sizes of subdomains in Table 10 and Table 11 respectively. For the LRNN-DG method, we choose $w_0 = 2.2$, $\eta = 130/h$ when $k = 1$, $w_0 = 1.4$, $\eta = 5/h$ when $k = 3$ and $w_0 = 1.4$, $\eta = 10/h$ when $k = 5$. For the LRNN-C^1DG method, the number of the collocation points on each edge is 11, we choose $w_0 = 1.1$ when $k = 1$, $w_0 = 0.6$ when $k = 3$ and $w_0 = 0.9$, when $k = 5$.

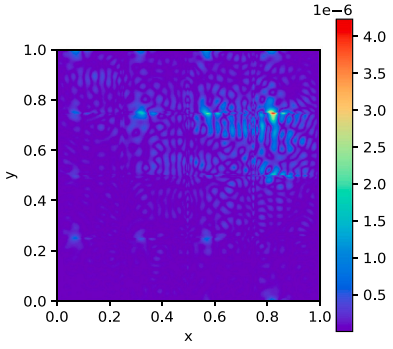
From Tables 10 and 11, we observe that with both methods the accuracy improves with more degrees of freedom in each subdomain and smaller subdomains, for both smooth and non-smooth solutions. The LRNN-DG method, which is based on a weak formulation, has a better performance for non-smooth solutions than the LRNN-C^1DG method. However, for the smooth solution, the result is opposite. One possible reason is that the penalty parameter of the LRNN-DG method increases the condition number of the stiffness matrix. We also compare our results with those in [47]. The LRNN-DG method and LRNN-C^1DG method achieve smaller errors for non-smooth solutions, but for smooth solutions, the methods in [47] perform better.



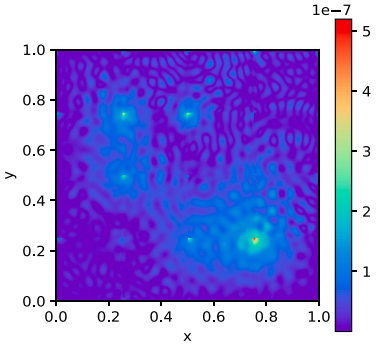
(a) Exact solution



(b) Absolute error for LRNN-DG method



(c) Absolute error for LRNN-C^0DG method



(d) Absolute error for LRNN-C^1DG method

Fig. 5. Absolute errors computed by three methods in Example 5.2.

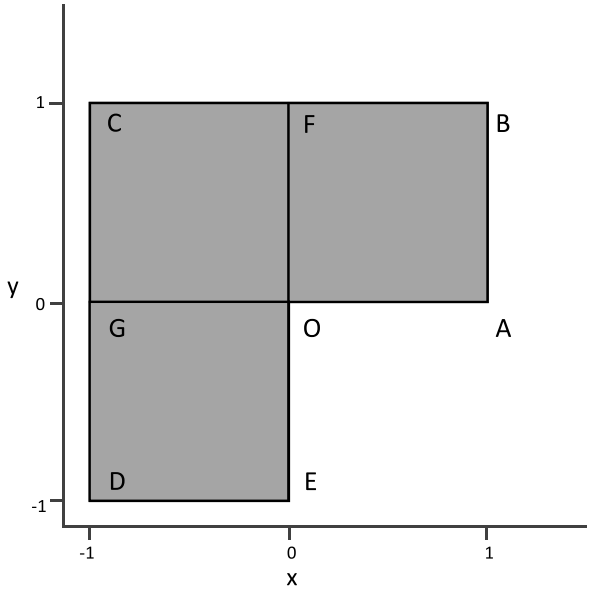


Fig. 6. The L-shaped domain $\Omega = \overline{OABCDEO}$ with an reentrant corner at O.

Table 11
Errors of the LRNN-C^1DG method for 2-d Poisson equation with the non-smooth solution in Example 5.3.

k	1		3		5	
Norm / h, DoF_K	L^2	H^1	L^2	H^1	L^2	H^1
1, 160	5.10E−03	1.53E−01	1.10E−08	2.43E−07	3.91E−06	1.12E−04
1/2, 160	1.92E−03	7.36E−02	4.56E−09	1.88E−07	1.14E−07	5.38E−06
1/4, 160	1.03E−03	4.61E−02	1.57E−09	1.25E−07	1.09E−08	8.10E−07
1/4, 40	4.05E−03	1.83E−01	2.52E−05	9.82E−04	1.69E−04	7.10E−03
1/4, 80	2.62E−03	9.87E−02	1.23E−08	8.47E−07	4.60E−07	3.32E−05
1/4, 160	1.03E−03	4.61E−02	1.57E−09	1.25E−07	1.09E−08	8.10E−07
1/4, 320	1.39E−03	4.29E−02	1.27E−09	1.12E−07	7.44E−09	4.71E−07

Table 12
Errors of the space–time LRNN-DG method when $t = 1$ in Example 5.4.

τ, h	2^{-1}		2^{-2}		2^{-3}	
Norm / DoF_σ	L^2	H^1	L^2	H^1	L^2	H^1
5	3.89E−01	5.56E+00	6.83E−01	8.51E+00	7.73E+00	1.95E+02
10	1.11E−01	1.84E+00	1.45E−01	4.14E+00	2.39E−01	9.07E+00
20	2.51E−01	9.07E+00	6.69E−02	4.24E+00	1.57E−02	1.38E+00
40	1.56E−03	9.17E−02	4.00E−04	4.27E−02	3.11E−04	6.21E−02
80	1.47E−05	1.17E−03	2.24E−06	3.88E−04	1.08E−06	3.68E−04
160	3.56E−06	3.17E−04	6.86E−07	1.30E−04	1.88E−07	7.71E−05
320	1.92E−06	1.92E−04	4.23E−07	6.51E−05	1.48E−07	5.81E−05

Table 13
Errors of the space–time LRNN-C^0DG method when $t = 1$ in Example 5.4.

τ, h	2^{-1}		2^{-2}		2^{-3}	
Norm / DoF_σ	L^2	H^1	L^2	H^1	L^2	H^1
5	1.76E+00	6.88E+00	1.80E+00	5.07E+00	2.41E+00	1.65E+01
10	9.84E−02	1.31E+00	1.82E−01	2.06E+00	2.66E−01	1.02E+01
20	1.37E−02	2.27E−01	4.63E−03	4.18E−01	1.36E−02	1.31E+00
40	7.27E−04	2.05E−02	4.11E−04	3.40E−02	1.09E−04	2.05E−02
80	4.60E−06	2.08E−04	5.02E−07	7.10E−05	1.24E−07	3.01E−05
160	4.33E−07	5.72E−05	1.21E−07	2.13E−05	4.29E−08	1.09E−05
320	4.05E−07	2.14E−05	8.26E−08	1.26E−05	1.68E−07	2.30E−05

Table 14
Errors of the space–time LRNN-C^1DG method when $t = 1$ in Example 5.4.

τ, h	2^{-1}		2^{-2}		2^{-3}	
Norm / DoF_σ	L^2	H^1	L^2	H^1	L^2	H^1
5	3.24E+00	1.82E+01	1.53E+00	7.10E+00	3.15E+00	7.55E+00
10	1.05E−01	1.12E+00	9.06E−01	7.35E+00	3.05E+00	1.65E+01
20	7.57E−02	1.48E+00	3.78E−02	9.87E−01	6.21E−02	2.79E+00
40	2.61E−03	6.34E−02	9.61E−04	4.74E−02	1.37E−03	1.32E−01
80	2.89E−05	1.33E−03	5.43E−06	4.47E−04	6.13E−06	9.69E−04
160	2.25E−07	2.53E−05	2.09E−07	5.50E−05	8.10E−08	1.90E−05
320	1.65E−07	1.13E−05	1.44E−07	1.06E−05	9.31E−08	3.31E−05

Furthermore, we show distributions of the exact solution, numerical solution, and absolute errors of the LRNN-DG method for $k = 1, 3, 5$ in Fig. 7. The size of the subdomain is $h = 1/2$, the degrees of freedom in each subdomain is $\mathrm{DoF}_K = 160$, and other parameters are the same as above.

Example 5.4 (*1-D Heat Equation*)**.** Consider the heat Eq. (4.1) with $\Omega = [0, 1]$, $I = [0, 1]$, and the constant $\lambda = 0.001$. We employ the manufactured exact solution

$$u = -e^{\cos(\pi x + 3\pi) + t^2}.$$

Let us consider the space–time LRNN with DG methods for solving the heat equation. Tables 12, 13, and 14 show the numerical errors measured in the L^2 norm and the H^1 seminorm at $t = 1$ for different numbers of degrees of freedom. The domain $I \times \Omega$ is partitioned into non-overlapping subdomains $\sigma = I_i \times K$, where I_i is a time interval, K is a space interval, and both have the same size h. In these methods, 14 quadrature points are used in each direction.

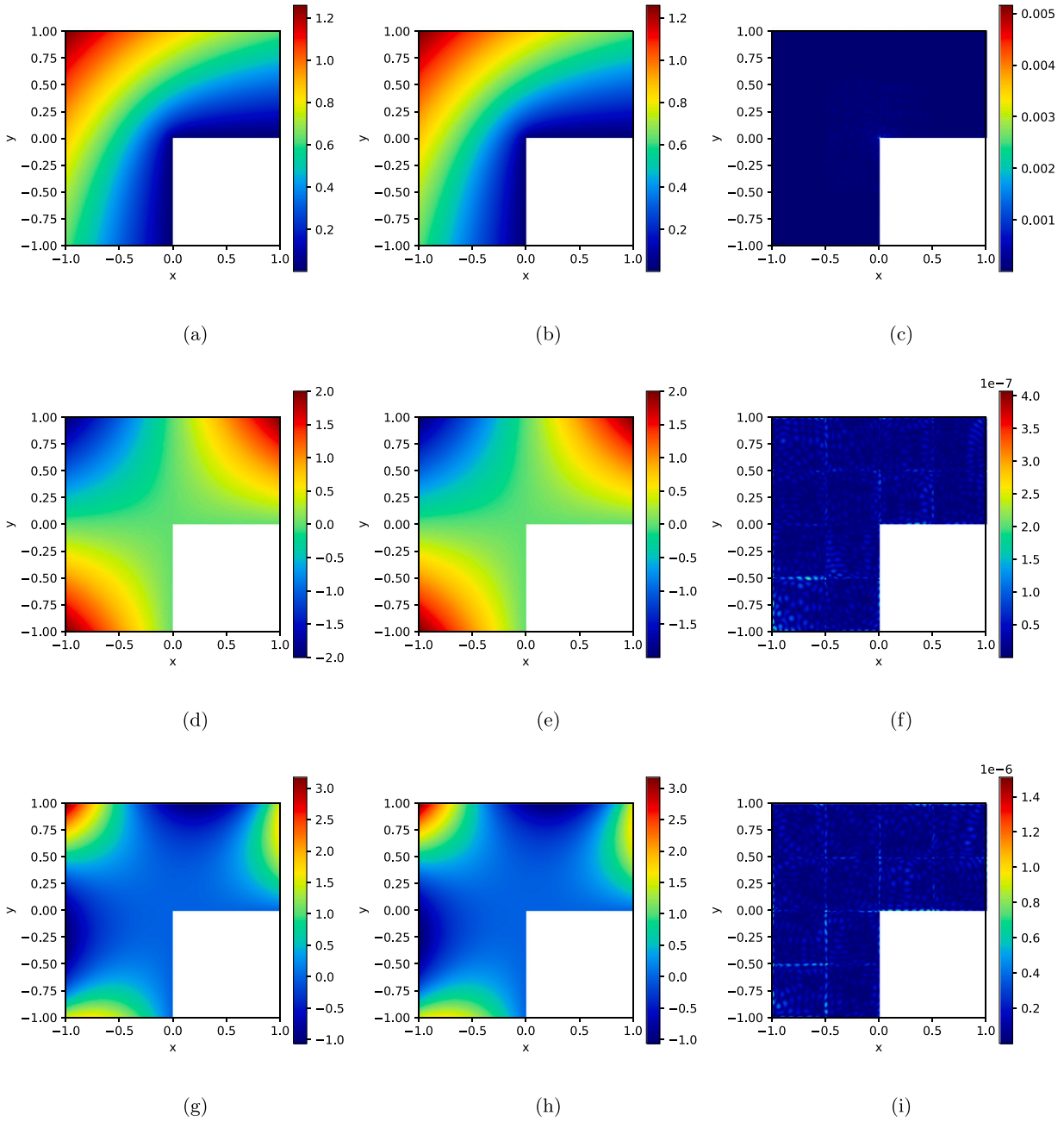


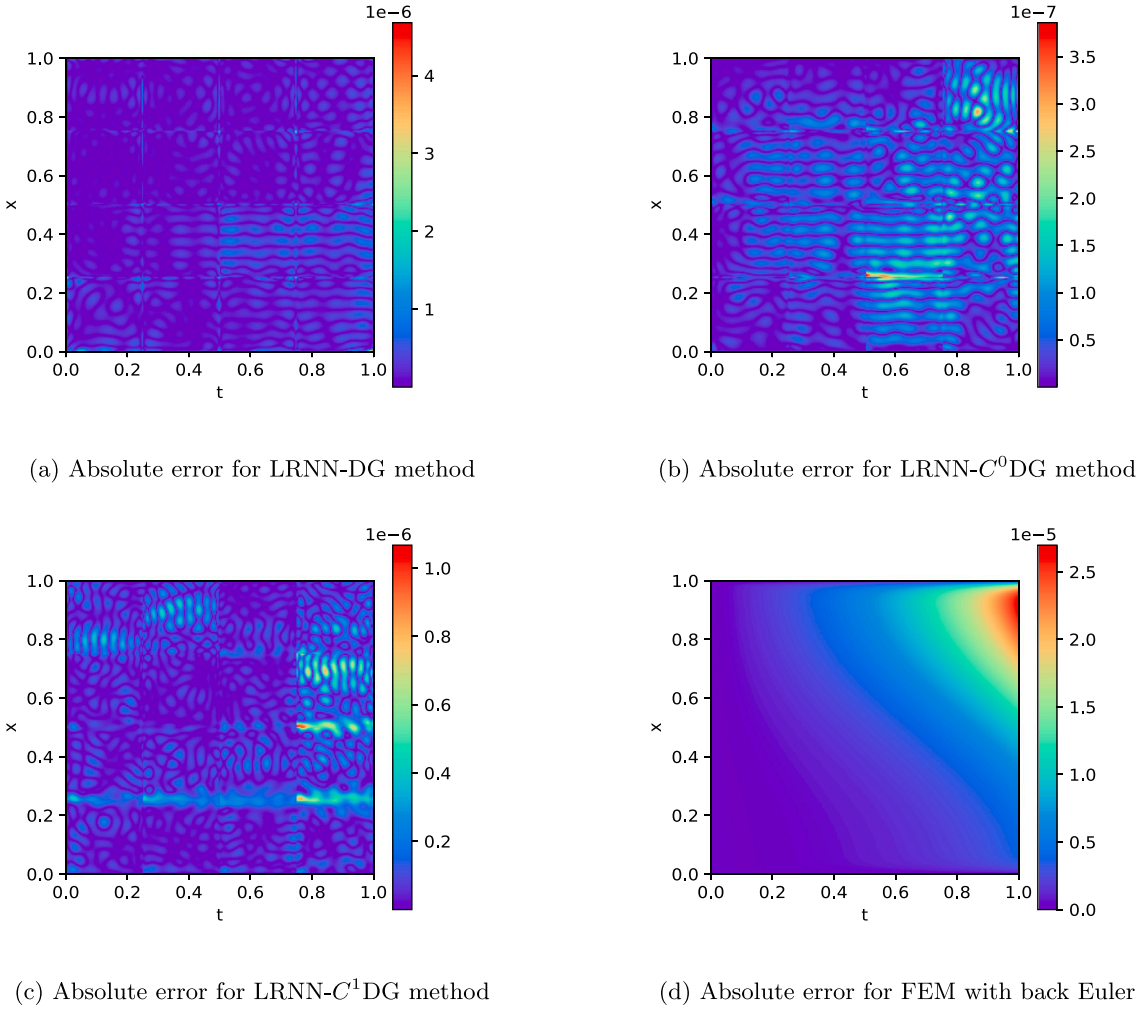
Fig. 7. The exact solution (a, d, g), the numerical solution (b, e, h) of the LRNN-DG method and the point-wise absolute error of the LRNN-DG method (c, f, i) to the Poisson equation. a–c: $k = 1$ (non-smooth), d–f: $k = 3$ (smooth) and g–i: $k = 5$ (non-smooth) in Example 5.3.

Table 12 presents the L^2 and H^1 errors of the space–time LRNN-DG method at $t = 1$, in terms of the number of degrees of freedom per element and the element size. For these tests, we have set the penalty parameter $\eta_e = 4$, and the weight/bias coefficients in the hidden layer of each local network are initialized as uniform random values within the range $[-0.9, 0.9]$.

Table 13 reports the errors of the space–time LRNN-C^0DG method at $t = 1$. In this case, we use 12 collection points on each edge, and the parameter w_0 is set to 1.

Table 14 presents the errors of the space–time LRNN-C^1DG method at $t = 1$. We use 14 collection points on each edge and choose the parameter w_0 to be 1.2.

For comparison, we also solved this problem using P_2 finite element for spatial discretization and the backward Euler scheme for time stepping. The numerical errors for $t = 1$ are presented in Table 15, where h denotes the mesh size and Δt denotes the time



(a) Absolute error for LRNN-DG method

(b) Absolute error for LRNN-C^0DG method

(c) Absolute error for LRNN-C^1DG method

(d) Absolute error for FEM with back Euler

Fig. 8. Absolute errors computed by proposed methods and FEM in Example 5.4.

Table 15
Errors of the P_2 FEM with back Euler when $t = 1$ in Example 5.4.

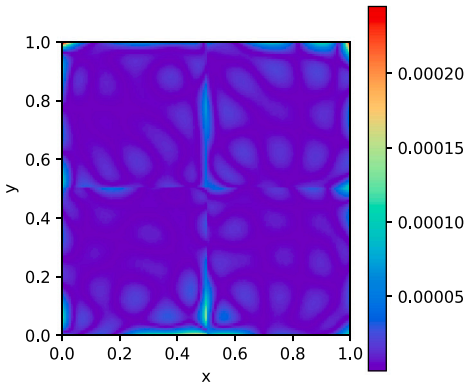
Norm \ h_2, Δt	2^{-5}, 2^{-10}	2^{-6}, 2^{-12}	2^{-7}, 2^{-14}	2^{-8}, 2^{-16}	2^{-9}, 2^{-18}
L^2	3.82E−03	9.56E−04	2.39E−04	5.97E−05	1.49E−05
H^1	3.84E−02	9.65E−03	2.41E−03	6.04E−04	1.51E−04

step. By comparing the results in this table with those of our proposed methods, we observe that the space–time LRNN-DG methods can achieve more accurate numerical solutions.

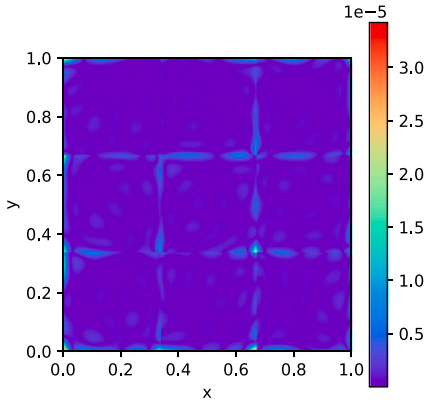
Fig. 8 shows the distribution of point-wise absolute errors in the spatial–temporal domain obtained using the proposed methods and the traditional FEM programmed with FEniCS. In the space–time LRNN with DG methods, the size of each element is $h = 2^{-2}$ and the number of degrees of freedom in each element is $\mathrm{DoF}_\sigma = 320$. In the P_2 FEM with the backward Euler scheme, the size of the mesh is $h_2 = 2^{-9}$ and the size of the time step is $\Delta t = 2^{-18}$. We can see that the absolute error of the LRNN-DG method is larger but smoother, while the absolute error of the LRNN-C^0DG and LRNN-C^1DG methods are smaller but have stronger variations. Due to time marching, error accumulation over time is evident in the result of FEM, whereas there is little or essentially no error accumulation for the proposed space–time LRNN with DG methods.

Example 5.5 (*2-D Heat Equation*)**.** We consider a two-dimensional heat Eq. (4.1) with spatial domain $\Omega = [0, 1]\times[0, 1]$ and temporal interval $I = (0, 5)$. The exact solution for this test is given by

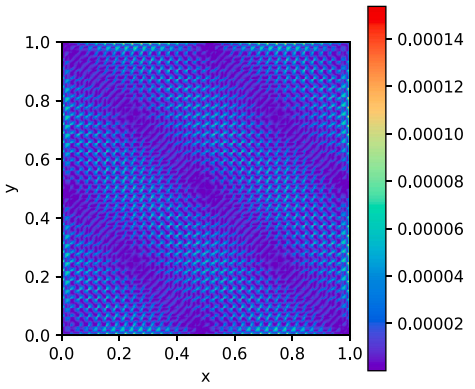
$$u = e^{-0.2t}\sin(2\pi x)\sin(2\pi y).$$



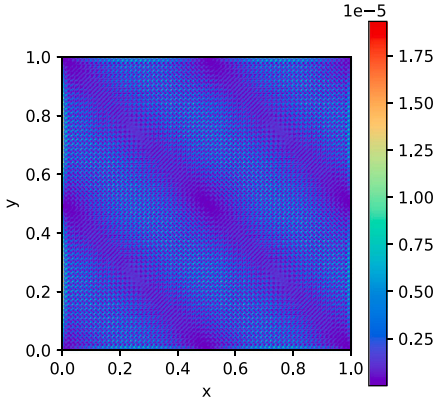
(a) Pointwise absolute error for LRNN-DG with $h = 2^{-1}$, $\tau = 5 \times 2^{-1}$ and $\mathrm{DoF}_\sigma = 640$. L^2 error is 1.48E-05, total computation time is 22 seconds.



(b) Pointwise absolute error for LRNN-DG with $h = 3^{-1}$, $\tau = 5 \times 3^{-1}$ and $\mathrm{DoF}_\sigma = 640$. L^2 error is 1.66E-06, total computation time is 338 seconds.



(c) Pointwise absolute error for traditional DG with $h_3 = 2^{-5}$, $\Delta t = 5 \times 2^{-6}$. L^2 error is 1.61E-05, total computation time is 69 seconds.



(d) Pointwise absolute error for traditional DG with $h_3 = 2^{-6}$, $\Delta t = 5 \times 2^{-6}$. L^2 error is 1.94E-06, total computation time is 656 seconds.

Fig. 9. Distributions of the pointwise absolute error at $t = 5$ and the computation time obtained by the LRNN-DG method and the traditional DG method in Example 5.5.

Table 16 presents the L^2 and H^1 errors of the space–time LRNN-DG method at $t = 5$, in terms of the number of degrees of freedom in each element and the element size. In this numerical experiment, we set the penalty parameter $\eta_e = 3$, the parameter $w_0 = 0.49$, and the number of quadrature points is 9 for each direction. The space–time LRNN-DG method has very good performance in this example as well. Note that the space–time LRNN-C^0DG and LRNN-C^1DG schemes exhibit similar performance to the LRNN-DG method, and so their results are not included here.

Fig. 9 compares the point-wise absolute errors at $t = 5$ and the total computation time obtained by the space–time LRNN-DG method and the traditional DG method (together with the Crank–Nicolson scheme). In the DG method, we use the IPDG scheme with P_2 elements applied on uniform triangular meshes with a mesh size h_3, and the Crank–Nicolson scheme is employed with a time step Δt. The DG implementation is based on the FEniCS library (version 2019.1.0), and the current LRNN-DG method is implemented based on PyTorch 1.12.1. To collect the computation time, all these programs for Example 5.5 have been run on the same CPU. Comparing Figs. 9(a) with 9(b) and 9(c) with 9(d), we observe that LRNN-DG achieves a better accuracy with less computation time. Note that the FEniCS library implements a number of techniques to boost performance and is highly optimized for its (traditional) DG implementation. In contrast, the LRNN-DG is based on our current implementation, which lacks those optimizations in FEniCS (and is available to the traditional DG). But still, the current method shows a very competitive performance.

Table 16
Errors of the space–time LRNN-DG method when $t = 5$ in Example 5.5.

τ, h	5×2^{-1}, 2^{-1}		5×3^{-1}, 3^{-1}		5×4^{-1}, 4^{-1}	
Norm / DoF_σ	L^2	H^1	L^2	H^1	L^2	H^1
20	2.60E−01	3.37E+00	3.96E−01	9.21E+00	2.41E−01	6.70E+00
40	1.16E−01	2.51E+00	3.12E−01	1.01E+01	1.16E−01	4.80E+00
80	2.98E−01	9.36E+00	1.11E−02	5.31E−01	2.04E−03	1.25E−01
160	6.22E−04	2.91E−02	7.34E−05	4.83E−03	1.91E−05	1.69E−03
320	6.42E−05	3.59E−03	3.93E−06	3.24E−04	1.39E−06	1.44E−04
640	1.48E−05	8.61E−04	1.66E−06	1.43E−04	6.07E−07	7.07E−05

6. Summary

The local randomized neural networks with discontinuous Galerkin formulations offer a new approach to solving partial differential equations. By decomposing the domain, we use LRNNs to approximate the solution on each subdomain and apply the IPDG scheme to couple these LRNNs together. The weights of output layers are obtained by the least-squares method. With appropriate assumptions, we prove the convergence of these methods. Additionally, we propose space–time LRNN-DG methods to solve the heat equation, which offer several advantages: (i) achieving better accuracy than FEM or the usual DG method with fewer degrees of freedom; (ii) LRNN-C^0DG and LRNN-C^1DG methods are penalty parameter-free compared to DG methods; (iii) the space–time LRNN-DG methods can solve time-dependent problems more precisely and efficiently.

We are confident that the proposed methods have significant potential for solving partial differential equations. However, there are still several aspects of these methods that require further investigation. In this paper, we have only considered linear partial differential equations. An immediate question is the following. Can one extend the methods and the analysis to nonlinear partial differential equations? This problem is non-trivial and currently under investigation, and it will be addressed in a future publication. Our numerical examples suggest that the errors of these methods plateau when the number of degrees of freedom reaches a certain threshold. Is it possible to design other neural networks (e.g., deep neural networks) to avoid this issue? Can we leverage parallel processing to improve their efficiency? How can we incorporate mesh adaptation to enhance their performance for complex problems? Additionally, deriving error estimates for the proposed methods is another crucial area for future work.

CRediT authorship contribution statement

Jingbo Sun: Methodology, Software, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization. **Suchuan Dong:** Methodology, Resources, Writing – review & editing, Supervision, Funding acquisition. **Fei Wang:** Conceptualization, Methodology, Resources, Writing – original draft, Writing – review & editing, Supervision, Project administration, Funding acquisition.

Data availability

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

Acknowledgments

The authors are grateful to Professor Zongben Xu for his valuable suggestions and discussions, and to the anonymous referee for the insightful comments and feedback that improved the paper.

References

[1] 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.
[2] J. Sirignano, K. Spiliopoulos, DGM: A deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364.
[3] Z. Fang, A high-efficient hybrid physics-informed neural networks based on convolutional neural network, IEEE Trans. Neural Netw. Learn. Syst. (2021) 1–13.
[4] W.T. Leung, G. Lin, Z. Zhang, NH-PINN: Neural homogenization-based physics-informed neural network for multiscale problems, J. Comput. Phys. 470 (2022) 111539.
[5] L. Lyu, Z. Zhang, M. Chen, J. Chen, MIM: A deep mixed residual method for solving high-order partial differential equations, J. Comput. Phys. 452 (2020) 110930.
[6] G. Pang, M. D'Elia, M. Parks, G.E. Karniadakis, nPINNs: nonlocal physics-informed neural networks for a parametrized nonlocal universal Laplacian operator, algorithms and applications, J. Comput. Phys. 422 (2020) 109760.
[7] G. Pang, L. Lu, G.E. Karniadakis, fPINNs: Fractional physics-informed neural networks, SIAM J. Sci. Comput. 41 (2019) A2603–A2626.
[8] A.A. Ramabathiran, P. Ramachandran, SPINN: Sparse, physics-based, and interpretable neural networks for PDEs, J. Comput. Phys. 445 (2021) 110600.
[9] L. Lu, X. Meng, Z. Mao, G.E. Karniadakis, DeepXDE: a deep learning library for solving differential equations, SIAM Rev. 63 (2021) 208–228.
[10] D. Zhang, L. Guo, G.E. Karniadakis, Learning in modal space: Solving time-dependent stochastic PDEs using physics-informed neural networks, SIAM J. Sci. Comput. 42 (2020) A639–A665.

[11] M.M. Almajid, M.O. Abu-Al-Saud, Prediction of porous media fluid flow using physics informed neural networks, J. Pet. Sci. Eng. 208 (2022) 109205.
[12] C. Irrgang, N. Boers, M. Sonnewald, et al., Towards neural earth system modeling by integrating artificial intelligence in earth system science, Nat. Mach. Intell. 3 (2021) 667–674.
[13] X. Jin, S. Cai, H. Li, G.E. Karniadakis, NSFnets (Navier–Stokes flow nets): Physics-informed neural networks for the incompressible Navier–Stokes equations, J. Comput. Phys. 426 (2021) 109951.
[14] X. Jiang, D. Wang, Q. Fan, et al., Physics-informed neural network for nonlinear dynamics in fiber optics, Laser Photonics Rev. 16 (2022) 2100483.
[15] M. Raissi, A. Yazdani, G.E. Karniadakis, Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations, Science 367 (2020) 1026–1030.
[16] 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.
[17] Y. Liao, P. Ming, Deep Nitsche method: Deep Ritz method with essential boundary conditions, Commun. Comput. Phys. 29 (2021) 1365–1384.
[18] Y. Zang, G. Bao, X. Ye, H. Zhou, Weak adversarial networks for high-dimensional partial dierential equations, J. Comput. Phys. 411 (2020) 109409.
[19] H. Sheng, C. Yang, PFNN: A penalty-free neural network method for solving a class of second-order boundary-value problems on complex geometries, J. Comput. Phys. 428 (2021) 110085.
[20] L. Zhang, J. Han, H. Wang, et al., Deep potential molecular dynamics: a scalable model with the accuracy of quantum mechanics, Phys. Rev. Lett. 120 (2018) 143001.
[21] B. Igelnik, Y.H. Pao, Stochastic choice of basis functions in adaptive function approximation and the functional-link net, IEEE Trans. Neural Netw. 6 (1995) 1320–1329.
[22] B. Igelnik, Y.H. Pao, S.R. LeClair, C.Y. Shen, The ensemble approach to neural-network learning and generalization, IEEE Trans. Neural Netw. 10 (1999) 19–30.
[23] Y.H. Pao, Y. Takefuji, Functional-link net computing: theory, system architecture, and functionalities, Computer 25 (1992) 76–79.
[24] Y.H. Pao, G.H. Park, D.J. Sobajic, Learning and generalization characteristics of the random vector functional-link net, Neurocomputing 6 (1994) 163–180.
[25] G.B. Huang, Q.Y. Zhu, C.K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (2006) 489–501.
[26] B. Frénay, M. Verleysen, Using SVMs with randomised feature spaces: an extreme learning approach, in: ESANN 2010.
[27] G.B. Huang, H. Zhou, X. Ding, R. Zhang, Extreme learning machine for regression and multiclass classification, IEEE Trans. Syst. Man Cybern. B 42 (2011) 513–529.
[28] G.B. Huang, X. Ding, H. Zhou, Optimization method based extreme learning machine for classification, Neurocomputing 74 (2010) 155–163.
[29] Q. Liu, Q. He, Z. Shi, Extreme support vector machine classifier, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, Berlin, Heidelberg, 2008.
[30] Y. Miche, A. Sorjamaa, A. Lendasse, OP-ELM: theory, experiments and a toolbox, in: Int. Conf. Artif. Neural Networks, Springer, Berlin, Heidelberg, 2008, pp. 145–154.
[31] E. Parviainen, J. Riihimäki, Y. Miche, A. Lendasse, Interpreting extreme learning machine as an approximation to an infinite neural network, in: KDIR, 2010, pp. 65–73.
[32] H.J. Rong, G.B. Huang, Y.S. Ong, Extreme learning machine for multi-categories classification applications, in: IEEE Int. Jt. Conf. Neural Networks, 2008, pp. 1709–1713.
[33] S. Balasundaram, Kapil, Application of error minimized extreme learning machine for simultaneous learning of a function and its derivatives, Neurocomputing 74 (2011) 2511–2519.
[34] V. Dwivedi, B. Srinivasan, Physics informed extreme learning machine (PIELM) - a rapid method for the numerical solution of partial differential equations, Neurocomputing 391 (2020) 96–118.
[35] H. Sun, M. Hou, Y. Yang, et al., Solving partial differential equations based on Bernstein neural network and extreme learning machine algorithm, Neural Process. Lett. 50 (2019) 1153–1172.
[36] Y. Yang, M. Hou, J. Luo, A novel improved extreme learning machine algorithm in solving ordinary differential equations by Legendre neural network methods, Adv. Differential Equations 2018 (2018) 1–24.
[37] F. Calabro, G. Fabiani, C. Siettos, Extreme learning machine collocation for the numerical solution of elliptic PDEs with sharp gradients, Comput. Methods Appl. Mech. Engrg. 387 (2021) 114188.
[38] X. Liu, S. Lin, J. Fang, Z. Xu, Is extreme learning machine feasible? A theoretical assessment (part 1), IEEE Trans. Neural Netw. Learn. Syst. 26 (2014) 7–20.
[39] 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.
[40] Y. Shang, F. Wang, J. Sun, Randomized neural network with Petrov–Galerkin methods for solving linear and nonlinear partial differential equations, Commun. Nonlinear Sci. Numer. Simul. 127 (2023) 107518.
[41] D.N. Arnold, F. Brezzi, B. Cockburn, L.D. Marini, Unified analysis of discontinuous Galerkin methods for elliptic problems, SIAM J. Numer. Anal. 39 (2002) 1749–1779.
[42] T. Mathew, Domain Decomposition Methods for the Numerical Solution of Partial Differential Equations, Springer Science & Business Media, 2008.
[43] I. Gühring, M. Raslan, Approximation rates for neural networks with encodable weights in smoothness spaces, Neural Netw. 134 (2021) 107–130.
[44] Y. Jiao, Y. Lai, Y. Lo, et al., Error analysis of deep Ritz methods for elliptic equations, Anal. Appl. (2023) 1–31.
[45] S. Dong, J. Yang, On computing the hyperparameter of extreme learning machines: Algorithm and application to computational PDEs, and comparison with classical and high-order finite elements, J. Comput. Phys. 463 (2022) 111290.
[46] H.P. Langtangen, A. Logg, Solving PDEs in Python - The FEniCS Tutorial I, in: Simula SpringerBriefs on Computing, Springer International Publishing, 2016.
[47] N. Ni, S. Dong, Numerical computation of partial differential equations by hidden-layer concatenated extreme learning machine, J. Sci. Comput. 95 (2023) 35.