Commun. Comput. Phys.
doi: 10.4208/cicp.OA-2024-0183

Vol. **39**, No. 1, pp. 147-184
January 2026

Improved Randomized Neural Network Methods with Boundary Processing for Solving Elliptic Equations

Huifang Zhou[1] and Zhiqiang Sheng[2,3,*]

[1] *School of Mathematics, Jilin University, Changchun 130012, P.R. China.*
[2] *Laboratory of Computational Physics, Institute of Applied Physics and Computational Mathematics, Beijing 100088, P.R. China.*
[3] *HEDPS, Center for Applied Physics and Technology, and College of Engineering, Peking University, Beijing 100871, P.R. China.*

Received 4 August 2024; Accepted (in revised version) 20 April 2025

Abstract. We present two improved randomized neural network methods, namely the RNN-Scaling and RNN-Boundary-Processing (RNN-BP) methods, for solving elliptic equations such as the Poisson equation and the biharmonic equation. The RNN-Scaling method modifies the optimization objective by increasing the weight of boundary equations, resulting in a more accurate approximation. We propose the boundary processing techniques for the rectangular domain that enforce the RNN method to satisfy the non-homogeneous Dirichlet and clamped boundary conditions exactly. We further prove that the RNN-BP method is exact for solutions with specific forms and validate it numerically. Numerical experiments demonstrate that the RNN-BP method is the most accurate among the three methods, with the error reduced by up to 6 orders of magnitude for some tests.

AMS subject classifications: 65M08, 35R05

Key words: Randomized neural network, elliptic equations, boundary conditions, scaling method.

1 Introduction

Elliptic partial differential equations (PDEs) model the steady-state conditions in various physical phenomena, including electrostatics, gravitational fields, elasticity, phase-field models, and image processing [1, 10, 30]. For instance, the Poisson equation characterizes the distribution of a scalar field based on boundary conditions and interior sources. In contrast, the biharmonic equation is employed to model phenomena such as the deflection of elastic plates and the flow of incompressible, inviscid fluids. Solving these

*Corresponding author. *Email addresses:* sheng_zhiqiang@iapcm.ac.cn (Z. Sheng), 13614405274@163.com (H. Zhou)

equations is essential for understanding and predicting system behavior in various applications. Traditional numerical methods for solving elliptic equations, such as finite difference methods [2,3,21,32], finite element methods [6,19,20,22,23,34,35], finite volume methods [14,26,27] and spectral methods [4,7,18] have been well studied and widely used. However, these methods often require careful discretization to obtain numerical solutions with high accuracy. Moreover, they may face challenges in handling mesh generation on complex domains and boundary conditions.

In recent years, deep neural network (DNN) methods have been greatly developed in various fields, such as image recognition, natural language processing, and scientific computing. One area where DNN has shown promise is in solving PDEs, including elliptic equations. The DNN-based method transforms the process of solving PDEs into optimization problems and utilizes gradient backpropagation to adjust the network parameters and minimize the residual error of the PDEs. Several effective DNN-based methods include the Physics-Informed Neural Networks (PINNs) [24], the deep Galerkin method [28], the deep Ritz method (DRM) [9], and the deep mixed residual method [17], among others [8,33]. The main difference between these methods lies in the construction of the loss function.

PINNs offer a promising approach for solving various types of PDEs. However, they still have limitations. One major limitation is the relatively low accuracy of the solutions [11], the absolute error rarely goes below the level of 10^{-3} to 10^{-4}. Accuracy at such levels is less than satisfactory for scientific computing, and in some cases, they may fail to converge. Another limitation is that PINNs require high computational cost and training time, which makes them less practical for large-scale or complex problems. PINNs require substantial resources to integrate the PDEs into the training process, especially for the problems involving high-dimensional PDEs or those requiring fine spatial and temporal resolutions.

RNN has recently attracted increasing attention for its application in solving partial differential equations. The weights and biases of the RNN method are randomly generated and fixed, and do not need to be trained. The optimization problem of PINNs is usually a complicated nonlinear optimization problem, requiring a great number of training steps. For the RNN method, the resulting optimization problem is a least-squares problem, which can be solved without training steps.

For deep neural networks, the exact imposition of boundary and initial conditions is crucial for the training speed and accuracy of the model, since it may accelerate the convergence of the training process and improve overall accuracy. For instance, the inexact enforcement of boundary and initial conditions severely affects the convergence and accuracy of PINN-based methods [29]. Recently, many methods have been developed for the exact imposition of Dirichlet and Neumann boundary conditions, which leads to more efficient and accurate training. The main approach is to divide the numerical approximation into two parts: a deterministic function satisfying the boundary condition and a trainable function with the homogeneous condition. This idea was first proposed by Lagaris et al. in [12,13]. The exact enforcement of boundary conditions is applied in

the deep Galerkin method and deep Ritz method for elliptic problems in [5]. The deep mixed residual method, employed in [16, 17] for solving PDEs, satisfies the Neumann boundary condition exactly by transforming the boundary condition into a homogeneous Dirichlet boundary. In [15], the authors propose a gradient-assisted PINNs for solving nonlinear biharmonic equations, introducing gradient auxiliary functions to transform the clamped or simply supported boundary conditions into Dirichlet boundary conditions and then constructing composite functions to satisfy these Dirichlet boundary conditions exactly. However, introducing the gradient-based auxiliary function or additional neural networks into a model leads to an increase in computation and may introduce additional errors. In [8], the authors use the universal approximation property of DNN and specifically designed periodic layers to ensure that the DNN's solution satisfies the specified periodic boundary conditions, including both C^∞ and C^k conditions. The PINNs method proposed in [29] exactly satisfies the Dirichlet, Neumann, and Robin boundary conditions on complex geometries. The main idea is to utilize R-functions and mean value potential fields to construct approximate distance functions, and to use transfinite interpolation to obtain approximations. A penalty-free neural network method is developed to solve second-order boundary-value problems on complex geometries in [25] by using two neural networks to satisfy essential boundary conditions, and introducing a length factor function to decouple the networks. The boundary-dependent PINNs in [31] solve PDEs with complex boundary conditions. The neural network utilizes the radial basis functions to construct trial functions that satisfy boundary conditions automatically, thus avoiding the need for manual trial function design when dealing with complex boundary conditions.

In this work, we present two improved RNN methods for solving elliptic equations, specifically the Poisson and biharmonic equations. Motivated by the observation that the error of the RNN method is concentrated around the boundary, we develop two methods to effectively reduce the boundary error. The first method, called the RNN-Scaling method, adjusts the optimization problem, resulting in a modified least-squares equation. The second improved RNN method, called the RNN-BP method, introduces interpolation techniques to enforce exact inhomogeneous Dirichlet or clamped boundary conditions for the rectangular domains. We conduct extensive numerical experiments to compare the accuracy and computation time of the standard RNN method with those of the improved RNN methods, varying the number of collocation points and the width of the last hidden layer. The numerical results demonstrate the effectiveness of both improved methods. Additionally, we compare the accuracy and computation times of the three RNN methods with those of PINNs and DRM. The results show that the errors and computation times of the RNN methods are several orders of magnitude smaller than those of PINNs and DRM.

The main contributions of this paper are summarized as follows:

- The RNN-BP method significantly reduces the errors of the RNN method by enforcing exact inhomogeneous Dirichlet or clamped boundary conditions. Specifically,

the RNN-BP method directly deals with the clamped boundary condition without introducing gradient auxiliary variables. Consequently, the optimization problem avoids introducing constraints related to gradient relationships, thereby potentially reducing additional errors.

- The RNN-BP method is proved to be exact for the solutions of form $u(x,y)=f_1(x)p_1(y)+f_2(y)p_2(x)$. For the Poisson equation, p_1 and p_2 are polynomials of degree no higher than 1, and f_1,f_2 are functions in $C(\Omega)$. For the biharmonic equation, p_1 and p_2 are polynomials of degree no higher than 3, and f_1,f_2 are functions in $C^1(\Omega)$.
- The RNN-Scaling method increases the weight of boundary equations in the optimization problem, resulting in a more accurate approximation without taking more collocation points.

The remainder of this paper is organized as follows. Section 2 and Section 3 describe the RNN methods for solving the Poisson and biharmonic equations, respectively. Section 4 presents numerical examples to illustrate the effectiveness of the RNN-Scaling and RNN-BP methods. Finally, Section 5 concludes the paper.

2 The improved RNN methods for the Poisson equation

In this section, we focus on the Poisson equation with Dirichlet boundary condition on a two-dimensional bounded domain $\Omega\subset\mathbb{R}^2$ with boundary $\partial\Omega$:

$$\begin{cases} -\Delta u(\mathbf{x})=f(\mathbf{x}), & \text{in } \Omega, \\ u(\mathbf{x})=g(\mathbf{x}), & \text{on } \partial\Omega. \end{cases} \tag{2.1}$$

The source term $f(\mathbf{x})$ and the boundary condition $g(\mathbf{x})$ are given functions.

2.1 Randomized neural networks

The randomized neural network utilizes a fully connected architecture. Let L denote the number of hidden layers, M denote the number of neurons in the last hidden layer, and $\phi_j(\mathbf{x})$ denote the output of the j-th neuron in the last hidden layer, where $1\le j\le M$. The fully connected neural network is expressed as

$$\hat{u}(\mathbf{x})=\sum_{j=1}^{M}\omega_j\phi_j(\mathbf{x})=\omega\Phi(\mathbf{x}),\quad \mathbf{x}\in\Omega,$$

where $\Phi(\mathbf{x})=(\phi_1(\mathbf{x}),\cdots,\phi_M(\mathbf{x}))=\sigma(W_L\cdot\sigma(\cdots\sigma(W_2\cdot\sigma(W_1\cdot\mathbf{x}+b_1)+b_2)\cdots)+b_L)$, σ denotes the activation function, and $\omega=(\omega_1,\cdots,\omega_M)^{\mathrm{T}}$, $W_k\in\mathbb{R}^{n_k\times n_{k-1}}$ and $b_k\in\mathbb{R}^{n_k}$ are the weight

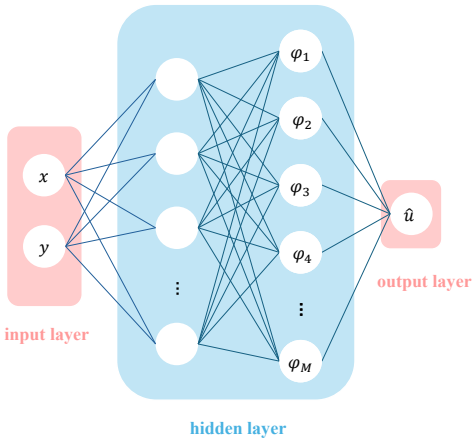


Figure 1: The architecture of the RNN.

matrices and bias vectors, respectively, and n_k denotes the number of neurons in the k-th hidden layer. We employ two methods to generate weights and biases: the default initialization method in PyTorch and uniform random initialization within the range $[-R_m, R_m]$.

2.2 RNN method

First, we select collocation points, which are divided into two types: interior points and boundary points. The interior collocation points consist of N_f points in Ω, while the boundary collocation points consist of N_b points on $\partial\Omega$. The selection of collocation points is not unique; they can be random or uniform. In this paper, we use uniformly distributed collocation points.

The basic idea of the RNN is that the weights and biases of the hidden layers are randomly generated and remain fixed. Thus, we only need to solve a least-squares problem and do not need to train the network. The system of linear algebraic equations is

$$\begin{aligned} -\sum_{j=1}^{M}\omega_j\Delta\varphi_j(\mathbf{x}_f^i) &= f(\mathbf{x}_f^i), \quad i=1,\cdots,N_f, \\ \sum_{j=1}^{M}\omega_j\varphi_j(\mathbf{x}_b^i) &= g(\mathbf{x}_b^i), \quad i=1,\cdots,N_b. \end{aligned} \tag{2.2}$$

Solving this system of equations yields ω, which consequently obtains the solution $\hat{u}(\mathbf{x})$.

We apply the RNN method to solve the Poisson equation (2.1) with the exact solution $u=\sin(2\pi x)\sin(2\pi y)$. The absolute error of the RNN method is shown in Fig. 2. It is

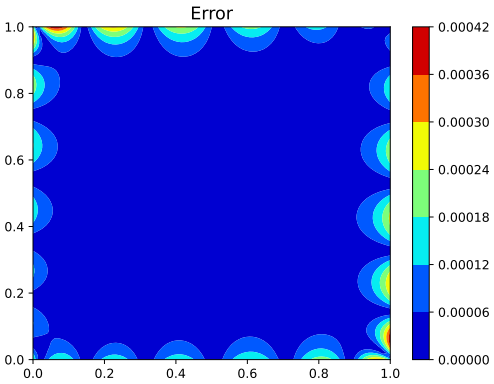


Figure 2: The absolute error of the RNN method.

observed that the error of the RNN method concentrates around the boundary $\partial\Omega$. To improve the accuracy of the RNN method, it is necessary to approximate more accurately around the boundary.

2.3 RNN-Scaling method

To achieve a more accurate approximation around the boundary, a straightforward approach is to increase N_b, which requires placing more collocation points on $\partial\Omega$. However, this may increase the computational cost without significant improvement in accuracy. Therefore, we slightly modify the algebraic equations (2.2) to increase the weight of the boundary equations.

We illustrate the RNN-Scaling method with an example on a square domain. Let the number of collocation points on both the interior and boundary in both x-direction and y-direction be the same, denoted as N. It is obvious that $N_f = N^2$ and $N_b = 4N$. The corresponding equations for the RNN-Scaling method are modified as follows:

$$\begin{aligned} \frac{1}{N^2}\sum_{j=1}^{M}\omega_j L\varphi_j(\mathbf{x}_f^i) &= \frac{1}{N^2}f(\mathbf{x}_f^i), \quad i=1,\cdots,N_f, \\ \sum_{j=1}^{M}\omega_j\varphi_j(\mathbf{x}_b^i) &= g(\mathbf{x}_b^i), \quad i=1,\cdots,N_b. \end{aligned} \tag{2.3}$$

Solving the system of equations (2.3) yields the solution $\hat{u}(\mathbf{x})$.

We present the absolute error of the RNN-Scaling method in Fig. 3 and observe that the error of the RNN-Scaling method is much reduced around the boundary $\partial\Omega$. Consequently, the total relative L^2 error is also reduced by about 3 orders of magnitude.

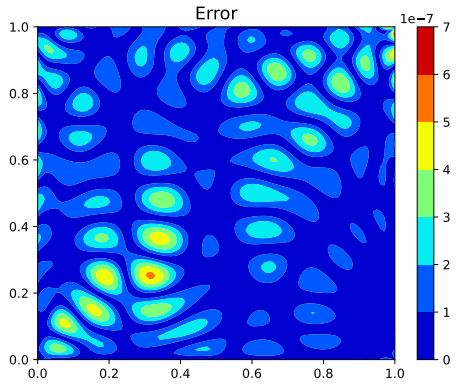


Figure 3: The absolute error of the RNN-Scaling method.

Remark 2.1. The weight coefficient $\frac{1}{N^2}$ is chosen based on both intuitive and numerical considerations. In the classical methods, such as finite element methods and finite difference methods, the scale of the equations for interior unknowns is typically larger than that for boundary unknowns by a factor of approximately $\frac{1}{N^2}$. For example, in the finite difference method, assuming $h=\frac{1}{N}$ is the mesh size, we have

$$-\Delta\varphi_{i,j}\approx\frac{1}{h^2}\left(4\varphi_{i,j}-\varphi_{i-1,j}-\varphi_{i+1,j}-\varphi_{i,j-1}-\varphi_{i,j+1}\right).$$

Thus, the coefficients for interior points are larger than those for boundary points. The standard RNN method could be seen as an extension of the classical method and may inherit this property. However, since the standard RNN method is based on the least-squares method, the interior points are much more "important" than the boundary points in the numerical scheme. This leads to increased errors at the boundary points. To balance this impact, we multiply the interior coefficients by $\frac{1}{N^2}$, analogous to h^2. We test several numerical examples and find that $\frac{1}{N^2}$ appears to be an appropriate choice for the Poisson equation.

Remark 2.2. The idea of the RNN-Scaling method could be extended to general domains. For instance, on a unit circular domain if we define $h=\frac{1}{N}$ as the "mesh size", representing the average distance between collocation points. Then, the interior collocation points can be uniformly distributed within the unit disk with a spacing of h, and the boundary collocation points can consist of $[2\pi N]$ uniformly distributed points on the circle. Fig. 8(a) shows the distribution of collocation points on the unit circle. The weight coefficient is similarly chosen as $\frac{1}{N^2}$. In this case, the total relative L^2 error is also reduced by one to two orders of magnitude.

2.4 RNN-BP method

As shown in Fig. 3, although the error of the RNN-Scaling method significantly decreased, it still exists on the boundary $\partial\Omega$. In this subsection, we introduce a boundary processing technique to enforce exact Dirichlet boundary conditions. This technique was initially proposed in [12]. The advantage of this boundary processing technique is that the boundary conditions are imposed on the numerical solution across the entire boundary, rather than only at collocation points. We now outline the construction of the RNN-BP method. For simplicity, we consider the unit square domain $\Omega=(0,1)\times(0,1)$ and denote $\mathbf{x}$ as (x,y). It should be noted that the technique can be easily extended to rectangular domains.

The numerical solution of the RNN-BP method is constructed as follows:

$$\hat{u}(x,y)=B(x,y)\sum_{j=1}^{M}\omega_j\varphi_j(x,y)+g_D(x,y), \tag{2.4}$$

where B and g_D should satisfy

$$\begin{aligned} B(x,y)&=0, \quad \text{on } \partial\Omega,\\ B(x,y)&\neq 0, \quad \text{in } \Omega,\\ g_D(x,y)&=g(x,y), \quad \text{on } \partial\Omega. \end{aligned}$$

A straightforward choice is $B(x,y)=x(1-x)y(1-y)$ for the Poisson equation. The network architecture is illustrated in Fig. 4.

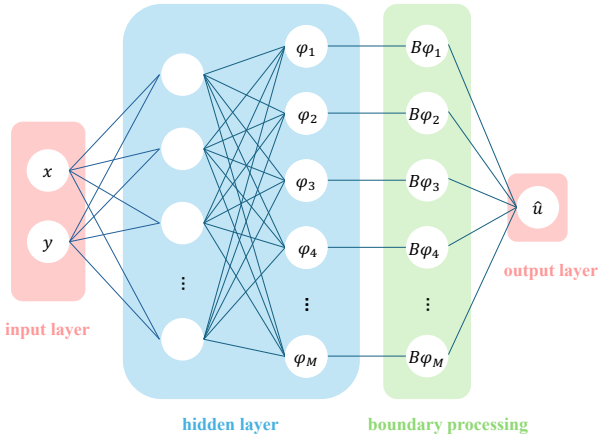


Figure 4: The architecture of the RNN-BP.

Next, we describe the construction of $g_D(x,y)$. Let $g_D(x,y)=s(x,y)+P(x,y)$, where $P(x,y)$ is constructed to satisfy the relations at the four corners:

$$P(x_i,y_j)=u(x_i,y_j),$$

where $i,j=0,1$, with $x_0=y_0=0$ and $x_1=y_1=1$. Rewrite the Dirichlet boundary condition as

$$\begin{aligned}
&u(0,y)=t_1(y), \quad 0\le y\le 1,\\
&u(1,y)=t_2(y), \quad 0\le y\le 1,\\
&u(x,0)=t_3(x), \quad 0\le x\le 1,\\
&u(x,1)=t_4(x), \quad 0\le x\le 1.
\end{aligned}$$

Using the one-dimensional two-point Lagrange interpolation functions $l_0(x)=1-x$ and $l_1(x)=x$, $P(x,y)$ can be expressed as a bilinear Lagrange interpolation function:

$$P(x,y)=\begin{bmatrix} l_0(x)\\ l_1(x)\end{bmatrix}^{\mathrm{T}}\begin{bmatrix} u(x_0,y_0) & u(x_0,y_1)\\ u(x_1,y_0) & u(x_1,y_1)\end{bmatrix}\begin{bmatrix} l_0(y)\\ l_1(y)\end{bmatrix},$$

or equivalently,

$$P(x,y)=\begin{bmatrix} l_0(x)\\ l_1(x)\end{bmatrix}^{\mathrm{T}}\begin{bmatrix} t_1(y_0) & t_1(y_1)\\ t_2(y_0) & t_2(y_1)\end{bmatrix}\begin{bmatrix} l_0(y)\\ l_1(y)\end{bmatrix}.$$

Denote $\tilde{u}(x,y)\triangleq u(x,y)-P(x,y)$. We then construct $s(x,y)$ to satisfy the Dirichlet condition of $\tilde{u}$ exactly, i.e.,

$$s(x,y)=\tilde{u}(x,y), \quad \text{on } \partial\Omega.$$

The boundary conditions of $\tilde{u}(x,y)$ are denoted by $\tilde{t}_i$, which satisfy the following relations:

$$\begin{aligned}
&\tilde{t}_1(y)=t_1(y)-P(0,y), \quad 0\le y\le 1,\\
&\tilde{t}_2(y)=t_2(y)-P(1,y), \quad 0\le y\le 1,\\
&\tilde{t}_3(x)=t_3(x)-P(x,0), \quad 0\le x\le 1,\\
&\tilde{t}_4(x)=t_4(x)-P(x,1), \quad 0\le x\le 1.
\end{aligned}$$

The function $s(x,y)$ is defined as

$$s(x,y)=l_0(x)\tilde{t}_1(y)+l_1(x)\tilde{t}_2(y)+l_0(y)\tilde{t}_3(x)+l_1(y)\tilde{t}_4(x).$$

After constructing $s(x,y)$ and $P(x,y)$, the corresponding equations for the RNN-BP method are derived as follows:

$$-\sum_{j=1}^{M}\omega_j\Delta(B\varphi_j)(\mathbf{x}_f^i)=f(\mathbf{x}_f^i)+\Delta s(\mathbf{x}_f^i)+\Delta P(\mathbf{x}_f^i), \quad i=1,\cdots,N_f. \tag{2.5}$$

Theorem 2.1. *The numerical solution* (2.4) *of the RNN-BP method satisfies the Dirichlet boundary condition exactly.*

Proof. Our aim is to prove that $\hat{u}(x,y)-P(x,y)=\tilde{u}(x,y)$ on $\partial\Omega$, which is equivalent to showing that $\sum_{i=1}^{M}\omega_i x(1-x)y(1-y)\varphi_i(x,y)+s(x,y)=\tilde{u}(x,y)$ on $\partial\Omega$. Note that $\sum_{i=1}^{M}\omega_i x(1-x)y(1-y)\varphi_i(x,y)$ satisfies the homogeneous Dirichlet boundary condition. Therefore, we only need to prove that $s(0,y)=\tilde{t}_1(y)$. Thus, we have

$$\begin{aligned}s(0,y)&=l_0(0)\tilde{t}_1(y)+l_1(0)\tilde{t}_2(y)+l_0(y)\tilde{t}_3(0)+l_1(y)\tilde{t}_4(0)\\&=\tilde{t}_1(y),\end{aligned}$$

where we use the fact that $\tilde{t}_i(0)=\tilde{t}_i(1)=0$ for $i=1,\cdots,4$.

The conditions on the other boundaries can be proved in a similar manner. This completes the proof. □

Theorem 2.2. *Assuming that the solution to the least-squares problem* (2.5) *is uniquely solvable. Then the RNN-BP method is exact for solutions of the form*

$$f_1(x)p_1(y)+f_2(y)p_2(x), \tag{2.6}$$

where $f_1,f_2\in C(\Omega)$, and p_1 and p_2 are polynomials of degree no higher than 1.

Proof. For simplicity, we only prove the case where $u(x,y)=f_1(x)p_1(y)$. The proof for the case $u(x,y)=f_1(x)p_1(y)+f_2(y)p_2(x)$ is similar and thus omitted.

The first step is to prove that $u(x,y)=s(x,y)$ under the assumption that $u(x,y)=0$ at the four corners of $\partial\Omega$. It is obvious that $P(x,y)\equiv 0$ from the above assumption. We analyze the four terms of $s(x,y)=l_0(x)t_1(y)+l_1(x)t_2(y)+l_0(y)t_3(x)+l_1(y)t_4(x)$ sequentially. For the term $t_1(y)=f_1(0)p_1(y)\in\mathbb{P}_1$, it follows that $t_1(0)=t_1(1)=0$. Consequently, $t_1(y)\equiv 0$ on $[0,1]$. Similarly, $t_2(y)\equiv 0$ in $[0,1]$. Hence we obtain

$$\begin{aligned}s(x,y)&=l_0(x)t_1(y)+l_1(x)t_2(y)+l_0(y)t_3(x)+l_1(y)t_4(x)\\&=l_0(y)f_1(x)p_1(0)+l_1(y)f_1(x)p_1(1)\\&=f_1(x)(l_0(y)p_1(0)+l_1(y)p_1(1))\\&=f_1(x)p_1(y).\end{aligned}$$

The second step is to prove that $u(x,y)=s(x,y)+P(x,y)$ for $u(x,y)=f_1(x)p_1(y)$. Let $Q(x)$ be a first-order polynomial satisfying $Q(0)=f_1(0)$ and $Q(1)=f_1(1)$. From the definition of $P(x,y)$ and the fact that $Q(x)p_1(y)=P(x,y)$ at the four corners of $\partial\Omega$, it follows that $Q(x)p_1(y)\equiv P(x,y)$. Thus, for any $(x,y)\in\Omega$, we have

$$\begin{aligned}u(x,y)-P(x,y)&=(f_1(x)-Q(x))p_1(y)\\&=s(x,y).\end{aligned} \tag{2.7}$$

The final equation follows from the conclusion of the first step, since $f_1(x)-Q(x)=0$ at the four corners of $\partial\Omega$. Substituting (2.7) into (2.5) yields $\sum_{j=1}^{M}\omega_j\Delta(B(\mathbf{x}_f^i)\varphi_j(\mathbf{x}_f^i))=0$ for

$i=1,\cdots,N_f$. Clearly, $\{\omega_j\}_{j=1}^M=\mathbf{0}$ is a solution to the least-squares problem. Hence, we obtain

$$\hat{u}(x,y)=u(x,y),\quad \text{in } \Omega,$$

which completes the proof. □

3 The improved RNN methods for the biharmonic equation

In this section, we focus on the biharmonic equation with clamped boundary conditions on a bounded domain with boundary $\partial\Omega$:

$$\begin{cases}\Delta^2 u(\mathbf{x})=f(\mathbf{x}), & \text{in } \Omega,\\ u(\mathbf{x})=g_1(\mathbf{x}), & \text{on } \partial\Omega,\\ \dfrac{\partial u}{\partial \mathbf{n}}(\mathbf{x})=g_2(\mathbf{x}), & \text{on } \partial\Omega.\end{cases}\tag{3.1}$$

The source term $f(\mathbf{x})$ and the boundary conditions $g_1(\mathbf{x})$ and $g_2(\mathbf{x})$ are given functions, and $\mathbf{n}$ denotes the unit outward normal vector to $\partial\Omega$.

3.1 RNN method and RNN-Scaling method

For the biharmonic equation, the equations of the RNN method are

$$\begin{aligned}\sum_{j=1}^M \omega_j \Delta^2\varphi_j(\mathbf{x}_f^i)&=f(\mathbf{x}_f^i),\quad i=1,\cdots,N_f,\\ \sum_{j=1}^M \omega_j \varphi_j(\mathbf{x}_b^i)&=g_1(\mathbf{x}_b^i),\quad i=1,\cdots,N_b,\\ \sum_{j=1}^M \omega_j \frac{\partial\varphi_j}{\partial\mathbf{n}}(\mathbf{x}_b^i)&=g_2(\mathbf{x}_b^i),\quad i=1,\cdots,N_b,\end{aligned}$$

where N_f and N_b still represent the numbers of interior and boundary collocation points, respectively. Solving this system of equations yields ω, which is then used to compute the solution $\hat{u}(\mathbf{x})$.

For the biharmonic equation, we also observe that the error of the RNN method is primarily concentrated around the boundary. Therefore, as with the Poisson equation, it is necessary to develop an improved method to reduce the error around the boundary.

We describe the RNN-Scaling method on the square domain. Let the numbers of points on both interior and boundary be the same in the x-direction and y-direction, still denoted as N. Then $N_f=N^2$ and $N_b=4N$. The least-squares problem is then formulated

as follows:

$$\begin{aligned}\frac{1}{N^4}\sum_{j=1}^{M}\omega_j\Delta^2\varphi_j(\mathbf{x}_f^i)&=\frac{1}{N^4}f(\mathbf{x}_f^i),\quad i=1,\cdots,N_f,\\ \sum_{j=1}^{M}\omega_j\varphi_j(\mathbf{x}_b^i)&=g_1(\mathbf{x}_b^i),\quad i=1,\cdots,N_b,\\ \frac{1}{N}\sum_{j=1}^{M}\omega_j\frac{\partial\varphi_j}{\partial\mathbf{n}}(\mathbf{x}_b^i)&=\frac{1}{N}g_2(\mathbf{x}_f^i),\quad i=1,\cdots,N_b.\end{aligned}\tag{3.2}$$

Solving the system (3.2) yields the numerical solution $\hat{u}(\mathbf{x})$.

Remark 3.1. The RNN-Scaling method can also be extended to general domains for the biharmonic equation. For instance, on a unit circular domain, if we take $h=\frac{1}{N}$ as the "mesh size", then the interior collocation points can be chosen as uniformly distributed points within the unit disk at a distance of h, and the boundary collocation points could be $[2\pi N]$ uniformly distributed points on the circle. The weight coefficients are set to $\frac{1}{N^4}$ and $\frac{1}{N}$, respectively. The total relative L^2 error is thereby reduced by 1-2 orders of magnitude.

3.2 RNN-BP method

The error of the RNN-Scaling method is significantly reduced, but a small error still exists on the boundary. Therefore, we enforce the neural network to satisfy the exact clamped boundary condition on the square domain.

The neural network used in the RNN-BP method for the biharmonic equation is constructed as follows:

$$\hat{u}(x,y)=B(x,y)\sum_{i=1}^{M}\omega_i\varphi_i(x,y)+g_D(x,y),\tag{3.3}$$

where $B(x,y)$ and $g_D(x,y)$ should satisfy the following conditions:

$$\begin{aligned}B(x,y)&=0,\quad \text{on } \partial\Omega,\\ \frac{\partial B}{\partial\mathbf{n}}(x,y)&=0,\quad \text{on } \partial\Omega,\\ B(x,y)&\neq 0,\quad \text{in } \Omega,\\ g_D(x,y)&=g_1(x,y),\quad \text{on } \partial\Omega,\\ \frac{\partial g_D}{\partial\mathbf{n}}(x,y)&=g_2(x,y),\quad \text{on } \partial\Omega.\end{aligned}$$

For the biharmonic equation, $B(x,y)$ is defined as $B(x,y)=x^2(1-x)^2y^2(1-y)^2$.

Next, we construct $g_D(x,y)$ as $g_D(x,y)=s(x,y)+P(x,y)$. We define $P(x,y)$ such that it satisfies the following conditions at the four corners:

$$\begin{aligned}
&P(x_i,y_j)=u(x_i,y_j),\\
&P_x(x_i,y_j)=u_x(x_i,y_j),\\
&P_y(x_i,y_j)=u_y(x_i,y_j),\\
&P_{xy}(x_i,y_j)=u_{xy}(x_i,y_j),
\end{aligned}$$

where $i,j=0,1$, and $x_0=y_0=0$, $x_1=y_1=1$. Rewrite the clamped boundary condition (3.1) as follows:

$$\begin{aligned}
&u(0,y)=t_1(y), \quad \frac{\partial u}{\partial x}(0,y)=h_1(y), \quad 0\le y\le 1,\\
&u(1,y)=t_2(y), \quad \frac{\partial u}{\partial x}(1,y)=h_2(y), \quad 0\le y\le 1,\\
&u(x,0)=t_3(x), \quad \frac{\partial u}{\partial y}(x,0)=h_3(x), \quad 0\le x\le 1,\\
&u(x,1)=t_4(x), \quad \frac{\partial u}{\partial y}(x,1)=h_4(x), \quad 0\le x\le 1.
\end{aligned}$$

Recall the two-point Hermite interpolation basis functions, which are defined as follows:

$$\begin{aligned}
&H_i(x)=l_i^2(x)\left(1-2l_i'(x_i)(x-x_i)\right),\\
&G_i(x)=l_i^2(x)(x-x_i),
\end{aligned}$$

for $i=0,1$, where $l_0(x)$ and $l_1(x)$ are the first-order Lagrange interpolation functions. The expression for $P(x,y)$ is given by

$$P(x,y)=\begin{bmatrix} H_0(x)\\ H_1(x)\\ G_0(x)\\ G_1(x)\end{bmatrix}^{\mathrm{T}}\begin{bmatrix} u(x_0,y_0) & u(x_0,y_1) & u_y(x_0,y_0) & u_y(x_0,y_1)\\ u(x_1,y_0) & u(x_1,y_1) & u_y(x_1,y_0) & u_y(x_1,y_1)\\ u_x(x_0,y_0) & u_x(x_0,y_1) & u_{xy}(x_0,y_0) & u_{xy}(x_0,y_1)\\ u_x(x_1,y_0) & u_x(x_1,y_1) & u_{xy}(x_1,y_0) & u_{xy}(x_1,y_1)\end{bmatrix}\begin{bmatrix} H_0(y)\\ H_1(y)\\ G_0(y)\\ G_1(y)\end{bmatrix}.$$

Alternatively, $P(x,y)$ can also be expressed as:

$$P(x,y)=\begin{bmatrix} H_0(x)\\ H_1(x)\\ G_0(x)\\ G_1(x)\end{bmatrix}^{\mathrm{T}}\begin{bmatrix} t_1(y_0) & t_1(y_1) & h_3(x_0) & h_4(x_0)\\ t_2(y_0) & t_2(y_1) & h_3(x_1) & h_4(x_1)\\ h_1(y_0) & h_1(y_1) & h_3'(x_0) & h_4'(x_0)\\ h_2(y_0) & h_2(y_1) & h_3'(x_1) & h_4'(x_1)\end{bmatrix}\begin{bmatrix} H_0(y)\\ H_1(y)\\ G_0(y)\\ G_1(y)\end{bmatrix}.$$

Then, we define $s(x,y)$ such that it satisfies the clamped boundary conditions for $\tilde{u}(x,y)$, where $\tilde{u}(x,y)\triangleq u(x,y)-P(x,y)$.

$$\begin{cases} s(x,y)=\tilde{u}(x,y), & \text{on } \partial\Omega,\\ \dfrac{\partial s}{\partial \mathbf{n}}(x,y)=\dfrac{\partial \tilde{u}}{\partial \mathbf{n}}(x,y), & \text{on } \partial\Omega. \end{cases}$$

The boundary functions of $\tilde{u}(x,y)$, denoted by $\tilde{t}_i$ and $\tilde{h}_i$, satisfy the following conditions:

$$\begin{aligned}
&\tilde{t}_1(y)=t_1(y)-P(0,y), \quad \tilde{h}_1(y)=h_1(y)-\frac{\partial P}{\partial x}(0,y), \quad 0\le y\le 1,\\
&\tilde{t}_2(y)=t_2(y)-P(1,y), \quad \tilde{h}_2(y)=h_2(y)-\frac{\partial P}{\partial x}(1,y), \quad 0\le y\le 1,\\
&\tilde{t}_3(x)=t_3(x)-P(x,0), \quad \tilde{h}_3(x)=h_3(x)-\frac{\partial P}{\partial y}(x,0), \quad 0\le x\le 1,\\
&\tilde{t}_4(x)=t_4(x)-P(x,1), \quad \tilde{h}_4(x)=h_4(x)-\frac{\partial P}{\partial y}(x,1), \quad 0\le x\le 1.
\end{aligned}$$

The function $s(x,y)$ is defined as:

$$\begin{aligned}
s(x,y)=&H_0(x)\tilde{t}_1(y)+H_1(x)\tilde{t}_2(y)+H_0(y)\tilde{t}_3(x)+H_1(y)\tilde{t}_4(x)\\
&+G_0(x)\tilde{h}_1(y)+G_1(x)\tilde{h}_2(y)+G_0(y)\tilde{h}_3(x)+G_1(y)\tilde{h}_4(x).
\end{aligned}$$

Finally, the corresponding equations for the RNN-BP method are

$$\sum_{j=1}^{M}\omega_j\Delta^2(B\varphi_j)(\mathbf{x}_f^i)=f(\mathbf{x}_f^i)-\Delta^2 s(\mathbf{x}_f^i)-\Delta^2 P(\mathbf{x}_f^i), \quad i=1,\cdots,N_f. \tag{3.4}$$

We emphasize that the clamped boundary condition can be exactly imposed. This boundary processing method, which is applicable to rectangular domains, can be used for any PDEs with clamped boundary conditions.

Theorem 3.1. *The numerical solution* (3.3) *of the RNN-BP method satisfies the clamped boundary condition of* (3.1) *exactly.*

Proof. To prove the theorem, it suffices to show that $\hat{u}(x,y)-P(x,y)=\sum_{i=1}^{M}\omega_i B(x,y)\varphi_i(x,y)+s(x,y)$ satisfies the clamped boundary condition of $\tilde{u}(x,y)$. Since $P(x,y)$ is the cubic Hermit interpolation of $u(x,y)$ on $\bar{\Omega}$, we have:

$$\begin{aligned}
&\tilde{t}_i(0)=\tilde{h}_i(0)=\tilde{h}_i'(0)=0,\\
&\tilde{t}_i(1)=\tilde{h}_i(1)=\tilde{h}_i'(1)=0,
\end{aligned} \tag{3.5}$$

for $i=1,\cdots,4$.

It can be verified that $\sum_{i=1}^{M}\omega_i x^2(1-x)^2y^2(1-y)^2\varphi_i(x,y)$ satisfies the homogeneous clamped boundary condition. Therefore, we need to prove that $s(0,y)=\tilde{t}_1(y)$, $\frac{\partial s}{\partial x}(0,y)=$

$\tilde{h}_1(y)$. From (3.5), we have

$$\begin{aligned} s(0,y) =& H_0(0)\tilde{t}_1(y)+H_1(0)\tilde{t}_2(y)+H_0(y)\tilde{t}_3(0)+H_1(y)\tilde{t}_4(0) \\ &+G_0(0)\tilde{h}_1(y)+G_1(0)\tilde{h}_2(y)+G_0(y)\tilde{h}_3(0)+G_1(y)\tilde{h}_4(0) \\ =& \tilde{t}_1(y), \\ \frac{\partial s}{\partial x}(0,y) =& H_0'(0)\tilde{t}_1(y)+H_1'(0)\tilde{t}_2(y)+H_0(y)\tilde{t}_3'(0)+H_1(y)\tilde{t}_4'(0) \\ &+G_0'(0)\tilde{h}_1(y)+G_1'(0)\tilde{h}_2(y)+G_0(y)\tilde{h}_3'(0)+G_1(y)\tilde{h}_4'(0) \\ =& \tilde{h}_1(y). \end{aligned}$$

The clamped conditions on the other boundary can be proved similarly. This completes the proof. □

Theorem 3.2. *Assuming the solution to the least-squares problem* (3.4) *is unique, then the RNN-BP method is exact for solutions of the form*

$$f_1(x)p_1(y)+f_2(y)p_2(x), \tag{3.6}$$

where $f_1,f_2\in C^1(\Omega)$*, and* p_1 *and* p_2 *are polynomials of degree no higher than 3.*

Proof. The proof of this theorem is similar to that of Theorem 2.2; however, for completeness, we provide the proof here. We only prove the case where $u(x,y)=f_1(x)p_1(y)$.

We also divide the proof into two steps. The first step is to prove that $u(x,y)=s(x,y)$ under the assumption that $u(x,y)=0$ and $\frac{\partial u}{\partial \mathbf{n}}(x,y)=0$ at the four corners of $\partial\Omega$. It is obvious that $P(x,y)\equiv 0$ from the above assumption. We analyze the eight terms of $s(x,y)$ one by one. Similar to the proof of Theorem 2.2, it follows that $t_1(y)=t_2(y)=h_1(y)=h_2(y)=0$ on $[0,1]$. Thus, we have

$$\begin{aligned} s(x,y) =& H_0(y)t_3(x)+H_1(y)t_4(x)+G_0(y)h_3(x)+G_1(y)h_4(x) \\ =& f_1(x)(H_0(y)p_1(0)+H_1(y)p_1(1)-G_0(y)p_1'(0)+G_1(y)p_1'(1)) \\ =& f_1(x)p_1(y). \end{aligned}$$

The second step is to prove that $u(x,y)=s(x,y)+P(x,y)$ for $u(x,y)=f_1(x)p_1(y)$. Let $Q(x)$ be a first-order polynomial satisfying $Q(0)=f_1(0)$, $Q(1)=f_1(1)$, $Q(0)=-f_1'(0)$ and $Q(1)=f_1'(1)$. From the definition of $P(x,y)$, it can be inferred that $Q(x)p_1(y)\equiv P(x,y)$ on Ω. This implies that

$$\begin{aligned} u(x,y)-P(x,y) =& (f_1(x)-Q(x))p_1(y) \\ =& s(x,y), \end{aligned} \tag{3.7}$$

for any $(x,y)\in\Omega$, where the final equation in (3.7) follows from the conclusion of the first step. Substituting (3.7) into (3.4) yields that $\sum_{j=1}^{M}\omega_j\Delta(B(\mathbf{x}_f^i)\varphi_j(\mathbf{x}_f^i))=0$ for $i=1,\cdots,N_f$,

which implies that $\{\omega_j\}_{j=1}^M = \mathbf{0}$ is a solution to the least-squares problem. Therefore, we have

$$\hat{u}(x,y) = u(x,y), \quad \text{in } \Omega.$$

This completes the proof. □

4 Numerical experiments

In the numerical experiments, we compare the performance of the three RNN methods using the relative L^2 error

$$\|e\|_{L^2} = \frac{\sqrt{\sum_{i=1}^{N_{test}} |\hat{u}(\mathbf{x}_i) - u(\mathbf{x}_i)|^2}}{\sqrt{\sum_{i=1}^{N_{test}} |u(\mathbf{x}_i)|^2}},$$

where N_{test} denotes the number of uniformly distributed collocation points in $\bar{\Omega}$, and we set $N_{test} = 10^4$ for all examples. Throughout this paper, we use the sine function as the activation function. In all tables, RNN-Scaling is abbreviated as RNN-S for simplicity. We compare the numerical performance of the three RNN methods, PINNs, and DRM. The RNN methods, DRM, and PINNs are implemented in Python, using the PyTorch library. The numerical experiments are performed on a workstation equipped with an Intel Xeon Platinum 8260L CPU @ 2.40 GHz.

4.1 Poisson equation

We begin by testing our methods on the Poisson equation.

4.1.1 Example 1.1

In the first example, the exact solution on $\Omega = (0,1) \times (0,1)$ is given by:

$$u(x,y) = \sin(2\pi x)\sin(2\pi y).$$

The number of points in x and y dimensions is set to be the same, denoted as N. A neural network with two hidden layers is used, containing 100 and M neurons, respectively. We compare the relative L^2 errors of the RNN, RNN-Scaling, and RNN-BP methods for different numbers of collocation points and values of M. Tables 1 and 2 compare the results for the default and uniform random initialization with R_m=1, respectively. Table 3 shows the CPU times of the three RNN methods under default initialization with varying N and M.

From Tables 1-3, we draw the following observations. The errors of all three RNN methods initially decrease and then stabilize under both initialization methods. The RNN-Scaling method consistently produces smaller errors than the RNN method. In

Table 1: Errors of the three RNN methods with default initialization for Example 1.1.

method	N	M	50	100	150	200	250	300	400	500
RNN	8	$\|e\|_{L^2}$	9.44e-1	2.76e-3	1.01e-3	4.82e-4	7.79e-4	5.49e-4	4.49e-4	3.66e-4
	12	$\|e\|_{L^2}$	9.15e-1	3.13e-4	1.11e-6	3.99e-7	1.83e-7	8.14e-7	2.53e-7	5.97e-7
	16	$\|e\|_{L^2}$	9.45e-1	6.63e-4	2.43e-6	2.66e-7	1.35e-7	4.53e-8	2.67e-7	1.16e-7
	24	$\|e\|_{L^2}$	1.04e+0	9.26e-4	3.61e-6	1.33e-6	2.06e-7	1.76e-7	1.46e-7	1.41e-7
	32	$\|e\|_{L^2}$	1.14e+0	1.04e-3	3.99e-6	2.51e-6	2.84e-7	1.87e-7	1.27e-6	2.83e-7
	48	$\|e\|_{L^2}$	1.34e+0	1.20e-3	4.40e-6	2.35e-6	1.13e-6	1.94e-6	1.65e-6	1.08e-6
RNN-S	8	$\|e\|_{L^2}$	2.09e-2	2.76e-3	1.01e-3	4.82e-4	7.79e-4	5.49e-4	3.06e-4	3.66e-4
	12	$\|e\|_{L^2}$	1.75e-2	2.04e-5	4.35e-7	2.18e-7	3.04e-7	3.38e-7	6.22e-7	5.11e-7
	16	$\|e\|_{L^2}$	1.57e-2	1.42e-5	9.53e-8	8.78e-8	1.08e-7	2.19e-8	1.65e-7	2.60e-8
	24	$\|e\|_{L^2}$	1.43e-2	1.31e-5	1.75e-7	3.29e-7	5.43e-8	8.21e-8	2.56e-7	3.32e-8
	32	$\|e\|_{L^2}$	1.41e-2	1.29e-5	3.67e-7	3.42e-7	1.63e-7	8.12e-8	2.43e-7	5.47e-8
	48	$\|e\|_{L^2}$	1.48e-2	1.27e-5	3.31e-6	1.26e-6	1.37e-6	3.31e-6	2.18e-6	2.27e-7
RNN-BP	8	$\|e\|_{L^2}$	3.55e-4	3.42e-4	1.89e-4	2.43e-4	1.01e-4	1.02e-4	1.64e-4	8.70e-5
	12	$\|e\|_{L^2}$	1.09e-4	2.37e-7	7.55e-8	5.59e-8	4.99e-8	7.74e-8	4.37e-8	7.58e-8
	16	$\|e\|_{L^2}$	1.05e-4	3.72e-8	7.54e-10	4.63e-10	6.68e-10	8.87e-11	6.72e-10	3.61e-10
	24	$\|e\|_{L^2}$	1.02e-4	2.30e-8	1.03e-10	1.20e-10	3.86e-10	1.13e-10	1.57e-10	3.86e-11
	32	$\|e\|_{L^2}$	9.89e-5	2.24e-8	1.63e-10	1.29e-10	2.36e-10	2.66e-10	2.74e-10	1.17e-10
	48	$\|e\|_{L^2}$	9.40e-5	2.13e-8	3.53e-10	5.00e-10	1.81e-10	1.37e-10	4.14e-10	4.34e-11

particular, the method exhibits more stable performance, with errors gradually decreasing as N and M increase. The RNN-BP method achieves the smallest errors among the three methods under both initialization conditions, with errors decreasing as N and M increase. The CPU times for the three methods under default initialization are compared in Table 3 and Fig. 6. Fig. 6(a) shows the results for a fixed $M=300$ with varying numbers of points, while Fig. 6(b) shows the results for a fixed $N=48$ with varying M. The CPU times for all three methods increase with the number of points and M. The CPU times for the RNN and RNN-Scaling methods are similar, while the RNN-BP method requires about 2-3 times the computational cost of the other two methods.

Figs. 5(a)-(b) show that the RNN-Scaling and RNN-BP methods consistently achieve smaller L^2 errors than the RNN method, particularly under uniform random initialization with $R_m=1$. This suggests that these two methods are more robust to variations in initialization and may offer better performance in practical applications. Moreover, the performance of all three methods is generally better under uniform random initialization than under the default initialization condition.

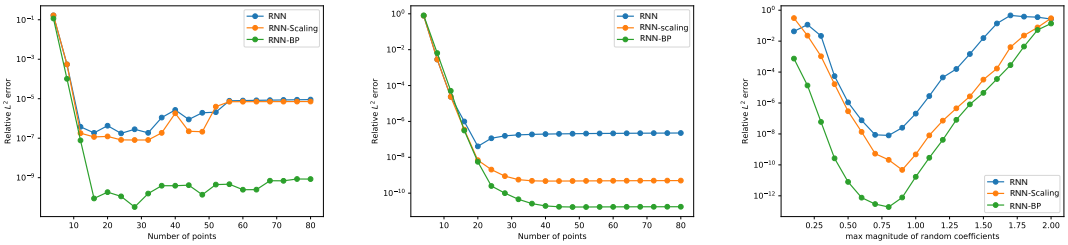
We compare the three methods for varying values of R_m (from 0.1 to 2) with fixed $M=300$ and $N=48$. From Fig. 5(c), it can be seen that as R_m varies, the three methods exhibit a similar trend. Specifically, for R_m in the range of 0.3 to 1.5, the errors of the three methods are relatively small; however, for $R_m>1.5$, the errors increase rapidly. Among the three methods, the RNN-BP method consistently achieves the smallest errors.

Table 2: Errors of the three RNN methods with uniform random initialization ($R_m = 1$) for Example 1.1.

method	N	M	50	100	150	200	250	300	400	500
RNN	8	$\|e\|_{L^2}$	2.19e+0	7.02e-3	4.37e-3	3.77e-3	2.89e-3	2.84e-3	4.05e-3	2.71e-3
	12	$\|e\|_{L^2}$	2.28e+0	5.55e-2	4.95e-4	1.32e-4	3.07e-5	2.36e-5	2.79e-5	2.35e-5
	16	$\|e\|_{L^2}$	2.47e+0	6.08e-2	1.88e-3	3.34e-5	7.44e-7	9.90e-7	1.22e-7	2.34e-7
	24	$\|e\|_{L^2}$	2.73e+0	6.30e-2	2.46e-3	8.14e-5	3.02e-6	1.15e-7	2.86e-10	4.17e-11
	32	$\|e\|_{L^2}$	2.94e+0	6.48e-2	2.73e-3	9.64e-5	3.61e-6	1.78e-7	9.82e-10	3.54e-11
	48	$\|e\|_{L^2}$	3.25e+0	6.83e-2	3.05e-3	1.11e-4	4.24e-6	2.06e-7	1.62e-9	1.24e-10
RNN-S	8	$\|e\|_{L^2}$	9.25e-2	7.02e-3	4.37e-3	3.77e-3	2.89e-3	2.84e-3	4.05e-3	2.71e-3
	12	$\|e\|_{L^2}$	8.07e-2	6.80e-4	4.51e-5	1.32e-4	3.07e-5	2.36e-5	2.79e-5	2.35e-5
	16	$\|e\|_{L^2}$	8.58e-2	5.73e-4	1.61e-5	8.15e-7	5.36e-7	3.49e-7	1.51e-7	2.34e-7
	24	$\|e\|_{L^2}$	9.02e-2	5.07e-4	1.22e-5	3.36e-7	1.87e-8	2.06e-9	1.42e-10	3.48e-11
	32	$\|e\|_{L^2}$	9.27e-2	5.00e-4	1.13e-5	3.01e-7	1.26e-8	5.67e-10	1.20e-11	2.69e-12
	48	$\|e\|_{L^2}$	9.85e-2	5.12e-4	1.08e-5	2.87e-7	1.24e-8	4.77e-10	5.73e-12	8.27e-13
RNN-BP	8	$\|e\|_{L^2}$	1.15e-2	1.01e-2	4.40e-3	3.90e-3	5.34e-3	6.46e-3	4.21e-3	4.61e-3
	12	$\|e\|_{L^2}$	7.21e-3	1.77e-4	4.44e-5	1.05e-4	1.20e-4	5.07e-5	1.26e-4	5.76e-5
	16	$\|e\|_{L^2}$	6.87e-3	5.64e-5	2.17e-6	6.42e-7	1.30e-6	3.11e-7	1.66e-7	5.13e-7
	24	$\|e\|_{L^2}$	6.43e-3	4.06e-5	4.01e-7	1.38e-8	2.18e-9	2.51e-10	2.45e-11	2.12e-11
	32	$\|e\|_{L^2}$	6.18e-3	3.85e-5	3.80e-7	8.44e-9	4.35e-10	4.56e-11	9.71e-13	3.72e-13
	48	$\|e\|_{L^2}$	5.91e-3	3.57e-5	3.82e-7	8.42e-9	3.63e-10	1.69e-11	8.15e-14	2.55e-14

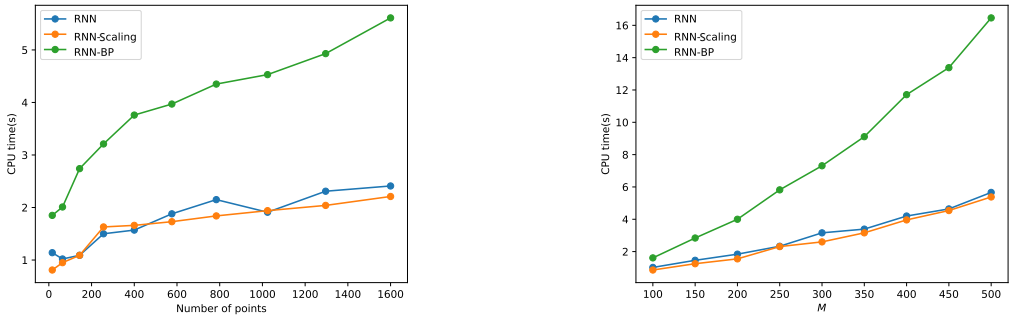
Table 3: The CPU times (seconds) for the three RNN methods with the default initialization for Example 1.1.

method	N	M	50	100	150	200	250	300	400	500
RNN	8	CPU time	0.61	0.54	0.62	0.76	0.85	1.19	1.19	1.50
	12	CPU time	0.40	0.53	0.65	1.09	0.89	1.12	1.54	1.70
	16	CPU time	0.62	0.57	0.73	0.98	1.18	1.23	1.83	2.15
	24	CPU time	0.57	0.76	0.95	1.35	1.58	1.67	2.19	2.85
	32	CPU time	0.56	1.00	1.04	1.38	1.58	1.82	2.81	3.51
	48	CPU time	0.71	0.96	1.42	1.78	2.38	2.77	4.15	5.78
RNN-S	8	CPU time	0.38	0.70	0.59	0.61	0.71	0.89	1.16	1.65
	12	CPU time	0.38	0.49	0.60	0.78	0.88	1.10	1.62	1.69
	16	CPU time	0.43	0.57	0.67	0.81	1.02	1.50	1.65	2.15
	24	CPU time	0.51	0.78	0.91	1.06	1.54	1.55	2.11	2.71
	32	CPU time	0.48	0.72	0.93	1.48	1.48	1.72	2.37	3.58
	48	CPU time	0.59	0.81	1.44	1.60	2.38	2.72	3.96	5.68
RNN-BP	8	CPU time	0.69	0.75	1.04	1.35	1.61	1.89	2.70	4.08
	12	CPU time	0.55	0.83	1.14	1.60	2.00	2.56	3.92	4.74
	16	CPU time	0.62	0.98	1.47	1.92	2.69	3.13	4.06	5.77
	24	CPU time	0.68	1.30	1.84	2.36	3.21	3.60	5.02	7.18
	32	CPU time	0.78	1.35	2.29	3.03	3.72	4.53	6.64	9.65
	48	CPU time	0.89	1.73	3.12	3.88	5.95	8.03	12.61	18.03



(a) (b) (c)

Figure 5: Comparison of relative L^2 errors of the three RNN methods for Example 1.1: (a) Varying numbers of points with the default initialization. (b) Varying numbers of points with uniform random initialization ($R_m=1$). (c) Varying max magnitude of random coefficients.



(a) Varying number of points with the default initialization.

(b) Varying M with the default initialization.

Figure 6: The CPU times of the three RNN methods for Example 1.1.

With the network architecture fixed as $[2,100,200,1]$ and $N=48$, we compare the accuracy and efficiency of the RNN methods with PINNs and DRM. For the Poisson equation, the loss functions for PINNs and DRM, evaluated on uniform points, are defined as:

$$L_{PINNs}=\frac{1}{N_f}\sum_{i=1}^{N_f}\left[\Delta\hat{u}(\mathbf{x}_f^i)+f(\mathbf{x}_f^i)\right]^2+\beta\frac{1}{N_b}\sum_{i=1}^{N_b}\left[\hat{u}(\mathbf{x}_b^i)-g(\mathbf{x}_b^i)\right]^2,$$

$$L_{DRM}=\frac{1}{N_f}\sum_{i=1}^{N_f}\left[\frac{1}{2}\left\|\nabla\hat{u}(\mathbf{x}_f^i)\right\|_2^2-f(\mathbf{x}_f^i)\hat{u}(\mathbf{x}_f^i)\right]+\beta\frac{1}{N_b}\sum_{i=1}^{N_b}\left[\hat{u}(\mathbf{x}_b^i)-g(\mathbf{x}_b^i)\right]^2,$$

respectively. In the PINNs/Adam and DGM/Adam methods, the network is trained for 50,000 epochs, with the learning rate following a cosine decay schedule from 0.001 to 10^{-5}. In the PINNs/L-BFGS and DRM/L-BFGS methods, the network is trained for 500 epochs using a constant learning rate of 1. The results are summarized in Table 4. The L-BFGS optimizer is more efficient than the Adam optimizer for both PINNs and DRM.

Table 4: Comparison between the three RNN methods and PINNs/DRM, in terms of relative L^2 errors and the computation times.

Method	$\|e\|_{L^2}$	Epochs	Computation time (seconds)
PINNs (Adam)	1.69e-4	50,000	833.84
PINNs (L-BFGS)	9.30e-5	500	69.05
DRM (Adam)	3.89e-2	50,000	467.05
DRM (L-BFGS)	4.36e-2	500	26.35
RNN	2.35e-6	0	1.78
RNN-S	1.26e-6	0	1.60
RNN-BP	5.00e-10	0	3.88

The L-BFGS optimizer takes about 1/10 to 1/20 of the computational cost of the Adam optimizer while achieving similar accuracy. However, even using the L-BFGS optimizer, PINNs and DRM achieve accuracies of only 10^{-5} and 10^{-2}, respectively. In contrast, the RNN methods achieve accuracies of at least 10^{-6}, with computational cost under 4 seconds. These results demonstrate the superior accuracy and efficiency of the RNN methods.

4.1.2 Example 1.2

The second numerical example uses the exact solution on $\Omega=(0,2)^2$:

$$u(x,y)=-\left[2\cos\left(\frac{3}{2}\pi x+\frac{2\pi}{5}\right)+\frac{3}{2}\cos\left(3\pi x-\frac{\pi}{5}\right)\right]\left[2\cos\left(\frac{3}{2}\pi y+\frac{2\pi}{5}\right)+\frac{3}{2}\cos\left(3\pi y-\frac{\pi}{5}\right)\right].$$

We compare the three methods by varying the maximum magnitude of the random coefficient, with the network architecture fixed as $[2,100,500,1]$ and $N=96$. The results are presented in Fig. 7(a). It is observed that the optimal value of $R_m\approx 1.2$. The RNN-BP method achieves the smallest error among the three methods, reaching a minimum error of 4.75×10^{-10}. We compare the methods for varying M, with $R_m=1$ and $N=64$ fixed. Fig. 7(b) illustrates that the errors of all three methods decrease as M increases, with the RNN-BP method consistently achieving the smallest errors.

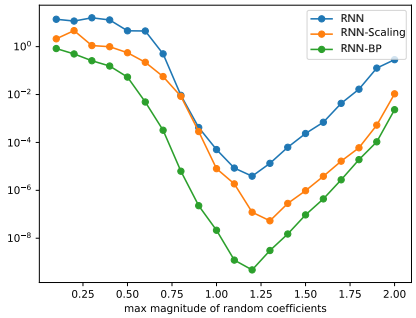
4.1.3 Example 1.3

This example demonstrates that the RNN-BP method is exact for solutions of the form (2.6). We consider the exact solution on $\Omega=(0,1)\times(0,1)$:

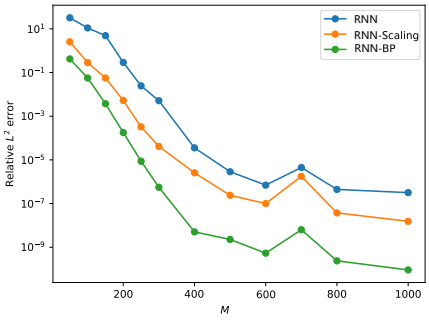
$$u(x,y)=x^{10}+y^{10}+x^k\sin y+y^k\cos x,$$

where k is an integer.

We compare the three methods for different numbers of collocation points with $k=1$ and $k=2$, using a fixed network [2, 100, 300, 1] and random initialization ($R_m=1$). The



(a) Varying max magnitude of random coefficient with uniform random initialization.



(b) Varying M with uniform random initialization ($R_m=1$).

Figure 7: Comparison of relative L^2 errors of the three RNN methods for Example 1.2.

Table 5: Comparison of the three RNN methods with different N for Example 1.3.

k	N	4	12	20	28	36	44	52
1	RNN	8.17e-1	1.41e-4	2.44e-7	6.32e-7	8.23e-7	9.00e-7	9.43e-7
	RNN-S	8.17e-1	1.41e-4	3.55e-8	5.47e-9	3.67e-9	3.56e-9	3.60e-9
	RNN-BP	2.35e-16	2.60e-16	2.57e-16	2.19e-16	2.19e-16	2.19e-16	2.19e-16
2	RNN	1.05e+0	1.70e-4	2.92e-7	7.58e-7	9.87e-7	1.08e-6	1.13e-6
	RNN-S	1.05e+0	1.70e-4	4.25e-8	6.56e-9	4.39e-9	4.26e-9	4.30e-9
	RNN-BP	2.64e-2	1.15e-6	1.39e-11	8.14e-13	1.62e-13	9.64e-14	9.06e-14

results are presented in Table 5. For the RNN-BP method, the relative L^2 errors reach machine precision when $k=1$. This verifies that the RNN-BP method is exact for the solution of the form (2.6). The errors decrease rapidly as the number of collocation points increase when $k=2$, with the smallest error about 10^{-14}. The relative L^2 errors of the RNN-Scaling method are at most three orders of magnitude smaller than those of the RNN method.

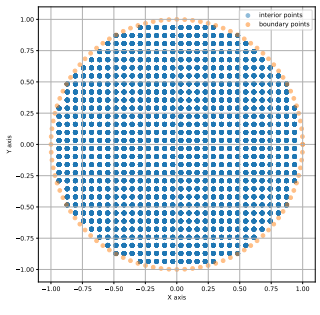
4.1.4 Example 1.4

This example considers the exact solution

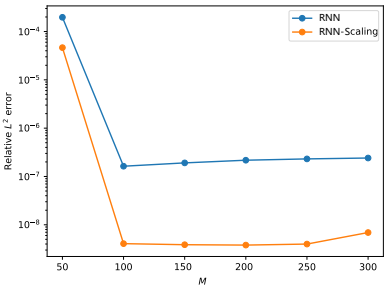
$$u(x,y)=x^4+y^4$$

on the unit circle centered at the origin $(0,0)$.

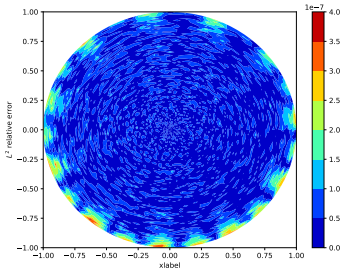
We fix $N=64$ and vary M to compare the RNN and RNN-Scaling method. The network architecture is $[2,100,100,M,1]$, and the default initialization method is employed. Fig. 8(a) illustrates the distribution of uniform collocation points for $N=32$. Fig. 8(b) shows the comparison results, indicating that the RNN-Scaling method achieves the



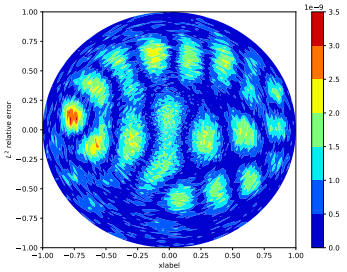
(a) The uniform collocation points on the unit circle.



(b) Varying M with the default initialization.



(c) The absolute error of the RNN method.



(d) The absolute error of the RNN-Scaling method.

Figure 8: The numerical results for Example 1.4.

smallest error (about 10^{-9}), typically reducing errors by two orders of magnitude compared to the RNN method. Figs. 8(c)-(d) display the absolute errors for both methods with $N=64$ and $M=200$. It can be observed that the errors of the RNN method are concentrated near the boundary, while the errors of the RNN-Scaling method are concentrated in the interior.

From Examples 1.1-1.4, we offer the following summary for solving the Poisson equation. The RNN-BP method consistently produces the smallest errors among the three methods. Specifically, for relatively large M and N, the RNN-BP method achieves errors that are, on average, 4 orders of magnitude smaller than those of the RNN method in Example 1.1 and 3 orders of magnitude smaller in Example 1.2. Moreover, under both initialization methods, the RNN-BP method achieves significantly smaller errors than the other two methods, demonstrating its robustness in practical applications.

On both the unit circular and square domains, the RNN-Scaling method outperforms the RNN method in terms of accuracy. Specifically, on the unit circular domain, the RNN-Scaling method achieves relative L^2 errors that are, on average, 2 orders of magnitude smaller than those of the RNN method. On the square domain, under the default initialization method, the RNN-Scaling method achieves errors that are, on average, 2 orders of

magnitude smaller than those of the RNN method when M is small; however, the difference between the two methods becomes less significant as M increases. Under uniform random initialization, the RNN-Scaling method achieves errors that are, on average, 2 orders of magnitude smaller than those of the RNN method.

4.2 Biharmonic equation

Next, we test our methods for the biharmonic equation.

4.2.1 Example 2.1

In the first example, the exact solution on $\Omega=(0,1)\times(0,1)$ is given by:

$$u(x,y)=\sin(\pi x)\sin(\pi y).$$

A neural network with two hidden layers is used, containing 100 and M neurons, respectively. We compare the relative L^2 errors of the three methods for varying numbers of collocation points and values of M. Tables 6 and 7 present the comparison results with the default and uniform random initialization (R_m=1), respectively. Tables 6 and 7 show that the errors of all three methods decrease significantly as M and N increase. The RNN-BP method achieves the best performance, followed by the RNN-Scaling method and then the RNN method. Specifically, under the default initialization, the RNN-Scaling method reduces errors by 1 to 4 orders of magnitude, while the RNN-BP method reduces

Table 6: Comparison of the three RNN methods with the default initialization for Example 2.1.

method	N	M	50	100	150	200	250	300
RNN	8	$\|e\|_{L^2}$	4.73e-2	1.65e-5	1.03e-8	4.13e-9	3.65e-9	4.40e-9
	12	$\|e\|_{L^2}$	5.48e-2	1.50e-4	3.62e-8	1.69e-10	5.20e-11	1.54e-10
	16	$\|e\|_{L^2}$	5.87e-2	1.78e-4	6.17e-8	1.20e-9	5.79e-10	1.20e-10
	24	$\|e\|_{L^2}$	6.48e-2	1.93e-4	7.35e-8	1.75e-9	1.92e-9	3.65e-10
	32	$\|e\|_{L^2}$	7.03e-2	2.01e-4	8.83e-8	1.98e-9	8.14e-10	1.96e-9
RNN-S	8	$\|e\|_{L^2}$	1.78e-4	1.26e-7	8.94e-9	4.00e-9	3.45e-9	4.29e-9
	12	$\|e\|_{L^2}$	7.22e-4	5.68e-8	3.19e-11	1.64e-11	7.06e-12	7.13e-12
	16	$\|e\|_{L^2}$	9.56e-4	3.78e-8	2.97e-11	5.76e-11	1.57e-11	6.29e-12
	24	$\|e\|_{L^2}$	7.57e-4	5.05e-8	5.16e-10	2.80e-10	1.29e-10	8.97e-11
	32	$\|e\|_{L^2}$	8.52e-4	6.57e-8	1.25e-9	1.39e-9	6.19e-10	8.57e-10
RNN-BP	8	$\|e\|_{L^2}$	3.37e-9	5.79e-10	2.80e-10	1.54e-10	1.05e-10	1.85e-10
	12	$\|e\|_{L^2}$	1.49e-8	8.37e-13	2.49e-14	8.27e-14	5.12e-14	3.00e-14
	16	$\|e\|_{L^2}$	1.20e-8	4.65e-13	9.27e-16	5.32e-16	2.67e-16	3.35e-16
	24	$\|e\|_{L^2}$	9.33e-9	5.44e-13	7.17e-16	1.15e-15	1.28e-16	2.37e-16
	32	$\|e\|_{L^2}$	8.18e-9	5.09e-13	6.73e-16	4.13e-16	1.87e-16	4.73e-16

Table 7: Comparison between the three RNN methods with uniform random initialization ($R_m = 1$) for Example 2.1.

method	N	M	50	100	150	200	250	300
RNN	8	$\|e\|_{L^2}$	4.96e+0	4.05e-3	6.38e-4	6.74e-4	3.30e-4	1.05e-3
	12	$\|e\|_{L^2}$	5.09e+0	4.27e-1	4.24e-2	4.19e-5	3.04e-7	4.32e-7
	16	$\|e\|_{L^2}$	5.05e+0	5.62e-1	4.28e-2	5.66e-3	1.28e-4	2.03e-6
	24	$\|e\|_{L^2}$	5.00e+0	6.33e-1	4.00e-2	7.98e-3	3.18e-4	3.30e-5
	32	$\|e\|_{L^2}$	4.98e+0	6.58e-1	4.08e-2	9.14e-3	3.94e-4	4.60e-5
RNN-S	8	$\|e\|_{L^2}$	8.15e-2	9.32e-4	6.38e-4	6.74e-4	3.30e-4	1.05e-3
	12	$\|e\|_{L^2}$	3.96e-2	3.00e-3	4.53e-5	2.79e-6	3.04e-7	4.32e-7
	16	$\|e\|_{L^2}$	4.36e-2	3.72e-3	3.21e-5	1.20e-6	2.97e-8	3.45e-9
	24	$\|e\|_{L^2}$	2.73e-2	3.50e-3	3.23e-5	7.27e-7	1.21e-8	7.52e-10
	32	$\|e\|_{L^2}$	1.01e-1	2.97e-3	5.86e-5	9.14e-7	2.13e-8	7.95e-10
RNN-BP	8	$\|e\|_{L^2}$	2.01e-4	2.35e-4	6.07e-5	3.25e-4	4.15e-4	3.09e-4
	12	$\|e\|_{L^2}$	6.49e-4	1.93e-6	1.03e-6	9.59e-7	2.86e-7	1.25e-6
	16	$\|e\|_{L^2}$	6.55e-4	1.17e-6	4.42e-8	8.24e-9	3.85e-9	2.34e-9
	24	$\|e\|_{L^2}$	6.32e-4	5.04e-7	6.13e-8	1.26e-9	1.82e-11	1.02e-11
	32	$\|e\|_{L^2}$	6.12e-4	3.85e-7	9.31e-8	1.54e-9	6.55e-11	1.11e-11

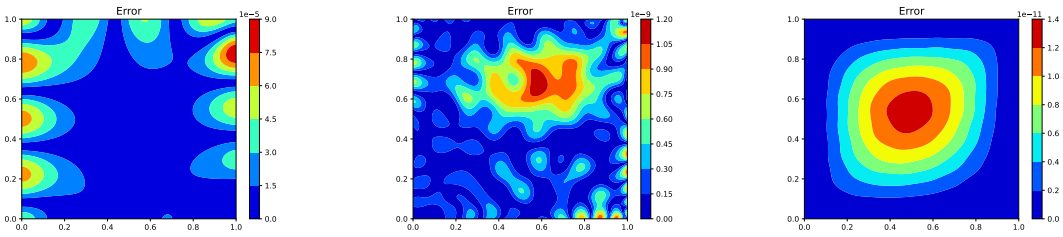
errors by 6 to 9 orders of magnitude. The RNN-BP method achieves errors at machine precision for $M \geq 150$ and $N \geq 16$. Under random initialization, the RNN-Scaling method reduces errors by up to 5 orders of magnitude, while the RNN-BP method reduces errors by up to 7 orders of magnitude. These results demonstrate that the RNN-BP and RNN-Scaling methods offer significant advantages over the RNN method. Figs. 9(a)-(c) show the error distributions for the three methods with $M = 300$ and $N = 32$. The errors of the RNN are predominantly concentrated near the boundary, while the RNN-Scaling method effectively reduces these boundary errors. Furthermore, the RNN-BP method enforces exact boundary conditions, achieving the highest accuracy.

Table 8 presents the computation times of the three methods. It can be observed that the CPU times of the RNN and RNN-Scaling methods are similar, while the RNN-BP method takes approximately twice as long as the other two methods.

We fix the network architecture as [2,100,300,1] and compare the performance of the three methods for varying numbers of collocation points. Figs. 10(a)-(b) present the comparison results with the default and random initialization ($R_m = 1$), respectively. It can be seen that, regardless of the initialization method, the RNN-BP method consistently achieves smaller errors than the other two methods. With the default initialization, the RNN-Scaling method performs slightly better than the RNN method. With the uniform random initialization, the RNN-Scaling method significantly outperforms the RNN method. Additionally, we compare the CPU times of the three methods for varying numbers of points. Fig. 12(a) illustrates the results with the default initialization. The CPU

Table 8: The CPU times (seconds) for the RNN methods with the default initialization for Example 2.1.

method	N	M	50	100	150	200	250	300
RNN	8	CPU time	1.15	2.09	2.82	3.91	5.11	6.35
	12	CPU time	1.59	2.50	3.97	5.30	6.68	7.62
	16	CPU time	1.70	3.38	4.87	6.45	8.44	9.76
	24	CPU time	2.07	4.23	6.50	9.49	11.25	13.57
	32	CPU time	2.31	4.72	7.05	10.58	13.00	16.01
RNN-S	8	CPU time	1.11	1.87	2.83	4.06	5.02	6.29
	12	CPU time	1.32	2.36	3.70	5.30	6.59	8.09
	16	CPU time	1.65	3.08	4.99	6.51	8.58	10.45
	24	CPU time	2.22	4.30	6.33	9.92	10.75	13.90
	32	CPU time	2.42	4.89	6.83	10.07	13.02	15.92
RNN-BP	8	CPU time	1.99	3.85	5.69	7.89	10.46	13.19
	12	CPU time	2.70	4.77	7.38	11.00	13.37	16.60
	16	CPU time	2.75	6.28	9.09	12.94	16.71	19.17
	24	CPU time	3.63	7.53	12.08	16.24	19.72	25.10
	32	CPU time	4.02	8.01	12.72	18.53	23.64	28.74



(a) The absolute error of the RNN method.

(b) The absolute error of the RNN-Scaling method.

(c) The absolute error of the RNN-BP method.

Figure 9: The absolute errors of three methods with uniform random initialization for Example 2.1. (The neural network architecture is [2, 100, 300, 1] and $N=32$.)

times for all three methods increase as the number of points increases. The CPU times for the RNN and RNN-Scaling methods are approximately the same, while the RNN-BP method takes roughly twice as long as the other two methods.

We fix the number of collocation points to compare the three methods for varying M. Figs. 10(c)-(d) show the comparison results with the default initialization and uniform random initialization, respectively. The RNN-BP method outperforms the other two methods in terms of accuracy under both initialization methods. With the default initialization, the RNN-Scaling method achieves smaller errors than the RNN method for $M \leq 150$; while the errors of both methods become similar for $M > 150$. With the

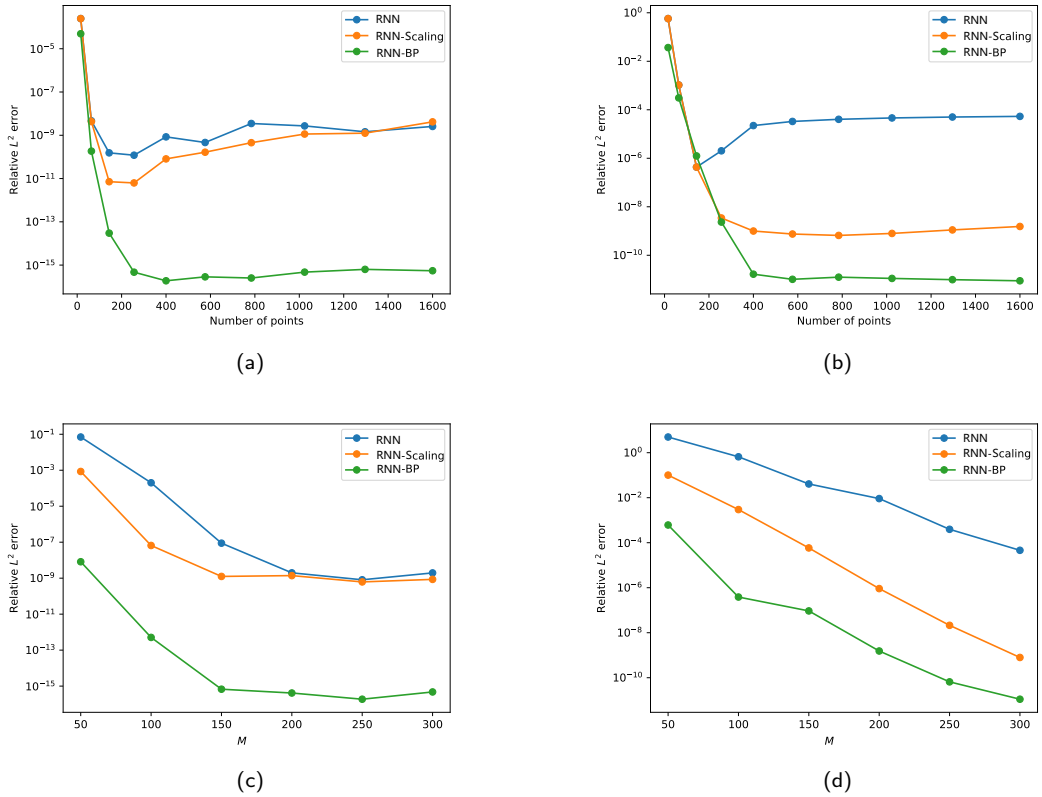


Figure 10: Comparison between the three RNN methods for Example 2.1: (a) Varying numbers of points with the default initialization. (b) Varying numbers of points with uniform random initialization ($R_m=1$). (c) Varying M with the default initialization. (d) Varying M with uniform random initialization ($R_m=1$).

uniform random initialization, the RNN-Scaling method consistently produces smaller errors than the RNN method, with a reduction of up to 5 orders of magnitude. We compare the CPU times of the three methods for varying M with fixed $N=12$ and default initialization, as shown in Fig. 12(b). The computation time for the standard RNN method is similar to that of the RNN-Scaling method, while the RNN-BP method takes approximately twice as long as the other two methods.

We compare the three methods for varying R_m, with $M=300$ and $N=32$, and vary R_m ranging from 0.1 to 2. Fig. 11 shows that the three methods exhibit a similar trend as R_m varies. Specifically, for R_m in the range of 0.2 to 1.1, the errors of the three methods are relatively small. For $R_m>1.1$, the errors increase rapidly. The RNN-BP method still produces the smallest error.

We fix the network architecture as $[2,100,300,1]$ and set N=48 to compare the accuracy and efficiency of the RNN methods with PINNs and DRM. For the biharmonic equation,

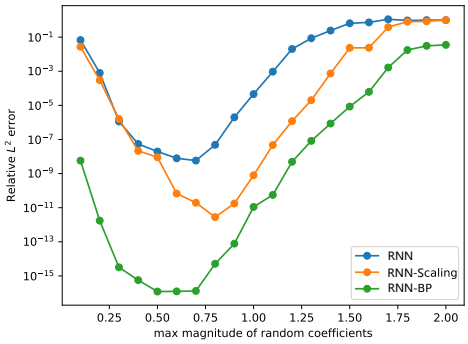
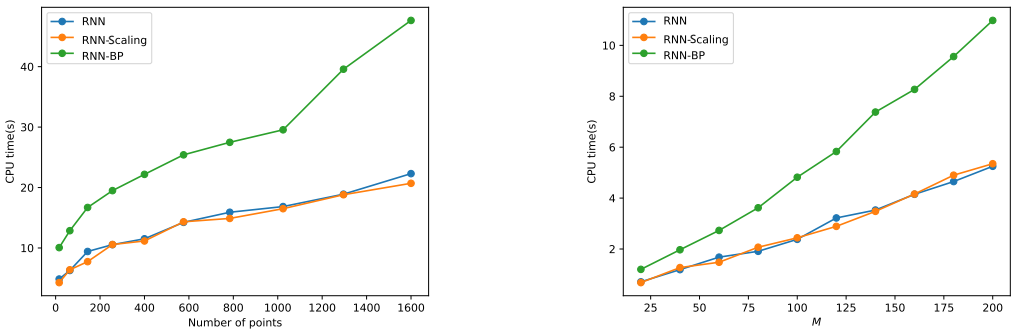


Figure 11: Comparison of relative L^2 errors of three methods varying max magnitude of random coefficients for Example 2.1.



(a) Varying number of points with the default initialization.

(b) Varying values of M with the default initialization.

Figure 12: Comparison of the CPU times of the three RNN methods for Example 2.1.

the loss functions of PINNs and DRM on uniform points are

$$L_{PINNs}=\frac{1}{N_f}\sum_{i=1}^{N_f}\left[\Delta^2\hat{u}(\mathbf{x}_f^i)-f(\mathbf{x}_f^i)\right]^2+\frac{\beta}{N_b}\sum_{i=1}^{N_b}\left[\hat{u}(\mathbf{x}_b^i)-g_1(\mathbf{x}_b^i)\right]^2 + \frac{\beta}{N_b}\sum_{i=1}^{N_b}\left[\frac{\partial\hat{u}}{\partial\mathbf{n}}(\mathbf{x}_b^i)-g_2(\mathbf{x}_b^i)\right]^2,$$

$$L_{DRM}=\frac{1}{N_f}\sum_{i=1}^{N_f}\left[\frac{1}{2}\left\|\Delta\hat{u}(\mathbf{x}_f^i)\right\|_2^2-f(\mathbf{x}_f^i)\hat{u}(\mathbf{x}_f^i)\right] + \frac{\beta}{N_b}\sum_{i=1}^{N_b}\left(\left[\hat{u}(\mathbf{x}_b^i)-g_1(\mathbf{x}_b^i)\right]^2+\left[\frac{\partial\hat{u}}{\partial\mathbf{n}}(\mathbf{x}_b^i)-g_2(\mathbf{x}_b^i)\right]^2\right),$$

Table 9: Comparison between the three RNN methods and PINNs/DRM, in terms of relative L^2 errors and the computation times.

Method	$\|e\|_{L^2}$	Epochs	Computation time (seconds)
PINNs (Adam)	3.79e-3	50,000	7,082.96
PINNs (L-BFGS)	1.38e-4	500	883.90
DRM (Adam)	9.72e-2	50,000	1,180.29
DRM (L-BFGS)	9.69e-2	500	79.49
RNN	1.96e-9	0	16.01
RNN-S	8.57e-10	0	15.93
RNN-BP	4.73e-16	0	28.74

respectively. In the PINNs/Adam and DGM/Adam methods, the network is trained for 50,000 epochs, with the learning rate coefficient gradually decreasing in a cosine schedule from 0.001 at the beginning to 10^{-5} at the end. In the PINNs/L-BFGS and DRM/L-BFGS methods, the network is trained for 500 epochs with a constant learning rate of 1. The comparison results are presented in Table 9. Compared to PINNs, DRM has a lower computational cost but a larger error. The computation time of RNN methods is 0.2% to 0.4% that of PINNs, and the error is 5 to 12 orders of magnitude smaller than that of PINNs. The three RNN methods achieve higher accuracy with less computation time.

4.2.2 Example 2.2

In this example, we consider the exact solution on $\Omega=(0,1)\times(0,1)$ as follows:

$$u(x,y)=\sin(2\pi x)\sin(2\pi y).$$

The numerical performance of this example is generally similar to that of Example 2.1, so we will provide a brief analysis of the numerical results. Tables 10 and 11 compare the three methods for varying numbers of collocation points and M, and initialization methods. It can be seen that the error with the default initialization is smaller, with a reduction of up to 5 orders of magnitude. Under both initialization methods, the RNN-BP method produces the smallest errors. With uniform random initialization with ($R_m=1$), the RNN-Scaling method is more accurate than the RNN method, with a reduction in error of up to 5 orders of magnitude. Figs. 13(a)-(c) plot the absolute errors of the three methods, showing that the RNN-Scaling method significantly reduces the error and the RNN-BP method is exact on the boundary.

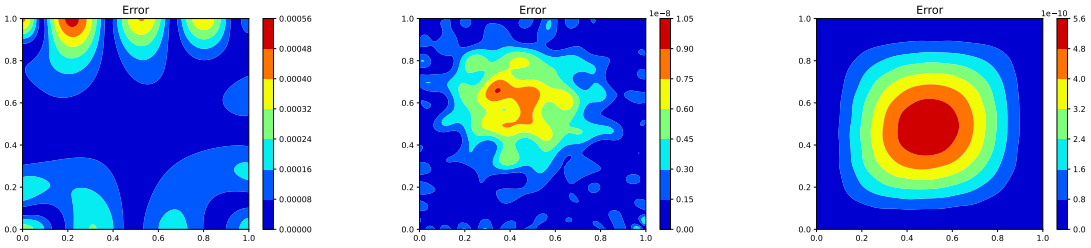
Figs. 14(a)-(b) compare the three methods for varying numbers of collocation points, with Fig. 14(a) employing the default initialization and Fig. 14(b) employing uniform random initialization, and the network fixed as $[2,100,300,1]$. As the number of collocation points increases, the errors of the three methods decrease rapidly. Figs. 14(c)-(d) compare the three methods for different M, with the network fixed as $[2,100,M,1]$ and N fixed at 32. As M increases, the errors of the three methods also decrease rapidly.

Table 10: Comparison between the three RNN methods with the default initialization for Example 2.2.

method	N	M	50	100	150	200	250	300
RNN	8	$\|e\|_{L^2}$	1.29e+2	4.61e-2	4.77e-5	1.16e-5	1.22e-5	1.80e-5
	12	$\|e\|_{L^2}$	1.15e+2	1.12e-1	6.48e-4	7.91e-7	5.14e-7	4.54e-7
	16	$\|e\|_{L^2}$	1.12e+2	2.16e-1	9.74e-4	8.81e-6	9.12e-7	6.00e-6
	24	$\|e\|_{L^2}$	1.13e+2	3.42e-1	1.17e-3	1.44e-5	5.13e-6	5.87e-6
	32	$\|e\|_{L^2}$	1.15e+2	4.18e-1	1.25e-3	1.65e-5	6.47e-6	2.41e-5
RNN-S	8	$\|e\|_{L^2}$	2.06e-1	1.21e-4	1.41e-5	8.28e-6	9.92e-6	6.01e-6
	12	$\|e\|_{L^2}$	1.76e-1	6.80e-5	3.22e-7	2.20e-7	4.97e-8	5.18e-8
	16	$\|e\|_{L^2}$	1.67e-1	5.36e-5	3.32e-7	1.29e-7	1.53e-7	7.00e-8
	24	$\|e\|_{L^2}$	1.82e-1	1.14e-4	8.63e-7	8.79e-7	6.97e-7	7.03e-8
	32	$\|e\|_{L^2}$	2.19e-1	1.50e-4	2.55e-6	3.17e-6	4.91e-6	7.05e-7
RNN-BP	8	$\|e\|_{L^2}$	8.12e-6	2.19e-6	7.74e-7	3.76e-7	3.25e-7	3.75e-7
	12	$\|e\|_{L^2}$	1.47e-5	5.28e-9	3.22e-10	2.11e-10	1.64e-10	9.05e-11
	16	$\|e\|_{L^2}$	1.37e-5	1.12e-9	8.14e-12	1.67e-12	5.67e-13	1.18e-12
	24	$\|e\|_{L^2}$	1.32e-5	2.05e-9	6.61e-12	1.58e-12	9.17e-13	1.37e-12
	32	$\|e\|_{L^2}$	1.30e-5	2.58e-9	1.04e-11	1.04e-12	7.39e-13	1.94e-12

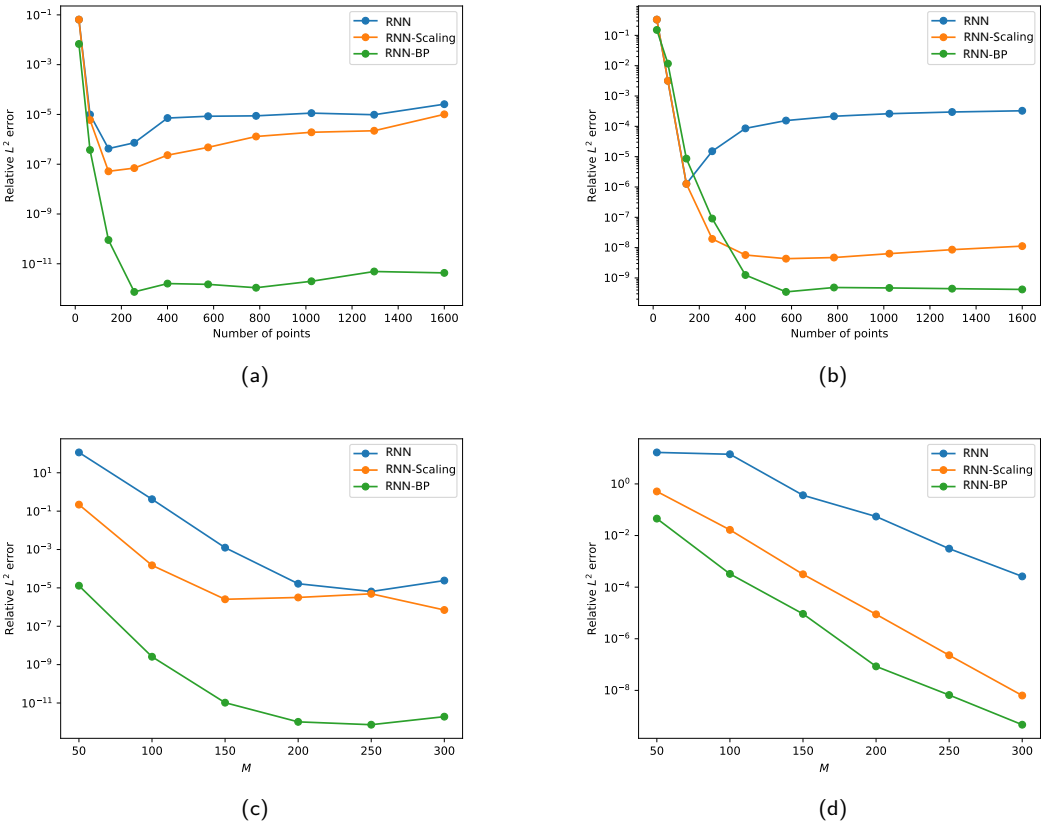
Table 11: Comparison between the three RNN methods with uniform random initialization ($R_m = 1$) for Example 2.2.

method	N	M	50	100	150	200	250	300
RNN	8	$\|e\|_{L^2}$	2.09e+1	1.87e-2	2.48e-3	2.26e-3	1.31e-3	3.17e-3
	12	$\|e\|_{L^2}$	2.09e+1	1.87e-2	2.48e-3	2.26e-3	1.31e-3	3.17e-3
	16	$\|e\|_{L^2}$	2.09e+1	1.87e-2	2.48e-3	2.26e-3	1.31e-3	3.17e-3
	24	$\|e\|_{L^2}$	1.69e+1	1.31e+1	3.35e-1	4.36e-2	2.59e-3	1.55e-4
	32	$\|e\|_{L^2}$	1.66e+1	1.41e+1	3.64e-1	5.45e-2	3.10e-3	2.61e-4
RNN-S	8	$\|e\|_{L^2}$	3.03e-1	7.73e-3	2.48e-3	2.26e-3	1.31e-3	3.17e-3
	12	$\|e\|_{L^2}$	2.02e-1	7.74e-3	2.24e-4	5.19e-6	2.82e-6	1.27e-6
	16	$\|e\|_{L^2}$	1.69e-1	1.40e-2	1.65e-4	4.60e-6	1.63e-7	1.94e-8
	24	$\|e\|_{L^2}$	2.65e-1	1.54e-2	2.48e-4	6.65e-6	1.20e-7	4.32e-9
	32	$\|e\|_{L^2}$	5.11e-1	1.66e-2	3.15e-4	8.86e-6	2.30e-7	6.33e-9
RNN-BP	8	$\|e\|_{L^2}$	2.03e-2	6.73e-3	4.88e-3	5.21e-3	4.64e-3	1.17e-2
	12	$\|e\|_{L^2}$	5.73e-2	2.70e-4	2.38e-5	1.57e-5	1.42e-5	8.63e-6
	16	$\|e\|_{L^2}$	5.25e-2	3.07e-4	3.23e-6	4.44e-7	4.19e-7	9.10e-8
	24	$\|e\|_{L^2}$	4.77e-2	3.14e-4	7.98e-6	9.57e-8	7.94e-9	3.47e-10
	32	$\|e\|_{L^2}$	4.53e-2	3.26e-4	9.20e-6	8.52e-8	6.61e-9	4.68e-10



(a) The absolute error of RNN. (b) The absolute error of RNN-Scaling method. (c) The absolute error of RNN-BP.

Figure 13: The absolute errors of three methods with uniform random initialization for Example 2.2. (The neural network architecture is $[2,100,300,1]$ and $N=32$.)



(a) (b) (c) (d)

Figure 14: Comparison between the three RNN methods for Example 2.2: (a) Varying numbers of points with the default initialization. (b) Varying numbers of points with uniform random initialization ($R_m=1$). (c) Varying M with the default initialization. (d) Varying M with uniform random initialization ($R_m=1$).

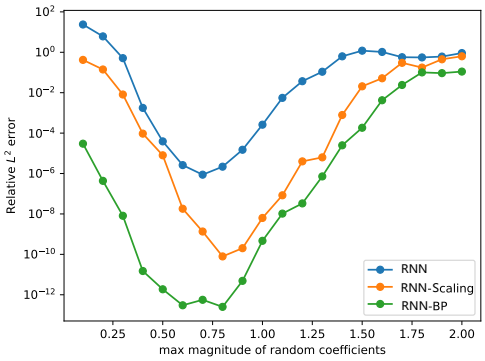


Figure 15: Comparison of relative L^2 errors of the three methods varying max magnitude of random coefficients for Example 2.2.

From Figs. 14(a)-(d), it can be observed that the RNN-BP method consistently produces the smallest error, and the RNN-Scaling method is more accurate than the RNN method under uniform random initialization ($R_m=1$).

Fig. 15 compares the three methods for varying R_m, with the fixed network $[2,100,300,1]$ and $N=32$. The three methods achieve smaller errors for R_m in the range of 0.2 to 1.1. The smallest errors of the RNN-BP, RNN-Scaling, and RNN methods are magnitudes of 10^{-12}, 10^{-10}, and 10^{-6}, respectively, indicating the effectiveness of the RNN-BP and RNN-Scaling methods.

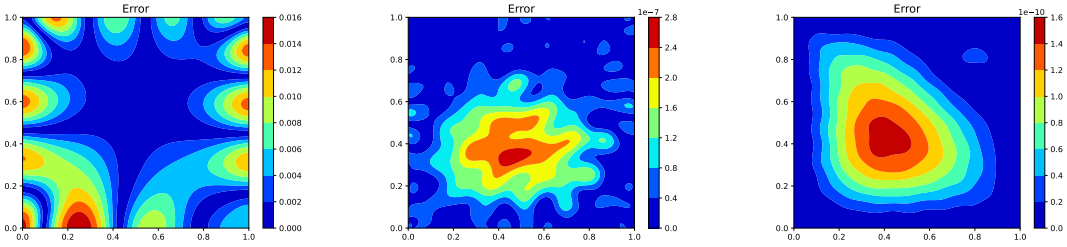
4.2.3 Example 2.3

This example considers the exact solution with the exponential form

$$u(x,y)=e^{x^2+y^2+xy}$$

on $\Omega=(0,1)\times(0,1)$, where the source term and boundary conditions are derived from this solution.

Similar to Example 2.1, Figs. 17(a)-(b) compare the three methods for varying numbers of collocation points, using the fixed network $[2,100,300,1]$. Subsequently, Figs. 17(c)-(d) compare the three methods for varying M, with the network architecture $[2,100,M,1]$ and $N=32$. Figs. 17(a) and (c) employ the default initialization, and Figs. 17(b) and (d) utilize uniform random initialization ($R_m=1$). As in Example 2.1, the RNN-BP method consistently achieves the smallest errors, reaching levels as low as 10^{-14}. Figs. 16(a)-(c) show the absolute errors for the three methods using the network architecture [2, 100, 300, 1] and $N=32$. The RNN-Scaling method achieves smaller errors than the RNN method, reducing errors by up to 5 orders of magnitude under uniform random initialization.



(a) The absolute error of the RNN method.

(b) The absolute error of the RNN-Scaling method.

(c) The absolute error of the RNN-BP method.

Figure 16: The absolute errors of the three methods with uniform random initialization for Example 2.3. (The neural network architecture is $[2,100,300,1]$ and $N=32$.)

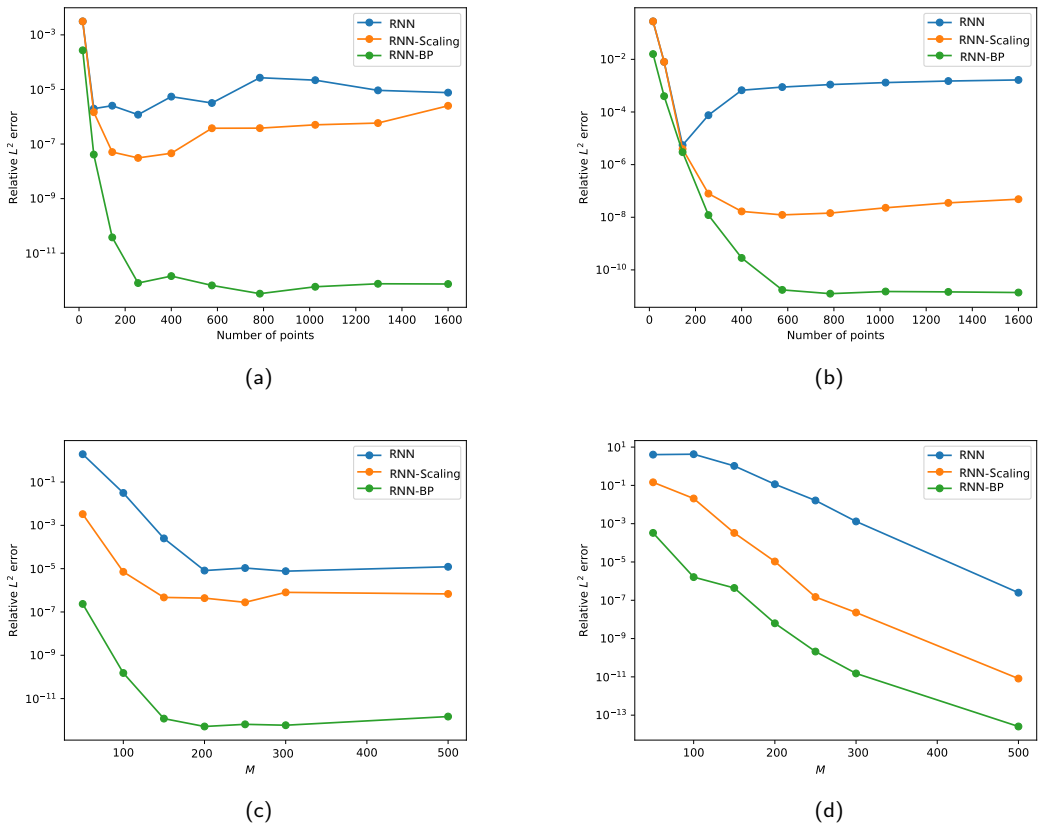


Figure 17: Comparison between the three RNN methods for Example 2.3: (a) Varying numbers of points with the default initialization. (b) Varying numbers of points with uniform random initialization ($R_m=1$). (c) Varying M with the default initialization. (d) Varying M with the uniform random initialization ($R_m=1$).

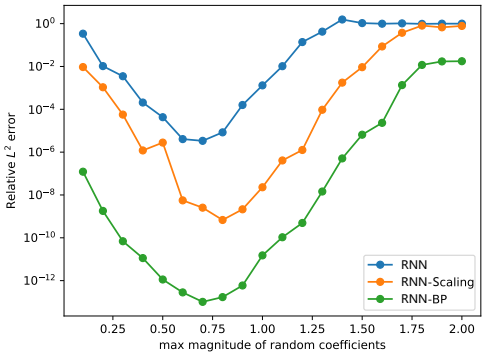


Figure 18: Comparison of relative L^2 errors of the three RNN methods varying max magnitude of random coefficients for Example 2.3.

Fig. 18 compares the three methods for varying R_m, using the fixed network [2,100,300,1] and $N=32$. The three methods achieve their smallest errors when R_m is in the range of 0.2 to 1.3. The smallest errors of the RNN-BP, RNN-Scaling, and RNN methods are magnitude of 10^{-13}, 10^{-10}, and 10^{-6}, respectively, demonstrating the effectiveness of the RNN-BP and RNN-Scaling methods.

4.2.4 Example 2.4

This example validates that the RNN-BP method is exact for solutions of the form (3.6). We consider the exact solution on $\Omega=(0,1)\times(0,1)$:

$$u(x,y)=x^{10}+y^{10}+x^k\sin y+y^k\cos x.$$

For $k=3$ and $k=4$, the three methods are compared for varying numbers of collocation points, using the fixed network [2,100,300,1] and uniform random initialization ($R_m=1$). The results are presented in Table 12. For the RNN-BP method, the relative L^2 errors reach machine precision when $k=3$, and the errors decrease rapidly as the number of

Table 12: Comparison between the three RNN methods with different N for Example 2.4.

k	N	4	8	12	16	20	24	28
3	RNN	7.30e-1	5.18e-2	5.55e-5	2.34e-4	3.75e-3	5.91e-3	7.47e-3
	RNN-S	7.30e-1	5.18e-2	5.16e-5	1.07e-6	1.44e-7	1.35e-7	1.49e-7
	RNN-BP	4.02e-16	4.03e-16	4.01e-16	9.82e-16	4.03e-16	4.00e-16	4.03e-16
4	RNN	8.07e-1	5.65e-2	5.62e-5	2.53e-4	4.08e-3	6.42e-3	8.11e-3
	RNN-S	8.07e-1	5.65e-2	5.62e-5	1.16e-6	1.56e-7	1.46e-7	1.61e-7
	RNN-BP	1.20e-4	1.22e-6	3.96e-9	1.56e-11	1.21e-13	5.44e-14	5.79e-14

collocation points increase when $k=4$, reaching levels as low as 10^{-14}. This confirms that the RNN-BP method is exact for solutions of the form (3.6). We also observe that as the number of collocation points increases, the errors of the RNN initially decrease but then increase. In contrast, the errors of the RNN-Scaling method decrease as the number of collocation points increase, and they are significantly smaller than those of the RNN method.

4.2.5 Example 2.5

The fifth numerical example uses the following exact solution on $(0,2)\times(0,2)$:

$$u(x,y)=-\left[2\cos\left(\frac{3}{2}\pi x+\frac{2\pi}{5}\right)+\frac{3}{2}\cos\left(3\pi x-\frac{\pi}{5}\right)\right]\left[2\cos\left(\frac{3}{2}\pi y+\frac{2\pi}{5}\right)+\frac{3}{2}\cos\left(3\pi y-\frac{\pi}{5}\right)\right].$$

The figure of the exact solution is displayed in 20(a).

Fig. 19(a) compares the three methods for varying numbers of collocation points, using the fixed network $[2,100,500,1]$ and uniform random initialization ($R_m=1$). Fig. 19(b) compares the methods for varying M, using the fixed network $[2,100,M,1]$ and uniform random initialization ($R_m=1$). Finally, Fig. 19(c) compares the methods for varying R_m, using the fixed network $[2,100,500,1]$ and $N=32$. The relative L^2 errors for the three methods reach the levels of 10^{-4}, 10^{-6}, and 10^{-10}, respectively. The RNN-BP method consistently achieves the smallest errors, followed by the RNN-Scaling and RNN methods, demonstrating the effectiveness of the RNN-BP and RNN-Scaling methods. The absolute errors are shown in Figs. 20(c)-(d). The errors of the RNN method are concentrated near the boundary, while the errors of the other two methods are not.

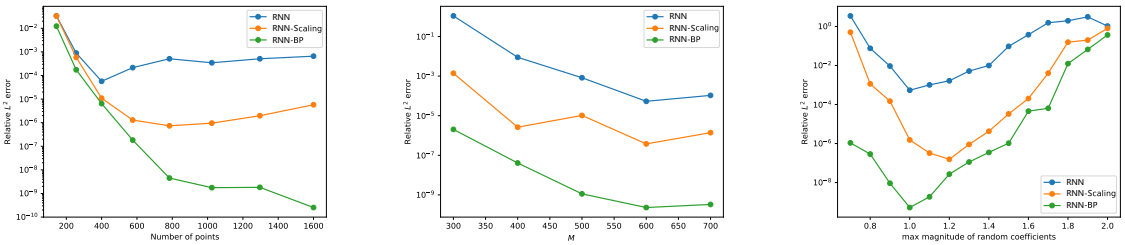
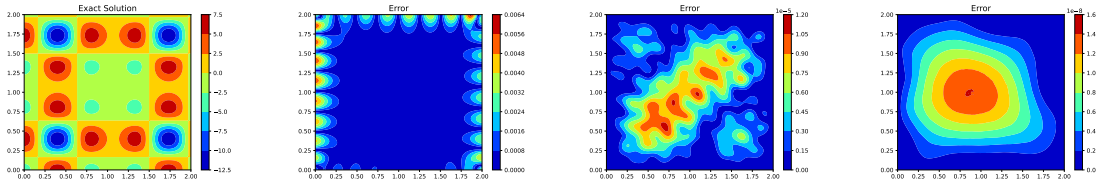


Figure 19: Comparison of relative L^2 errors of the three RNN methods for Example 2.5: (a) Varying numbers of points with uniform random initialization ($R_m=1$) and the fixed network architecture $[2,100,500,1]$. (b) Varying M with uniform random initialization ($R_m=1$), network architecture $[2,100,M,1]$ and $N=32$. (c) Varying R_m with the fixed network architecture $[2,100,500,1]$ and $N=32$.

4.2.6 Example 2.6

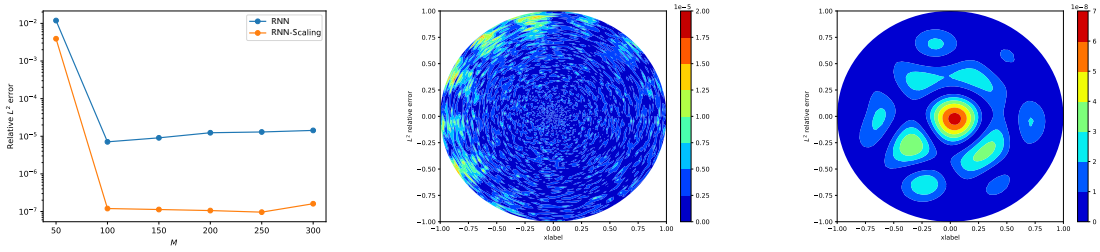
The final example uses the exact solution

$$u(x,y)=x^4+y^4$$



(a) The exact solution. (b) The absolute error of the RNN method. (c) The absolute error of the RNN-Scaling method. (d) The absolute error of the RNN-BP method.

Figure 20: The exact solution, absolute errors of three methods with uniform random initialization ($R_m=1$) for Example 2.5. (The network architecture is $[2,100,500,1]$ and $N=32$.)



(a) (b) (c)

Figure 21: Comparison between the RNN and RNN-Scaling methods for Example 2.6: (a) Varying M with the default initialization, network architecture $[2,100,100,M,1]$ and $N=64$. (b) The absolute error of the RNN method. (c) The absolute error of the RNN-Scaling method.

on the unit circle centered at the origin $(0,0)$.

We vary M to compare the RNN method and the RNN-Scaling method, using the network architecture $[2,100,100,M,1]$, $N=64$, and the default initialization. Fig. 21(a) shows the comparison results, indicating that the RNN-Scaling method achieves the smallest errors, typically reducing errors by 2 orders of magnitude compared to the RNN method. Figs. 21(b)-(c) show the absolute errors for both methods with $N=64$ and $M=250$. We observe that the errors of the RNN method are concentrated near the boundary, while the errors of the RNN-Scaling method are concentrated in the interior.

From Examples 2.1 to 2.6, we summarize as follows: the RNN-BP method consistently achieves the smallest errors, although its computation time is approximately 2 to 3 times that of the other two methods. In Examples 2.1 to 2.3, for relatively large M and numbers of the collocation points, the RNN-BP method reduces errors by 6-7 orders of magnitude compared to the RNN method, with reductions of up to 9 orders of magnitude in some cases.

For the RNN-Scaling method, when M and the number of the collocation points are relatively large, the errors in Examples 2.1 to 2.4 are reduced by 4-5 orders of magnitude, while the errors in Example 2.5 are reduced by 2-3 orders of magnitude, demonstrating the efficiency of the RNN-Scaling method.

5 Conclusion

This work proposes two improved RNN methods for solving elliptic equations. The first is the RNN-BP method, which enforces exact boundary conditions on rectangular domains, including both Dirichlet and clamped boundary conditions. The enforcement approach for the clamped boundary condition is direct and does not require the introduction of auxiliary gradient variables, which reduces computational cost and avoids potential additional errors. We demonstrate theoretically and numerically that the RNN-BP method is exact for solutions with specific forms.

Secondly, the RNN-Scaling method is introduced, which modifies the linear algebraic equations by increasing the weight of boundary equations. The method is extended to the circular domain, achieving errors 1-2 orders of magnitude smaller than those of the RNN, with potential for generalization to general domains.

Finally, we provide several numerical examples to compare the performance of the three RNN methods. The computation time of the RNN and RNN-Scaling methods is almost the same, and the computation time of the RNN-BP method is about 2-3 times that of the RNN method. Both improved randomized neural network methods achieve higher accuracy than the RNN method, with the error reduced by 6 orders of magnitude for some tests. We also compare the three RNN methods with PINNs and DRM. The errors of the RNN methods are approximately 5-14 orders of magnitude smaller than those of PINNs and DRM, while the computation time is significantly reduced, demonstrating the superiority of the RNN methods.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China (12201246, 12071045), Fund of National Key Laboratory of Computational Physics, LCP Fund for Young Scholar (6142A05QN22010), National Key R&D Program of China (2020YFA0713601), and by the Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun, 130012, P.R. China.

References

[1] J. W. Barrett, J. F. Blowey, and H. Garcke, *Finite element approximation of the Cahn–Hilliard equation with degenerate mobility*, SIAM Journal on Numerical Analysis, 37 (1999), pp. 286–318.

[2] M. Ben-Artzi, I. Chorev, J.-P. Croisille, and D. Fishelov, *A compact difference scheme for the biharmonic equation in planar irregular domains*, SIAM Journal on Numerical Analysis, 47 (2009), pp. 3087–3108.

[3] B. Bialecki, *A fourth order finite difference method for the Dirichlet biharmonic problem*, Numerical Algorithms, 61 (2012), pp. 351–375.

[4] B. Bialecki and A. Karageorghis, *Spectral Chebyshev collocation for the Poisson and biharmonic equations*, SIAM Journal on Scientific Computing, 32 (2010), pp. 2995–3019.
[5] J. Chen, R. Du, and K. Wu, *A comparison study of deep Galerkin method and deep Ritz method for elliptic problems with different boundary conditions*, Communications in Mathematical Research, 36 (2020), pp. 354–376.
[6] M. Cui and S. Zhang, *On the uniform convergence of the weak Galerkin finite element method for a singularly-perturbed biharmonic equation*, Journal of Scientific Computing, 82 (2020), pp. 1–15.
[7] E. Doha and A. Bhrawy, *Efficient spectral-Galerkin algorithms for direct solution of fourth-order differential equations using Jacobi polynomials*, Applied Numerical Mathematics, 58 (2008), pp. 1224–1244.
[8] S. Dong and N. Ni, *A method for representing periodic functions and enforcing exactly periodic boundary conditions with deep neural networks*, Journal of Computational Physics, 435 (2021), p. 110242.
[9] W. E and B. Yu, *The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems*, Communications in Mathematics and Statistics, 6 (2018), pp. 1–12.
[10] S. Henn, *A multigrid method for a fourth-order diffusion equation with application to image processing*, SIAM Journal on Scientific Computing, 27 (2005), pp. 831–849.
[11] A. D. Jagtap, E. Kharazmi, and G. E. Karniadakis, *Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems*, Computer Methods in Applied Mechanics and Engineering, 365 (2020), p. 113028.
[12] I. Lagaris, A. Likas, and D. Fotiadis, *Artificial neural networks for solving ordinary and partial differential equations*, IEEE Transactions on Neural Networks, 9 (1998), pp. 987–1000.
[13] I. Lagaris, A. Likas, and D. Papageorgiou, *Neural-network methods for boundary value problems with irregular boundaries*, IEEE Transactions on Neural Networks, 11 (2000), pp. 1041–1049.
[14] C. Le Potier, *Finite volume monotone scheme for highly anisotropic diffusion operators on unstructured triangular meshes*, Comptes Rendus Mathematique, 341 (2005), pp. 787–792.
[15] Y. Liu and W. Ma, *Gradient auxiliary physics-informed neural network for nonlinear biharmonic equation*, Engineering Analysis with Boundary Elements, 157 (2023), pp. 272–282.
[16] L. Lyu, K. Wu, R. Du, and J. Chen, *Enforcing exact boundary and initial conditions in the deep mixed residual method*, CSIAM Transactions on Applied Mathematics, 2 (2021), pp. 748–775.
[17] L. Lyu, Z. Zhang, M. Chen, and J. Chen, *MIM: A deep mixed residual method for solving high-order partial differential equations*, Journal of Computational Physics, 452 (2022), p. 110930.
[18] N. Mai-Duy and R. Tanner, *A spectral collocation method based on integrated Chebyshev polynomials for two-dimensional biharmonic boundary-value problems*, Journal of Computational and Applied Mathematics, 201 (2007), pp. 30–47.
[19] W. Ming and J. Xu, *The Morley element for fourth order elliptic equations in any dimensions*, Numerische Mathematik, 103 (2006), pp. 155–169.
[20] ——, *Nonconforming tetrahedral finite elements for fourth order elliptic equations*, Mathematics of Computation, 76 (2007), pp. 1–19.
[21] R. Mohanty, *A fourth-order finite difference method for the general one-dimensional nonlinear biharmonic problems of first kind*, Journal of Computational and Applied Mathematics, 114 (2000), pp. 275–290.
[22] I. Mozolevski and E. Süli, *A priori error analysis for the hp-version of the discontinuous Galerkin finite element method for the biharmonic equation*, Computational Methods in Applied

Mathematics, 3 (2003), pp. 596–607.
[23] C. PARK AND D. SHEEN, *A quadrilateral Morley element for biharmonic equations*, Numerische Mathematik, 124 (2013), pp. 395–413.
[24] M. RAISSI, P. PERDIKARIS, AND G. KARNIADAKIS, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, Journal of Computational Physics, 378 (2019), pp. 686–707.
[25] H. SHENG AND C. YANG, *PFNN: A penalty-free neural network method for solving a class of second-order boundary-value problems on complex geometries*, Journal of Computational Physics, 428 (2021), p. 110085.
[26] Z. SHENG AND G. YUAN, *A nine point scheme for the approximation of diffusion operators on distorted quadrilateral meshes*, SIAM Journal on Scientific Computing, 30 (2008), pp. 1341–1361.
[27] ——, *A new nonlinear finite volume scheme preserving positivity for diffusion equations*, Journal of Computational Physics, 315 (2016), pp. 182–193.
[28] J. SIRIGNANO AND K. SPILIOPOULOS, *DGM: A deep learning algorithm for solving partial differential equations*, Journal of Computational Physics, 375 (2018), pp. 1339–1364.
[29] N. SUKUMAR AND A. SRIVASTAVA, *Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks*, Computer Methods in Applied Mechanics and Engineering, 389 (2022), p. 114333.
[30] E. VENTSEL AND T. KRAUTHAMMER, *Thin Plates and Shells: Theory, Analysis, and Applications*, Marcel Deekker Inc, New York, 2nd edition ed., 2001.
[31] Y. XIE, Y. MA, AND Y. WANG, *Automatic boundary fitting framework of boundary dependent physics-informed neural network solving partial differential equation with complex boundary conditions*, Computer Methods in Applied Mechanics and Engineering, 414 (2023), p. 116139.
[32] M. XU AND C. SHI, *A Hessian recovery-based finite difference method for biharmonic problems*, Applied Mathematics Letters, 137 (2023), p. 108503.
[33] Y. ZANG, G. BAO, X. YE, AND H. ZHOU, *Weak adversarial networks for high-dimensional partial differential equations*, Journal of Computational Physics, 411 (2020), p. 109409.
[34] R. ZHANG AND Q. ZHAI, *A weak Galerkin finite element scheme for the biharmonic equations by using polynomials of reduced order*, Journal of Scientific Computing, 64 (2015), pp. 559–585.
[35] S. ZHANG, *Minimal consistent finite element space for the biharmonic equation on quadrilateral grids*, IMA Journal of Numerical Analysis, 40 (2020), pp. 1390–1406.