scientific reports

Check for updates

OPEN

A neural network-based PDE solving algorithm with high precision

Zichao Jiang, Junyang Jiang, Qinghe Yao✉ & Gengchao Yang

The consumption of solving large-scale linear equations is one of the most critical issues in numerical computation. An innovative method is introduced in this study to solve linear equations based on deep neural networks. To achieve a high accuracy, we employ the residual network architecture and the correction iteration inspired by the classic iteration methods. By solving the one-dimensional Burgers equation and the two-dimensional heat-conduction equation, the precision and effectiveness of the proposed method have been proven. Numerical results indicate that this DNN-based technique is capable of obtaining an error of less than 10^{-7}. Moreover, its computation time is less sensitive to the problem size than that of classic iterative methods. Consequently, the proposed method possesses a significant efficiency advantage for large-scale problems.

The numerical methods for solving partial differential equations (PDEs) are among the most challenging and critical engineering problems. The discrete PDEs form sparse linear equations and are usually solved by iteration methods, e.g., the Gauss–Seidel method[1], the conjugate gradient (PCG) method, etc.[2–4]. Classic iterative solvers assure the precision and the reliability of the solutions but bring the challenge in terms of computational consumption.

In recent years, deep neural network (DNN), which reveals superior capability to process and to predict complicated systems, has been widely employed in a variety of fields, e.g., natural language processing[5], computer vision[6], etc.[7,8] Due to its high computational efficiency and scalability, especially on heterogeneous platforms, DNN has become a promising technique in scientific computing and even provides the possibility for real-time PDE solving[9]. Ray et al.[10] proposed a neural network-based indicator to correct the irregular solution in the discontinuous Galerkin scheme. Chan et al. and Wang et al.[11,12] employed neural networks to solve multiscale problems. In the numerical simulation of engineering problems, DNN is utilized as a direct solver of the approximated system, and solutions are found with high efficiency[13–16].

The idea of utilizing DNN to achieve numerical solutions of PDEs brings certain potential risks as well. According to the universal approximation theorem, the piecewise continuous objective function is a necessary condition for the reliability of DNN approximation[17,18]. However, solutions of the PDEs, particularly the nonlinear PDEs, do not always satisfy the regularity condition[19]. Accordingly, the solutions achieved from the direct DNN prediction usually have inadequate numerical accuracy. Therefore, many published neural network-based studies focus on semi-analytic models as an alternative to the direct prediction. For instance, Raissi et al.[20] first proposed physics-informed neural network (PINN) to solve PDEs by embedding integral forms into the loss function. Ehsan et al.[21] developed the hp-VPINN, an improved version of PINN with a higher accuracy. Ew[22] employed the DNN to solve variational problems that arise from PDEs. These methods translate the original PDEs into parametric models and design the penalty functions to ensure the compatibility of the DNN models with the original PDEs and improve the accuracy of the DNN-based solutions but their accuracy is usually limited. Some recent studies have employed novel models that have been employed in other areas, e.g. using ConvLSTM[23] and data-driven GMG[24] to solve Naiver-Stokes equation, but their lowest error is in the range of 10^{-3}–10^{-2}.

Solving the intermediate linear equations through DNN is another intuitive and enlightening idea to combine the advantages of neural networks and classic numerical methods. Replacing the classic iteration methods like PCG and GMRES[25] with DNN can directly accelerate the computational process of the finite element method (FEM), the finite difference method (FDM), and the spectral methods. For instance, Xiao et al.[26] proposed a neural network-based solver to expedite the FDM solving process of the Poisson equation in fluid simulations. Following this approach, the network's architecture and the linear equations' regularity are the essential factors,

School of Aeronautics and Astronautics, Sun Yat-sen University, Guangzhou 510275, China. ✉email: yaoqhe@mail.sysu.edu.cn

and the former will be discussed in detail later in this manuscript. The regularity of the intermediate linear equations is usually controllable and readily quantifiable relative to the original PEDs or their parametric models. Therefore, solving the intermediate linear equations by DNN has potentially better improvement in terms of precision.

Interpretability[27] is another critical issue for the application of DNN in PDE, which can be comprehended as an explanation of how a particular input leads to a particular output in DNN and what factual information the DNN learned during training. The interpretability of a generic DNN model is still an open problem, which inevitably restricts the application of DNNs in numerical computation. However, DNN is proven sufficiently reliable as an inner interpolation when the training set is sampled uniformly enough to support almost all possible inputs. The homogeneous data set of linear equations is fortunately convenient to generate. Moreover, for a particular algorithm, the discretization scheme of the PDE is usually unique; thus, the corresponding linear equations are constrained to a small range. Therefore, the DNN-based solvers of the intermediate linear equations effectively avoid the challenge of interpretability.

Regarding the network architecture, the classic DNN model suffers from network degradation and is consequently unattainable to obtain high accuracy[28]. Specifically, the higher depth of the network does not improve the accuracy but even has a negative effect. To fix the problem of network degradation, He et al. proposed Res-Net[28]. A short connection called Res-block covering the hidden layers is involved in the network to maintain the training information transition to each hidden layer. It is proved effective in many practical applications, for example, Transformer proposed by Google[29]. In numerical computation and PDE, Tong et al. employed Res-Net in the simulations of the linear and nonlinear self-consistent systems[30]. Res-Net was also utilized by Ew[22], mentioned above.

This paper presents an innovative DNN-based algorithm to solve linear equations with high precision and efficiency. The basic idea of this method is to solve the linear equations based on DNN, as a substitute for the classic iteration algorithms. The proposed method maintains the high efficiency of DNN and assures the necessary precision for PDE solving via the following two aspects. Firstly, we employed the Res-Net architecture in the DNN model; thus, the accuracy can improve with the increasing depth of the network. Moreover, inspired by classic residual correction methods, we combined the DNN model with a correction iteration process, which reduces the error of the results from DNN iteratively.

To balance the network architecture's simplicity and the algorithm's generality, we focus on multi-diagonal linear equations. As one of the most common types of matrices, the multi-diagonal matrix is the discrete form of various PDEs. In FDM, many schemes correspond to differential equations with a naturally multi-diagonal structure. The equations can be further reduced even to tridiagonal equations when combined with the alternating direction implicit (ADI) method. For instance, both the Burgers equation and the heat-conduction equation have a 2nd-order precision scheme satisfying this property. More importantly, several vectors can directly represent a multi-diagonal equation, implying that an intuitive network input layer can be constructed based on this feature.

The DNN model predicts the solution of the equation based on the input set of vectors representing the equation and offers another vital performance advantage. In FDM, solving several linear equations with the same structure is often necessary. For example, in the ADI method, such a process exists at each time step. Traditional methods usually require circular construction of the matrix, which brings unavoidable additional computational time overhead. The algorithm, however, can solve multiple equations simultaneously, effectively avoiding this problem.

This paper is organized as follows: the main ideas are introduced in section "Methodology". In section "Precision-oriented parametric analysis", we investigate several factors affecting the numerical precision. Section "Numerical experiments and validation" presents the numerical results that demonstrate the precision and efficiency of the proposed method. The conclusions are drawn in section "Conclusion".

Methodology

Inputting equations in matrix form into a DNN model and improving the accuracy are two critical subroutines in the proposed algorithm. In this section we present a dimension-reduced representation of the matrices in FDM, which is also utilized as input to the DNN model. In design of the DNN model, we employ the Res-Net architecture mentioned in section "Introduction" to address some of the difficulties of DNN models, e.g. network degradation. However, the optimization of the network is not sufficient to improve the accuracy to that required for numerically solving the PDE, and we have therefore developed the iteration algorithm that will be introduced in detail in section "A correction iteration".

The representation of the linear equation. The inputs of a DNN model are usually vectors, whereas the linear equations in FDM are always in matrix form, thus we first consider how to convert the equations into a form that can be fed into the DNN model. Since the linear equations in FDM describe the relationships between neighboring nodes, the matrices are sparse and have fixed and regular non-zero element positions, which inspired us to adopt a dimension-reduced matrix representation.

In FDM, the discrete equations generally form

$$\sum_{d=1}^{D} a_0^{(d)} x_{\boldsymbol{p}} + \sum_{d=1}^{D} \sum_{m=-k}^{k} a_m^{(d)} x_{\boldsymbol{p}+m\boldsymbol{e}^{(d)}} = f_{\boldsymbol{p}}, \tag{1}$$

where D is the dimension and $\boldsymbol{p} \in \mathbb{Z}^D$ is the index vector of the node, $\boldsymbol{e}^{(d)}$ is the d-th unit vector, $x_{\boldsymbol{p}}$ is the solution while $f_{\boldsymbol{p}}$ is the righthand term at node $\boldsymbol{p}$. $a_m^{(d)}$ $(d \le D, m \le k)$ represents the constant coefficients of the difference scheme, where is the order of the scheme.

In this paper, we focus on the PDEs in rectangular domains in Cartesian coordinate systems, where the difference equations are correspondingly multi-diagonal and sparse. The choice of matrix representation is particularly important for the solving process, which drives us to adopt a formulation of the multidiagonal matrixes that is more appropriate for the neural networks.

Considering an arbitrary linear system $A\boldsymbol{x} = \boldsymbol{f}$, the left-hand matrix A can be decomposed in the form of

$$\begin{aligned} A &= \textstyle\sum_{i=0,\pm1,\pm2,\cdots} \mathrm{diag}_i\left(a_{i,1}, a_{i,2}, \cdots a_{i,(n-i)}\right) \\ &= \begin{bmatrix} a_{0,1} & a_{1,1} & \cdots & a_{(n-2),1} & a_{(n-1),1} \\ a_{-1,1} & a_{0,2} & a_{1,2} & \ddots & a_{(n-2),2} \\ \vdots & a_{-1,2} & a_{0,3} & \ddots & \vdots \\ a_{-(n-2),1} & \ddots & \ddots & \ddots & a_{1,(n-1)} \\ a_{-(n-1),1} & a_{-(n-2),2} & \cdots & a_{-1,(n-1)} & a_{0,n} \end{bmatrix}, \end{aligned} \tag{2}$$

where the symbol $\mathrm{diag}_i(\cdot)$ denotes the matrix that places the elements of the vector $\left(a_{i,1}, a_{i,2}, \cdots a_{i,(n-i)}\right)$ on the i-th diagonal. In particular, $\mathrm{diag}_0(\cdot)$ represents the main diagonal, $\mathrm{diag}_i(\cdot)(i > 0)$ is the diagonal above the main diagonal, and oppositely $\mathrm{diag}_i(\cdot)(i < 0)$ is the lower diagonal elements.

The matrix in (2) can be represented by a series of vectors, called diagonal vectors, and the number of diagonal vectors coincides with the size of the equation. Practically, for a sparse multi-diagonal matrix, there will only be a few non-zero diagonal vectors. We can therefore represent the complete linear equations in terms of several vectors, including the right-hand vector $\boldsymbol{f}$, which will naturally be the input of the DNN model.

With this approach, we achieve the input of any multi-diagonal equation into the DNN model; thus, the DNN model can solve the linear equations of an arbitrary shape. However, in this paper, to simplify the description, we have chosen the tridiagonal equations as an example to illustrate the mechanism of the algorithm.

An arbitrary multi-diagonal matrix can have all its main diagonal elements transformed to 1 by a linear transformation, as shown by

$$\begin{aligned} &\left(\mathrm{diag}_0(d_1, d_2, \cdots, d_n) + \mathrm{diag}_1(r_1, r_2, \cdots, r_{n-1}) + \mathrm{diag}_{-1}(l_1, l_2, \cdots, l_n)\right)\boldsymbol{x} = \boldsymbol{f} \\ \Rightarrow &\left(I_n + \mathrm{diag}_1\left(\tfrac{r_1}{d_2}, \tfrac{r_2}{d_3}, \cdots, \tfrac{r_{n-1}}{d_n}\right) + \mathrm{diag}_{-1}\left(\tfrac{l_1}{d_1}, \tfrac{l_2}{d_2}, \cdots, \tfrac{l_{n-1}}{d_{n-1}}\right)\right)\boldsymbol{x} = \left(\tfrac{f_1}{d_1}, \tfrac{f_2}{d_2}, \cdots, \tfrac{f_n}{f_n}\right)^{\mathrm{T}}, \end{aligned} \tag{3}$$

where the original equation is represented by three diagonal vectors $\boldsymbol{d}$, $\boldsymbol{r}$, $\boldsymbol{l}$ and the right-hand vector $\boldsymbol{f}$. After the transformation shown in (3), the input vectors are reduced by one, and more significantly, the three input vectors are normalized.

The DNN model can so far be written in the form of $\mathcal{N}(\boldsymbol{r}, \boldsymbol{l}, \boldsymbol{f}; \Theta)$, where Θ is the set consisting of the parameters and hyperparameters of the DNN model. On the other hand, the DNN model can be considered as a regression of the operator $\boldsymbol{y} = \mathcal{S}(A, \boldsymbol{f}) = A^{-1}\boldsymbol{f}$ .We accordingly selected mean squared error (MSE) as the loss function $L(\cdot)$, shown by

$$L(\cdot) = \frac{1}{M}\sum_{i=1}^{M}\left(\mathcal{N}(\boldsymbol{r}_i, \boldsymbol{l}_i, \boldsymbol{f}_i; \Theta) - \mathcal{S}(A_i, \boldsymbol{f}_i)\right)^2, \tag{4}$$

where M is the size of the training set.

The training set we utilize consists of the random vectors of the Gaussian distribution

$$\begin{aligned} T_N(\rho) = \big\{\{\boldsymbol{r}_i, \boldsymbol{l}_i, \boldsymbol{f}_i, \boldsymbol{y}_i\} : \boldsymbol{r}_i \in \mathbb{R}^{N-1}; \boldsymbol{l}_i \in \mathbb{R}^{N-1}; \boldsymbol{f}_i \in \mathbb{R}^N; \\ \boldsymbol{y}_i = \left(I + \mathrm{diag}_1(\boldsymbol{r}_i) + \mathrm{diag}_{-1}(\boldsymbol{l}_i)\right)^{-1}\boldsymbol{f}_i; r_{ij}, l_{ij}, f_{ij} \sim N\left(0, \rho^2\right), j \le N\big\} \end{aligned} \tag{5}$$

where ρ is the standard deviation of the training set and is treated as a given parameter in establishing the training set. It is worth noting that, although the training process of the DNN model is time-consuming, it is done offline[10] and only once for equations of the same size.

The residual network architecture. There are kinds of particular problems of classic deep networks[28,30,31] that directly constrain the precision of the predicted solution, e.g., vanishing gradient, exploding gradient, and network degradation[32–35]. Focusing the above problems, especially the network degradation, Res-Net establish a direct connection between the shallow layer and deep layer imitating a short circuit. Res-Net is thus capable to fit the identity operator and forces the network to approximate the "residue" of the input–output map effectively[30,36].

To prevent the vanishing gradient, we employ ReLU (rectified linear unit) as the activation function in the hidden layers. We denote the weight and bias of a single hidden layer as w and $\boldsymbol{b}$, the residual block $\mathcal{L}(\cdot)$ can be represented as

$$\begin{cases} F(\boldsymbol{x}) = ReLU(w\boldsymbol{x} + \boldsymbol{b}), \\ L(\boldsymbol{x}) = F(\boldsymbol{x}) + x. \end{cases} \tag{6}$$

The DNN model $\mathcal{N}$ can be denoted by a composite mapping, which forms

$$\mathcal{N}(\boldsymbol{r}, \boldsymbol{l}, \boldsymbol{f}; \Theta) = \mathcal{L}_N \circ \mathcal{L}_{N-1} \circ \cdots \circ \mathcal{L}_1, \tag{7}$$

where $\circ$ stands for operator composition.

In addition to the Res-block, we utilize another short connection covering the whole network model. Based on this short connection, the network can be comprehended as the approximation of $(I - A^{-1})\boldsymbol{f}$, that the output vector has smaller values and a zero mean. To evaluate the practical optimization via Res-Net, the training loss comparison between the DNN models with and without Res-Net is shown in Fig. 1.

From Fig. 1, the training loss of the DNN model containing Res-Net decreases to 1/4500–1/5000 of the original model. Moreover, the optimization of Res-Net is even more significant for large-scale linear equations. Therefore, Res-Net can be considered a reliable method to improve the precision of the DNN model.

Based on the above architecture, we designed the DNN model as shown in Fig. 2. In this model, there are three input vectors whose total dimension is $(3N - 2)$, while the dimension of the output vector is N. We thus established four groups of Res-blocks, three of which receive the input $\boldsymbol{r}$, $\boldsymbol{l}$, and $\boldsymbol{f}$. After concatenation, the output of these three groups is fed into the left Res-block and the final output $\boldsymbol{x}$ is achieved after adding $\boldsymbol{f}$.

In Fig. 2, both the number of the Res-blocks (the depth of the network) and the size of the weight matrix (the number of the neurons) in each Res-block should be carefully considered based on the balance of the computational consumption and the precision. In this manuscript, we select the scheme that all the Res-block groups consist of three Res-blocks, i.e., $n = m = 3$ in Fig. 2. The number of the neurons is 100 in the group $\boldsymbol{l}_i, \boldsymbol{r}_i$ and $\boldsymbol{f}_i$, and 200 in the group $\boldsymbol{c}_i (i \leq 3)$. This scheme is a compromise choice based on the numerical tests.

A correction iteration. Theoretically, based on the universal approximation theorem[17,18], the network can approximate the objective function with arbitrary accuracy when the number of neurons and layers of the neural network is sufficiently large. However, an excessively large model is not acceptable for actual computation. Therefore it is impracticable to obtain an accurate solution to an arbitrary equation via an individual prediction. In addition to the Res-Net, we employed another approach to improve the numerical precision. Inspired by the classic iteration methods, e.g., CG and GMRES, we proposed a correction algorithm shown in Algorithm 1.

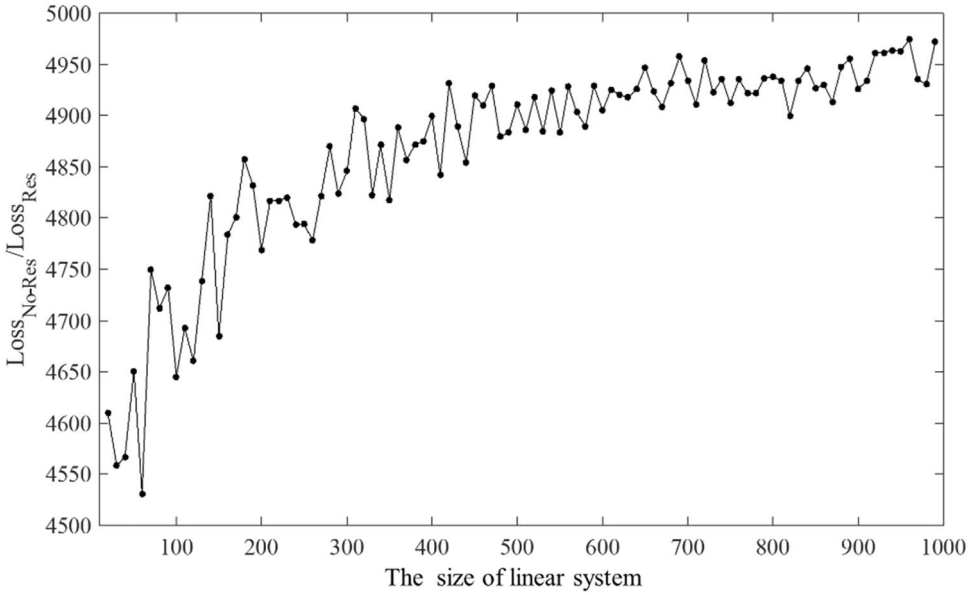


Figure 1. The ratio of the training loss of Res-Net and non-Res-Net at different sizes.

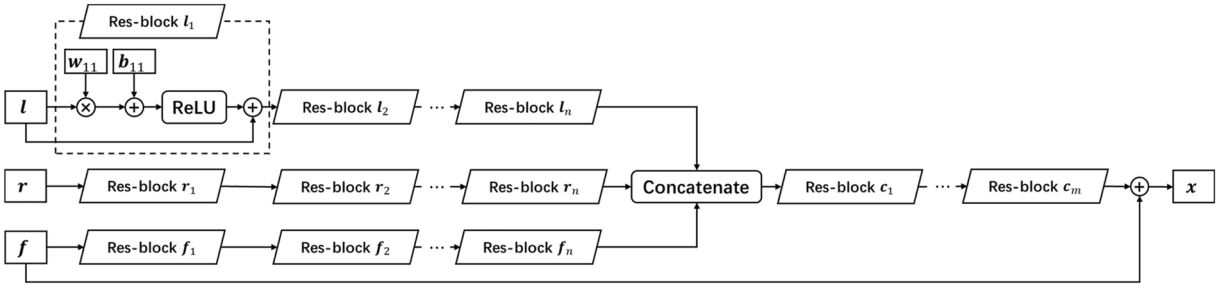


Figure 2. The schematic of the DNN model.

Algorithm 1 Correction routine for the coarse solution

$\boldsymbol{x^0 = \mathcal{S}(A, f)}$, $\boldsymbol{it = 1}$, $\boldsymbol{\|c^0\| = 1}$, $\boldsymbol{\|c^1\| = 0}$
while $\boldsymbol{it < N_{max}}$ **and** $\boldsymbol{\|c^{it}\| \le \varepsilon \|c^{it}\|}$
$\boldsymbol{M = [0 \quad l]^{\mathrm{T}} \odot [0 \quad x^{it}_{1\sim n-1}]^{\mathrm{T}} + [r \quad 0]^{\mathrm{T}} \odot [x^{it}_{2\sim n} \quad 0]^{\mathrm{T}} + x^{it}}$
$\boldsymbol{c^{it} = \mathcal{S}(A, f - M)}$
$\boldsymbol{c^{it+1} = x^{it} + c^{it}}$
$\boldsymbol{it = it + 1}$
end while

In Algorithm 1, ε is the parameter that determines the iterative convergence criterion, which is a constant smaller than 1. The other parameter restricts the iteration number is the maximum iteration number N_{max}. The symbol $\boldsymbol{x}_{a\sim b}(a, b \in \mathbb{N}_+)$ denotes the vector consisting of the a-th to b-th elements of $\boldsymbol{x}$. The operator $\odot$ in Algorithm 1 indicates the elementwise product of two vectors, i.e., $[\, a_1 \ \cdots \ a_n \,] \odot [\, b_1 \ \cdots \ b_n \,] = [\, a_1 b_1 \ \cdots \ a_n b_n \,]$. It can be proven that the matrix M is the matrix–vector product $A\boldsymbol{r}^{it}$.

In the calculation of matrix M, the tridiagonal matrix and vector production can be considered a sum of three vector operations that minimize the computational complexity and memory usage. In addition, the effect of the iterative algorithm to improve the accuracy will be evaluated by the numerical results proposed in Chapter 3.

Precision-oriented parametric analysis

In general, the numerical precision of the proposed method could be influenced by two aspects of the input: the properties of the linear system and the parameters of the algorithm. It has been mentioned in section The representation of the linear equation that the condition number of the matrix is significantly related to the diagonal dominance. In contrast, the condition number can reflect the sensitivity of the solution. The performance of the DNN model is consequently affected by diagonal dominance. Moreover, the algorithm's relative precision (the norm ratio between the solution and error) is another essential benchmark that we evaluate in this section.

To quantify the effect of the above parameters on the precision and the convergence rate in the correction iteration, we conducted a series of numerical experiments proposed in this section. Specifically, we utilized 5000 sets of input vectors generated by standard random numbers for each set of parameters to collect the mean results. In each set of parameters, ρ_c and ρ_r represents the ρ of the off-diagonal vector and right-hand vector, respectively.

The influence of the input vectors. The off-diagonal vectors affect the diagonal dominance of the linear equation and the regularity of the approximated mapping. Therefore, with the increase of the off-diagonal vectors' norm, the DNN model's accuracy gradually deteriorates. The network would become invalid after reaching an uncertain threshold.

On the other hand, for a fixed matrix A, the mapping between the right-hand vector and solution is linear. Consequently, the norm of the solution is approximately proportional to the norm of the right-hand vector. A well-trained network should achieve high accuracy and simulate this linearity. Thus, the norm of the error would be approximately proportional to the norm of the right-hand vector. Figure 3 demonstrates the effect of the off-diagonal and the right-hand vectors on the network, respectively.

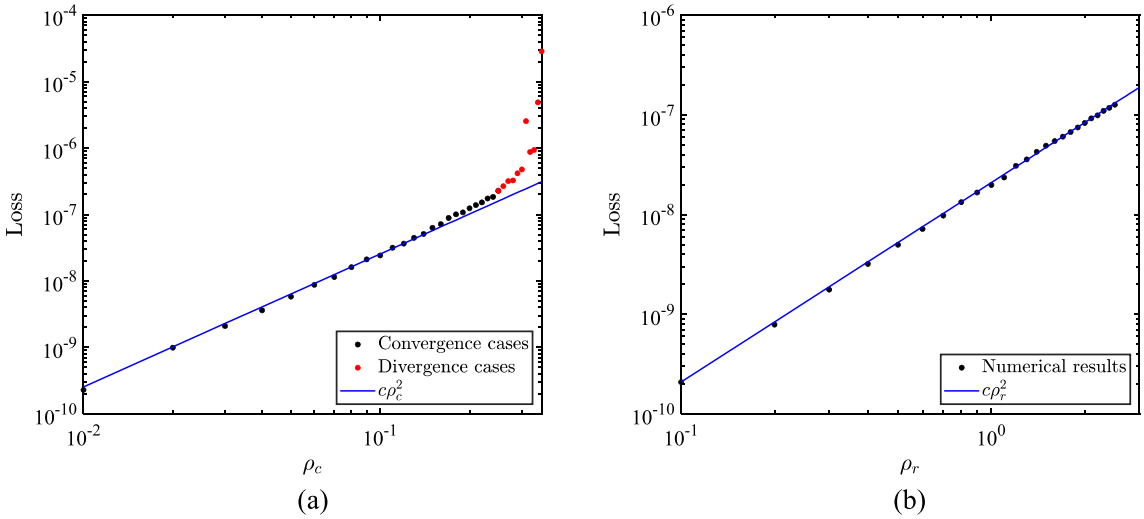


Figure 3. The influence of the off-diagonal elements on the training loss (Left: the influence of ρ_c when $\rho_r = 0.1$; Right: the influence of ρ_r when $\rho_c = 0.1$).

In Fig. 3, training loss represents the final value of the loss function in the training process. To investigate the effect of the single parameter, ρ_r in Fig. 3a is set to 0.1 in ,and ρ_c in Fig. 3b is set to 0.01. In Fig. 3a, the training loss is significantly related to ρ_c, and the training process shows the phenomenon of non-convergence as ρ_c increases to more than 0.3. Therefore, to achieve a reliable predicted solution, diagonal dominance is one of the essential factors that need to be considered.

According to the results shown in Fig. 3b, DNN can be considered as a regression with an approximately constant relative accuracy, i.e., the ratio $\|y^* - y\|/\|y\|$ is nearly a constant with a given network. In the view of the relative accuracy, Fig. 4 presents the relative accuracy of the network.

In Fig. 4, the relative error calculated from $\|y^* - y\|/\|y\|$ is approximately 2.53×10^{-8} and remains within a small range for different ρ_r. This demonstrates that the algorithm has an advantage in terms of numerical accuracy and is sufficient for the numerical solution of the PDE.

The amplification factor. Since the operator $\mathcal{S}(A, \boldsymbol{f}) = A^{-1}\boldsymbol{f}$ is linear to $\boldsymbol{f}$, the following equality must be valid for arbitrary constant α:

$$\frac{1}{\alpha}\mathcal{S}(A, \alpha\boldsymbol{f}) = \mathcal{S}(A, \boldsymbol{f}) \tag{8}$$

As a regression of $\mathcal{S}(A, \boldsymbol{f})$, the output of DNN must satisfy this property as well. Nevertheless, the DNN, as a nonlinear mapping, is incapable of this linear property but approximately meets it. We were inspired by this property to conceive a factor α, called the amplification factor, as a hyperparameter to improve the precision of the DNN model. In particular, the prediction of DNN $\mathcal{S}(A, \boldsymbol{f})$ is replaced by $\mathcal{S}(A, \alpha\boldsymbol{f})/\alpha$. According to the numerical results in section "The influence of the input vectors", the constant α can be comprehended as an amplification factor of the prediction error and drive the correction iteration to correct it easier. The comparison among the iteration processes with different α is shown in Fig. 5.

In Fig. 5, as the amplification factor α gradually increases, the error in iterative convergence decreases significantly. However, the increase of α beyond a certain threshold has approximately no effect on the precision. Besides, for the given set of input vectors, α has little impact on the speed of convergence; thus, larger α also causes more iterative steps. To further demonstrate the effect of the hyperparameter α, Fig. 6 shows the residual norm and the number of iterations with different α.

As can be observed in Fig. 6a, the influence of the off-diagonal elements on the precision has been approximately eliminated when the off-diagonal elements are smaller than the threshold ($\rho_c < 0.2$). Combining the results in Fig. 3, we can assert that for diagonal-dominated equations, the correction iteration practically eliminates the effect of diagonal dominance on precision. Nevertheless, for non-diagonal dominant equations ($\rho_c > 0.3$), the correction iteration inevitably fails to converge due to the large condition number. The amplification factor does not produce any effect in this case. Figure 7b shows that the number of iterations increases as the amplification factor increases until the iterations fail to achieve convergence. The results further demonstrate that the amplification factor practically amplifies the error of the DNN prediction and improves the numerical precision while causing an increase in the number of iterations. The number of iterations for the non-diagonal dominant equations is small due to the iterative divergence.

Numerical experiments and validation

Computational efficiency. To demonstrate the high efficiency of the proposed method, we compare it against the LU method in this section, as shown in Fig. 7.

The current version of our solver is developed based on the TensorFlow platform[37], while the classic solvers in this comparison are provided by NumPy[38] library, SciPy[39] library (for CPU implementation), and CuPy library (for GPU implementation). On the homogeneous (CPU) platform, both the NumPy-based solver and SciPy solver natively support parallel computation. Since GPU devices can accelerate the TensorFlow platform, we utilized the CuPy-based solvers as the classic solvers on the GPU platform for a fair comparison. In this test,

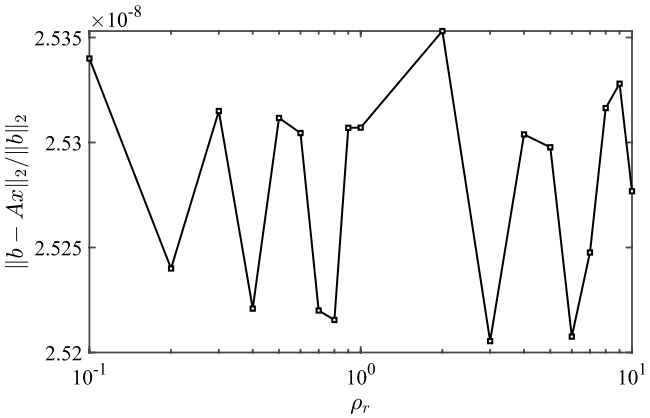
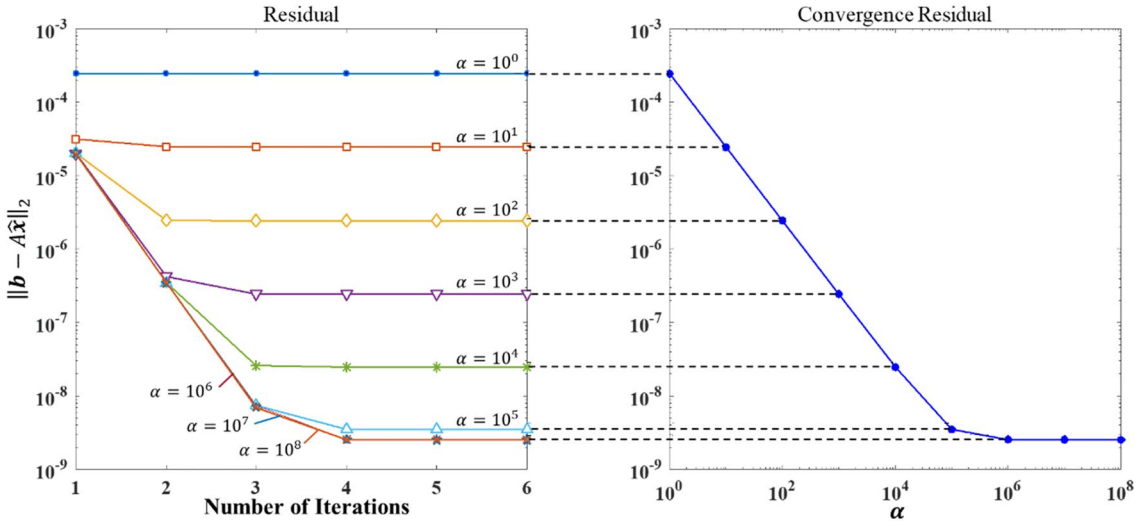
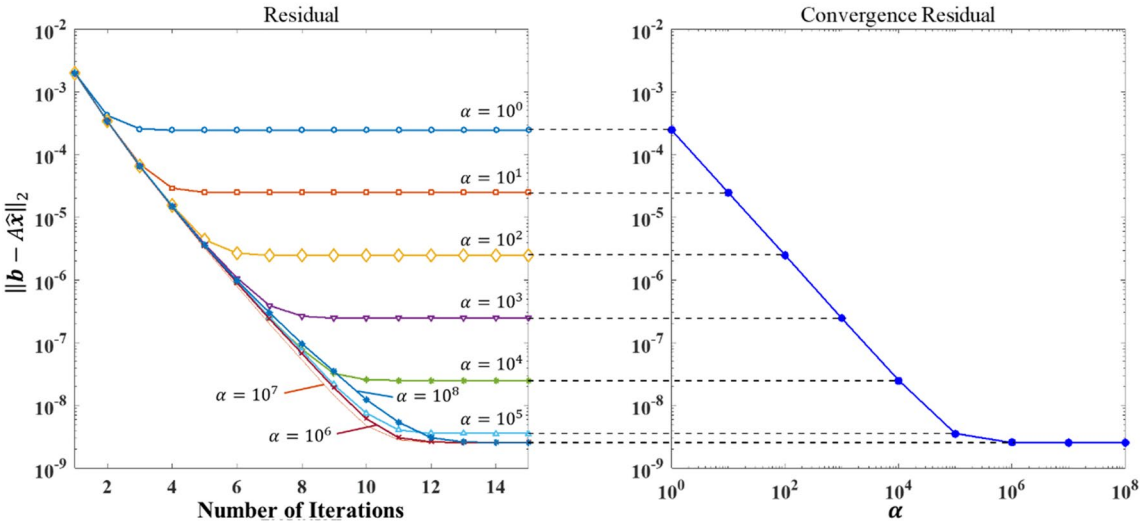


Figure 4. The relative error for different ρ_r ($\rho_c = 0.01$).



(a) $\rho_c = 0.01$



(b) $\rho_c = 0.1$

Figure 5. The norm of the residual vs. different α (Left: the decreasing of the residual in each iteration; Right: the residuals with different α).

the NumPy-based solver is based on the LU algorithm, the SciPy-based solver utilizes the Least Square QR-factorization (LSQR) algorithm, and both of the algorithms are implemented by the CuPy-based solver.

The hardware platform in this test is Intel Xeon-W3175X CPU (28 cores, 3.1 GHz) and Nvidia RTX2080Ti GPU. In every single test, we solved 5000 random linear equations of a given size and calculated the average solving time.

In Fig. 7, as the size of the matrix increases, the computation time of the proposed DNN-based algorithm grows much slower than other classic solvers, and for matrices of size over 1000, this algorithm can achieve an acceleration of 10 to 100 times compared with the CPU-implemented algorithms.

The CuPy-based LU algorithm performs best in small equations but worst in large ones due to the bandwidth constraint between the CPU and GPU. The CuPy-based LSQR algorithm keeps most of the computation sub-routines on the GPU, thus having advantages over the CPU-implemented algorithms. However, as the matrix increases, its time increases faster than the DNN-based solver; therefore, the DNN-based algorithm is more suitable for solving large-scale problems.

The heat conduction equation. In order to further highlight the high efficiency and accuracy of our algorithm, a set of PDEs is solved numerically using the proposed method. The first case is a 2D heat-conduction equation with mixed boundary conditions on a rectangle area shown by

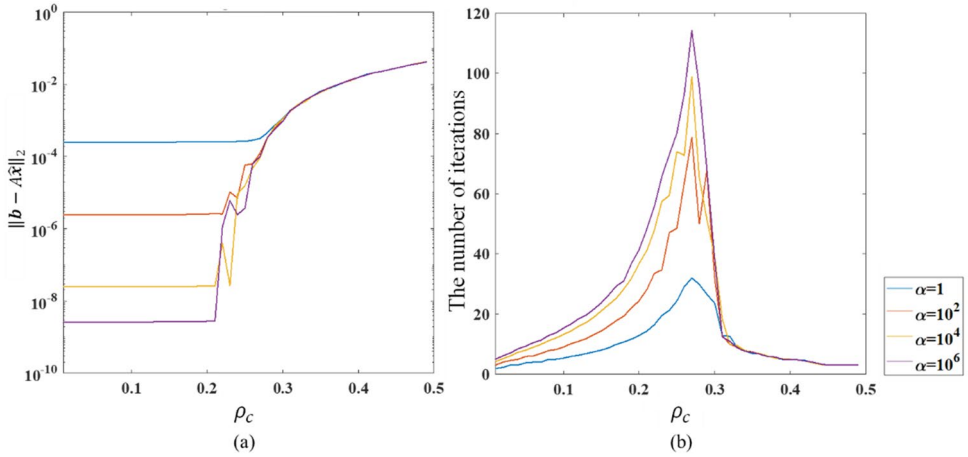


Figure 6. The effects of ρ_c on convergence error (**a**) and the number of iterations (**b**).

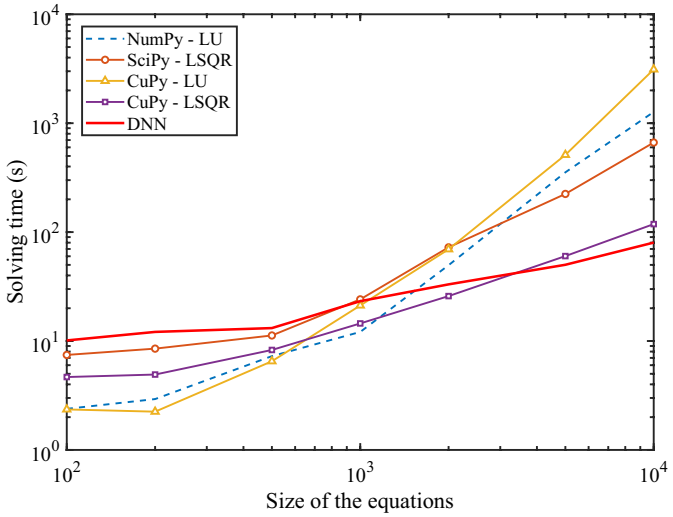


Figure 7. Comparison of the computation time among different solvers.

$$\begin{cases} \partial_t u - \beta\left(\partial_{xx}^2 u + \partial_{yy}^2 u\right) = 0, (x, y) \in \Omega \\ \frac{\partial u}{\partial n}|_{y=0,1} = 0 \\ u|_{x=0} = 0, u|_{x=1} = 1 \end{cases} \tag{9}$$

where the problem domain $\Omega = \{(x, y)|0 \le x \le 1, 0 \le y \le 1\}$,$\beta$ is the conduction coefficient, assumed as a constant in this problem.

In the present case, the parameters of the simulation are shown in Table 1.

The ADI scheme with 2nd order spatial precision and 1st temporal precision is utilized, which forms

Parameter	Value
β	0.1
Δt	0.001
Δx	0.0025
Δy	0.0025
T	1.0

Table 1. Parameters of the 2D heat-conduction equation.

$$\begin{cases} u_{i,j}^{n+\frac{1}{2}} - \frac{1}{\tau_x^+ \Delta x^2} u_{i+1,j}^{n+\frac{1}{2}} - \frac{1}{\tau_x^+ \Delta x^2} u_{i-1,j}^{n+\frac{1}{2}} = \frac{1}{\tau_x^+ \tau_y^-} u_{i,j}^{n} + \frac{\beta}{\tau_x^+ \Delta y^2} \left(u_{i,j+1}^{n} + u_{i,j-1}^{n} \right) \\ u_{i,j}^{n+1} - \frac{1}{\tau_y^+ \Delta y^2} u_{i,j+1}^{n+1} - \frac{1}{\tau_y^+ \Delta y^2} u_{i,j-1}^{n+1} = \frac{1}{\tau_y^+ \tau_x^-} u_{i,j}^{n+\frac{1}{2}} + \frac{\beta}{\tau_y^+ \Delta x^2} \left(u_{i+1,j}^{n+\frac{1}{2}} + u_{i-1,j}^{n+\frac{1}{2}} \right) \end{cases} \tag{10}$$

where $\tau_x^{\pm} = 2\left(\Delta t^{-1} \pm \Delta x^{-2}\right)^{-1}$. The tridiagonal linear equation can represent each semi-step, and the proposed method can thus solve the linear equation. The comparison between the numerical solutions at the first 200 timesteps is shown in Fig. 8.

To demonstrate the error distribution across the field more clearly, we have chosen the x-u section to illustrate the error distribution in Fig. 8. Although the LU linear solver achieves the reference solution, all the classic solvers mentioned in Fig. 7 can achieve an approximately precise solution as well. The error across the area is approximately 10^{-6}, in the same order of magnitude as the discretization error of the difference scheme in Eq. (8).

The convection–diffusion equation. This section presents a nonlinear problem solved by the proposed method. The Burgers equation is one of the fundamental PDEs in various fields such as nonlinear acoustics, gas dynamics, fluid mechanics, etc.[40]. The Burgers equation was first introduced by H. Bateman[41] and later studied by J. M. Burgers[42] in the theory of turbulence. Considering the nonlinear Burgers equation shown by

$$\begin{cases} \partial_t u - u\partial_x u - \beta \partial_{xx}^2 u = 0, x \in [0,1] \\ u|_{x=0,1} = 0, \\ u|_{t=0} = \sin 2\pi x, \end{cases} \tag{11}$$

where β is the factor of diffusion.

The difference scheme with 2nd precision we selected is shown by

$$\begin{aligned} \left(1 - \alpha\left(u_{i+1}^n - u_{i-1}^n\right) + 2\widehat{\beta}\right) u_i^{n+1} + \left(\alpha u_i^n - \widehat{\beta}\right) u_{i-1}^{n+1} - \left(\alpha u_i^n + \widehat{\beta}\right) u_{i+1}^{n+1} \\ = u_i^n + \widehat{\beta}\left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right), \end{aligned} \tag{12}$$

where the constant $\alpha = 1/4\Delta x$, $\widehat{\beta} = \beta \Delta t / 2\Delta x^2$ the parameter is set as time step $\Delta t = 0.001$, spatial step $\Delta x = 1/400$, $\beta = 0.25$.

Analytic solutions exist for this equation; thus, the numerical results can be employed to evaluate the precision of both the differential format and the proposed algorithm. Figure 9 shows the numerical results and error distribution in the first 500 timesteps.

In Fig. 9, the error of both two algorithms across the entire field is approximately 10^{-7}, as shown in Fig. 9c, while the discretization error of the difference scheme is approximately 10^{-5}, as shown in Fig. 9b.

Both linear and nonlinear equation examples illustrate the high precision of the proposed method, while most other NN-based solvers, including PINN[20], hp-VPINN[21], etc.[26,43], have errors in the magnitudes range of 10^{-2} to 10^{-3}.

Conclusion

This paper introduces a DNN-based algorithm to solve the linear equation with high efficiency and precision. The algorithm combines the Res-Net architecture and the correction iteration method.

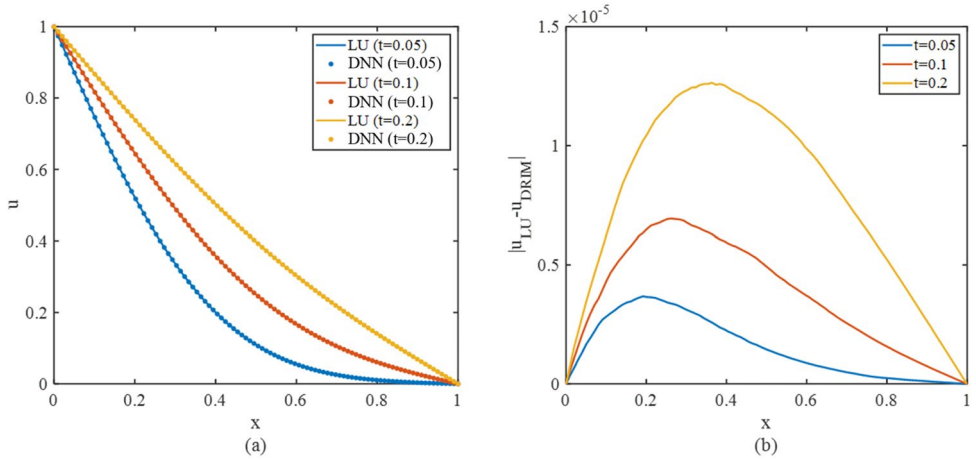


Figure 8. The numerical solution to the heat conduction equation at y = 0.5 (**a**) and the error distribution (**b**).

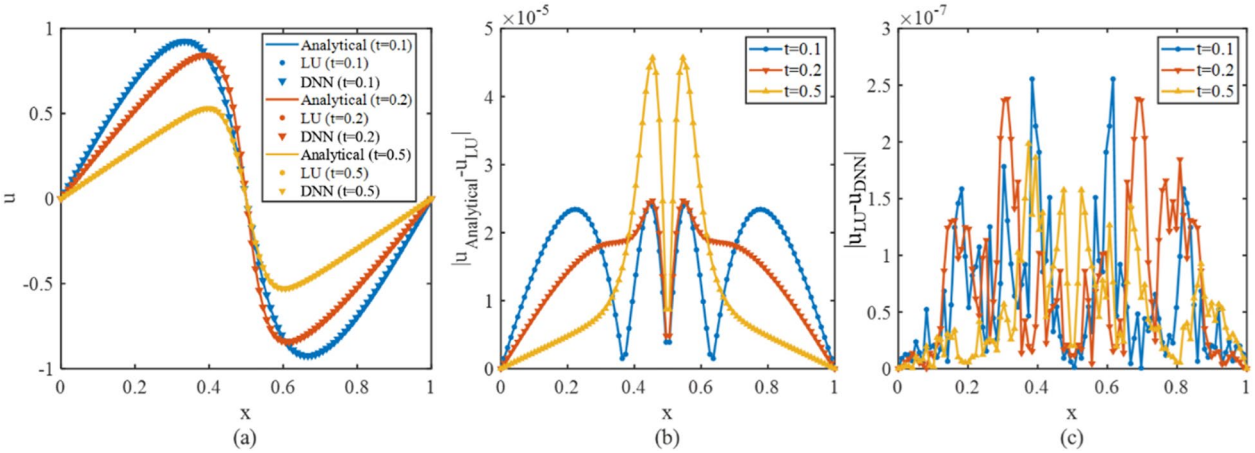


Figure 9. The numerical and the analytical solution to the convection–diffusion equation (**a**) and the error distribution (**b**,**c**).

1. Based on the DNN model, the proposed method has high computation efficiency and the native hardware compatibility on GPU and other hybrid platforms. Compared with classic methods, this method can simultaneously solve numerous linear equations with a low computation complexity.
2. We employed the Res-Net architecture to avoid the network degradation of the DNN and consequently improve the precision of the DNN model. The proposed method also includes the correction iteration to reduce the error further to achieve acceptable algorithm precision iteratively.
3. We verify the proposed method's reliability, numerical precision, and computational efficiency by applying it to solve practical PDEs (including linear and nonlinear equations) in this paper. In addition, we provide several numerical results to evaluate the numerical precision of the proposed method and its affecting factors, e.g., the diagonal dominance of the input matrix.
4. Like the other algorithms based on the DNN, the performance of the proposed method is also significantly affected by the factors of input variables. We investigate some typical elements, e.g., the diagonal dominance and the norm of the right-hand vector, based on the numerical results.
5. Inspired by the properties of the analytical solution of the linear equation, we conceive the amplification factor to improve the precision of the proposed method. The amplification factor can efficiently reduce the error of the solution and eliminate the effect of the diagonal dominance but causes more iteration steps.
6. According to the numerical results, including a nonlinear problem and linear problem, the proposed method is reliable in solving the PDEs numerically with high computational efficiency and sufficient precision.

For a long time, accuracy and interpretability have been fatal problems in exploring the DNN for applications in numerical computation. The method proposed in this research provides an innovative idea with a reference value. However, the currently proposed method still has some limitations, including the inability to solve non-diagonally dominated equations.

Some limitations still exist in the current version of the algorithm, such as difficulties with rigid problems. In our subsequent research, we will improve the proposed algorithm to address these limitations and continue to investigate the combination of the proposed DNN models with classic algorithms, e.g., the preconditioner based on the proposed method. Moreover, the proposed method has been proven effective in accelerating the solving process of multi-diagonal equations. We will continually explore its application to three-dimensional problems and other PDEs.

Data availability

The training data set in the manuscript is composed of random numbers (the generating method is introduced in section "Methodology" of the manuscript). The trained model we obtained will be submit as attachment (h5 file), which is also the data file needed to achieve the results of this manuscript. The other data used or analysed during the current study available from the corresponding author on reasonable request.

Received: 12 November 2022; Accepted: 8 March 2023
Published online: 18 March 2023

References

1. Niki, H., Kohno, T. & Morimoto, M. The preconditioned Gauss-Seidel method faster than the SOR method. *J. Comput. Appl. Math.* **219**(1), 59–71 (2008).
2. Golub, G. H. & Van Loan, C. F. *Matrix Computations* Vol. 3 (JHU Press, 2012).
3. G.W.S. Templates for the solution of linear systems: building blocks for iterative methods. *Math. Comput.* **64**(211), 1349–1352 (1995).
4. Fedkiw, R., Stam, J., & Jensen, H. W. Visual simulation of smoke. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*,pp. 15–22. (Association for Computing Machinery, 2001).

5. Alshemali, B. & Kalita, J. Improving the reliability of deep neural networks in NLP: A review. *Knowl.-Based Syst.* **191**, 19 (2020).
6. Zhang, X. Y. *et al.* Accelerating very deep convolutional networks for classification and detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **38**(10), 1943–1955 (2016).
7. Xiao, Y. *et al.* Construction of a new automatic grading system for jaw bone mineral density level based on deep learning using cone beam computed tomography. *Sci. Rep.* **12**(1), 12841 (2022).
8. Sebastian, A. *et al.* Revealing low-temperature plasma efficacy through a dose-rate assessment by DNA damage detection combined with machine learning models. *Sci. Rep.* **12**(1), 18353 (2022).
9. Lagaris, I. E., Likas, A. & Fotiadis, D. I. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. Neural Networks* **9**(5), 987–1000 (1998).
10. Ray, D. & Hesthaven, J. S. An artificial neural network as a troubled-cell indicator. *J. Comput. Phys.* **367**, 166–191 (2018).
11. Chan, S. & Elsheikh, A. H. A machine learning approach for efficient uncertainty quantification using multiscale methods. *J. Comput. Phys.* **354**, 493–511 (2018).
12. Wang, Y. *et al.* Deep multiscale model learning. *J. Comput. Phys.* **406**, 109071 (2020).
13. Mardt, A. *et al.* VAMPnets for deep learning of molecular kinetics. *Nat. Commun.* **9**(1), 5 (2018).
14. Chen, R. T. Q., et al. *Neural Ordinary Differential Equations*. arXiv e-prints (2018).
15. Long, Z., et al. *PDE-Net: Learning PDEs from Data*. arXiv e-prints (2017).
16. Khoo, Y., Lu, J. & Ying, L. *Solving Parametric PDE Problems with Artificial Neural Networks*. arXiv e-prints (2017).
17. Cybenko, G. Approximation by superpositions of a sigmoidal function. *Math. Control Signals Syst.* **2**(4), 303–314 (1989).
18. Hornik, K., Stinchcombe, M. & White, H. Multilayer feedforward networks are universal approximators. *Neural Netw.* **2**(5), 359–366 (1989).
19. Ezzinbi, K. & Fu, X. Existence and regularity of solutions for some neutral partial differential equations with nonlocal conditions. *Nonlinear Anal. Theory Methods Appl.* **57**(7), 1029–1041 (2004).
20. Raissi, M., Perdikaris, P. & Karniadakis, G. E. Machine learning of linear differential equations using Gaussian processes. *J. Comput. Phys.* **348**, 683–693 (2017).
21. Kharazmi, E., Zhang, Z. & Karniadakis, G. E. M. hp-VPINNs: Variational physics-informed neural networks with domain decomposition. *Comput. Methods Appl. Mech. Eng.* **374**, 113547 (2021).
22. Ew, Y. B. The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems. *Commun. Math. Stat.* **6**(1), 1–12 (2018).
23. Chen, X. Y. *et al.* A compressed lattice Boltzmann method based on ConvLSTM and ResNet. *Comput. Math. Appl.* **97**, 162–174 (2021).
24. Weymouth, G. D. Data-driven multi-grid solver for accelerated pressure projection. *Comput. Fluids* **246**, 1 (2022).
25. Saad, Y. & Schultz, M. H. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.* **7**(3), 856–869 (1986).
26. Xiao, X. *et al.* A novel CNN-based poisson solver for fluid simulation. *IEEE Trans. Visual Comput. Gr.* **26**(3), 1454–1465 (2020).
27. He, C., Ma, M. & Wang, P. Extract interpretability-accuracy balanced rules from artificial neural networks: A review. *Neurocomputing* **387**, 346–358 (2020).
28. He, K., et al., *Deep Residual Learning for Image Recognition.* arXiv e-prints: arXiv:1512.03385 (2015).
29. Vaswani, A. *et al.* Attention is all you need. In *Advances in Neural Information Processing Systems 30* (eds Guyon, I. *et al.*) (Neural Information Processing Systems (Nips), La Jolla, 2017).
30. Qin, T., Wu, K. & Xiu, D. Data driven governing equations approximation using deep neural networks. *J. Comput. Phys.* **395**, 620–635 (2019).
31. Chang, B., et al. *Multi-level Residual Networks from Dynamical Systems View*. arXiv e-prints (2017).
32. Bengio, Y., Simard, P. & Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks* **5**(2), 157–166 (1994).
33. Glorot, X., & Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *13th International Conference on Artificial Intelligence and Statistics, AISTATS 2010, May 13, 2010 - May 15, 2010*. (Microtome Publishing, Sardinia, Italy, 2010).
34. He, K., & Sun, J. Convolutional neural networks at constrained time cost. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015).
35. Srivastava, R. K., Greff, K., & Schmidhuber, J. *Highway Networks*. arXiv e-prints (2015).
36. Li, H., et al. *Visualizing the Loss Landscape of Neural Nets*. arXiv e-prints (2017).
37. Abadi, M., et al., *Tensorflow: Large-scale machine learning on heterogeneous distributed systems* (2016).
38. Harris, C. R. *et al.* Array programming with NumPy. *Nature* **585**, 357–362 (2020).
39. Pauli, V. *et al.* SciPy 1.0: Fundamental algorithms for scientific computing in python. *Nat. Methods* **17**, 261–272 (2020).
40. Whitham, G. B. *Linear and Nonlinear Waves* Vol. 42 (Wiley, 2011).
41. Bateman, H. Some recent researches on the motion of fluids. *Mon. Weather Rev.* **43**, 163 (1915).
42. Burgers, J. M. A Mathematical model illustrating the theory of turbulence. In *Advances in Applied Mechanics* (eds Von Mises, R. & Von Kármán, T.) 171–199 (Elsevier, 1948).
43. Piscopo, M. L., Spannowsky, M. & Waite, P. Solving differential equations with neural networks: Applications to the calculation of cosmological phase transitions. *Physical Review D* **100**(1), 12 (2019).

Acknowledgements

This work was partly supported by the National Key R&D program for international cooperation Grant No. 2018YFE9103900, and for Key issues of transformative science and technology, under Grant 2020YFA0712502. The Natural Science Foundation of China (NSFC), Grant 11972384, and Guangdong Science and Technology Fund, Grant 2021B1515310001, also supported this work.

Author contributions

Z.J., Q.Y. and G.Y. wrote the main manuscript. Z.J. and J.J. developed the NN-based solver mentioned in the manuscript. All authors reviewed the manuscript.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1038/s41598-023-31236-0.

Correspondence and requests for materials should be addressed to Q.Y.