Applied Soft Computing 148 (2023) 110872

Contents lists available at ScienceDirect

Applied Soft Computing

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

A high resolution Physics-informed neural networks for high-dimensional convection–diffusion–reaction equations

Jiangong Pan [a], Xufeng Xiao [a,*], Lei Guo [b], Xinlong Feng [a]

[a] College of Mathematics and System Sciences, Xinjiang University, Urumqi 830046, PR China
[b] Computer Science and Engineering, University of Electronic Science and Technology, Chengdu 610054, PR China

ARTICLE INFO

MSC:
65N30
68T20
35B50
68W01

Keywords:
Deep learning network
Convection–diffusion–reaction equation
High resolution
Loss functions
Maximum principle

ABSTRACT

In practical problems, some partial differential equations defined in high-dimensional domains or complex surfaces are difficult to calculate by traditional methods. In this paper, a novel data-driven deep learning algorithm is proposed to solve high-dimensional convection–diffusion–reaction equations. The main idea of the method is to use the neural network which combines the physical characteristics of the equation to get high accuracy numerical solution. The proposed method not only avoids the high cost of mesh generation, but also effectively reduces the numerical oscillation caused by the domination of the convection. In addition, two types of loss functions are designed to force physical properties, such as the positivity or maximum principle of the solution. Various numerical examples are performed to demonstrate the validity and accuracy of the proposed method.

1. Introduction

The convection–diffusion–reaction equations (CDREs) describe the combination of convection, diffusion and reaction [1,2]. They are basic partial differential equations (PDEs) for material transport modeling. Their numerical solutions are the core of the research on numerical differential equations and computational fluid dynamics. CDRE has many important applications in various research fields. For example, it is necessary to establish and solve CDRE numerically to describe and simulate the material transfer in 3-dimensional (3D) space such as the spread of gas pollutants and oil in pipelines [3–6]. In addition, the related modeling is also required for the heat transfer on ultra-thin materials, the movement of surfactants on the interface, and chemotaxis on biofilms, which involves the CDRE on surfaces [3,7–9].

The 3D steady convection–diffusion–reaction equation: The CDRE is considered in this study as follow. Let Ω be a bounded polygonal domain, $\partial\Omega$ be piecewise smooth boundary of Ω in $\mathbb{R}^3$, and Γ^* be a regular open subset of $\partial\Omega$. We introduce **the 3D steady convection–diffusion–reaction equation** at first:

$$\begin{aligned} -\epsilon\Delta u + (\beta\cdot\nabla)u + \gamma u &= f, && \text{in } \Omega,\\ u &= g_d, && \text{on } \Gamma^*,\\ \frac{\partial u}{\partial n} &= g_n, && \text{on } \partial\Omega\setminus\Gamma^*, \end{aligned} \tag{1.1}$$

where u represents fluid velocity or substance concentration, the small constant $\epsilon > 0$ is the diffusion coefficient, the convection coefficient β is velocity field in Ω, the function $\gamma(\gamma(\boldsymbol{x}) \geq \gamma_0 > 0)$ corresponds to the reaction coefficient, f is the source term, g_d and g_n are the Dirichlet boundary condition and Neumann boundary condition, respectively.

The surface steady convection–diffusion–reaction equation: Let Γ be a connected and oriented surface in $\mathbb{R}^3$ with no boundary, $\mathcal{N}\subset\mathbb{R}^3$ be an open tube domain, and ϕ be a function in $C^2(\mathcal{N})$. Suppose the surface Γ is expressed as a zero-level set function form:

$$\Gamma = \{\boldsymbol{x} = (x,y,z)\in\mathcal{N} \mid \phi(\boldsymbol{x}) = 0\}, \tag{1.2}$$

where the function ϕ has a property that $\nabla\phi(\boldsymbol{x}) \neq 0$ with ∇ being the standard gradient operator in $\mathbb{R}^3$. **The surface steady convection–diffusion–reaction equation** is introduced as follow:

$$-\epsilon\Delta_\Gamma u + (\beta\cdot\nabla_\Gamma)u + \gamma u = f, \quad \text{in } \Gamma, \tag{1.3}$$

where the unknown function u usually represents the heat or the concentration of a pollutant on Γ. The three coefficients ϵ, β, and γ are the same as in (1.1), and f is the source term, $\nabla_\Gamma = (\mathcal{I} - n(\boldsymbol{x})n^T(\boldsymbol{x}))\nabla$, $\nabla_\Gamma\cdot = \mathrm{trace}(\nabla_\Gamma)$, $\Delta_\Gamma = \nabla_\Gamma\cdot\nabla_\Gamma$, and $\mathcal{I}$ represent the surface gradient, surface divergence, Laplace–Beltrami, and identical operators, respectively. $n(\boldsymbol{x}) = \frac{\nabla\phi(\boldsymbol{x})}{|\nabla\phi(\boldsymbol{x})|}$ represents the unit normal vector field of Γ.

There are many discussions on the numerical methods for the above high-dimensional CDRE for an incomplete list [10,11]. Traditional finite element methods are extremely dependent on mesh in Ω and Γ. However, generating a mesh requires a high amount of computing

* Corresponding author.
E-mail addresses: mathpjg@sina.com (J. Pan), xiaoxufeng111@sina.com (X. Xiao), leiguo@uestc.edu.cn (L. Guo), fxlmath@xju.edu.cn (X. Feng).

https://doi.org/10.1016/j.asoc.2023.110872
Received 18 April 2022; Received in revised form 20 August 2023; Accepted 17 September 2023
Available online 29 September 2023

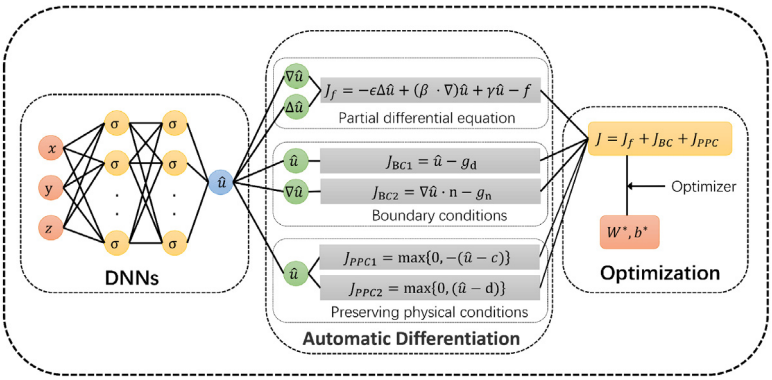

Fig. 1. DNNs for CDRE.

resources. In addition, if the true solution is not smooth and the mesh is sparse, numerical oscillation is likely to occur, and the extremum principle cannot be guaranteed when we are solving convection-dominated problems. We can effectively overcome the two problems by using the data-driven deep learning methods.

Nowadays, deep neural networks have been widely used in machine learning and have achieved remarkable success in the fields of computer vision and natural language processing [1]. Such networks have also been applied in solving PDEs [2,12–16] and developing a new field named Scientific Machine Learning (SciML) [17–19]. The neural network is a kind of complex nonlinear function composed of multi-layer neurons and activation functions. Its nonlinear expression ability increases with the increase of neurons and layers. Therefore, it is more advantageous to use a neural network to approximate the solution of PDEs. Exhilaratingly, the determination of neural networks is driven by data. Through the study of data, we can understand the mechanism and experience contained in the data. Therefore, we avoid mesh generation and only need to collect data points according to a random distribution. Randomly selecting the points reduces the cost of generating a mesh for complex problems. Furthermore, by designing a new loss function and adding an extreme constraint to the loss function, new methods can reduce the numerical oscillation, and the numerical solution with higher resolution is obtained. The essence of deep neural network function is a nonlinear function obtained by the nonlinear combination of multi-scale bases. Hence its expression ability is stronger than the traditional linear basis function, and it can obtain more accurate numerical solutions. To sum up, this novel method is worthy of further research for solving PDEs.

In recent years, a series of different machine learning frameworks have been proposed to solve PDEs. They are called as Finite-dimensional operators [20–23], Neural Operators [24–26], Neural-FEM [15,27]. Physics-informed neural networks (PINNs) are a kind of Neural-FEM [15,17,24]. They regard neural networks as solution surrogates and seek to find the best neural network guided by data and physical laws expressed as PDEs. These methods also have a wide range of applications in practical problems and have advantages over traditional methods [28–33].

In this paper, we propose a novel data-driven deep learning algorithm, which is based on PINNs, for solving the high-dimensional convection–diffusion–reaction equation. We have done the following:

- The proposed algorithm avoids the high cost of mesh generation and only collects data points randomly according to the given distribution.
- Since the deep neural network is a strongly nonlinear function, the proposed algorithm is better than the traditional piecewise linear function for approximating the solutions of partial differential equations.
- The proposed algorithm can effectively reduce the numerical oscillation in the strong convection-dominated problem.
- To satisfy the physical properties such as positive preserving and extremum preserving, two types of loss functions are designed to obtain different algorithms which are named as DLM_{3d}, $DLM_{3d,p}$ for the 3D problem, and DLM_{s}, $DLM_{s,p}$ for the surface problem.

The rest of this paper is organized as follows. Section 2, introduces the framework of deep neural networks and the deep learning method which we propose. Section 3 mainly describes the implementation details of the deep learning method. In Section 4, some numerical results are presented to verify the effectiveness of our method. Finally, we give a brief conclusion.

2. Deep learning method for convection–diffusion–reaction equation

In deep learning, the computational graph of neural network determines the function and significance of the whole neural network. For specific tasks, we design different neural networks. For example, deep convolution neural network is often used in image classification tasks, and recurrent neural network is usually used to solve natural language processing tasks and time series prediction tasks. Naturally, it is necessary to solve CRDE and design the computational graph according to its characteristics. Combined with the physical properties of the equations and the physical laws contained in it, a special computational graph of CRDE is designed, as shown in Fig. 1. The graph consists of three parts. The first part is the most basic deep neural networks (DNNs), as the core of approximation equations solution. In this paper, we use a fully connected neural network. In case of time-containing problems, recurrent neural networks can also be used. The use of neural network functions as approximation functions is similar to the use of piecewise linear units as approximation functions in standard finite elements. Different numerical results are obtained for cells of different orders in the finite element. Likewise, DNNs with different structures lead to different results, which we discuss later. Next, the purpose of the second part is to calculate the partial derivative of the output, which is called automatic differentiation (AD) and equivalent to the differential operator. The purpose of calculating derivatives is to compute various different loss functions (residuals of equations, residuals of boundary conditions, etc.). Combining the chain rule and the special structure of neural networks, it is possible to compute them quickly. The last part is to find the optimal solution by optimizing operator, and the numerical solution of CRDE has been obtained. For optimization problems, the use of different optimization algorithms can directly lead to differences in computational speed. Later we will consider the impact of several different optimization algorithms, for our problem.

Then, we use this graph to design a deep learning method (DLM) to solve CDRE. According to Eq. (1.1), we set up the data set $\mathcal{D}$:

$$\mathcal{D} = \mathcal{D}_f \cup \mathcal{D}_g, \tag{2.1}$$

where $\mathcal{D}_f = \{(\boldsymbol{x}_i, f_i)\}_{i=1,\dots,M}$ are generated from the source terms, $\mathcal{D}_g = \{(\boldsymbol{x}_j, g_j)\}_{j=1,\dots,N}$ are generated from boundary conditions and all x in $\mathcal{D}_f$ and $\mathcal{D}_g$ are random. Normally, data set $\mathcal{D}$ is divided into training set $\mathcal{D}_{train}$ and test set $\mathcal{D}_{test}$. Although we train the neural network on the training set $\mathcal{D}_{train}$, we pay more attention to the performance of neural network in the test set $\mathcal{D}_{test}$.

Then, select the relevant parameters of DNNs: number of hidden layers N_{hidden}, number of neurons per hidden layer N_{neuron}, number of training data $\mathcal{N}_{train}$, activation function σ, optimization algorithm (OA), learning rate η and number of training N_{epoch}.

Algorithm Deep learning method

Input: N_{hidden}, N_{neuron}, σ, $\mathcal{D}$, OA, η, N_{epoch}
Output: u^*_{DLM}
Build DNNs by N_{hidden}, N_{neuron} and σ
Initialize parameters (W^0_m, b^0_m) in DNNs, and obtain approximate solution u^0_{DLM}
for $i \le N_{epoch}$ **do**
Training process:
Input $\mathcal{D}_{train}$ into u^{i-1}_{DLM}, and get $u^{i-1}_{DLM}(\boldsymbol{x}_j)$, $j = 1, \cdots, |\mathcal{D}_{train}|$
Combine CDRE and $u^{i-1}_{DLM}(\boldsymbol{x}_j)$ to calculate the loss function J^i_{train}
Use back propagation algorithm, calculate $\frac{\partial J}{\partial W_m}|^i_{train}$, $\frac{\partial J}{\partial b_m}|^i_{train}$
Update W^i_m and b^i_m by OA, and obtain new approximation function u^i_{DLM}
Testing process:
Input $\mathcal{D}_{test}$ into u^i_{DLM}, and get $u^i_{DLM}(\boldsymbol{x}_j)$, $j = 1, \cdots, |\mathcal{D}_{test}|$
Combine CDRE and $u^i_{DLM}(\boldsymbol{x}_j)$ to calculate the loss function J^i_{test}
if $J^i_{test} \le J^{i-1}_{test}$ **then**
Save the optimal approximation solution $u^*_{DLM} = u^i_{DLM}$, $W^*_m = W^i_m$, $b^*_m = b^i_m$
else
Deal with different OA
Obtain optimal approximate solution u^*_{DLM}, and optimal weight W^*_m, b^*_m
Predict any point by u^*_{DLM}

Next, we introduce the details in Fig. 1: DNNs, Automatic differentiation, Optimization.

DNNs: Deep neural network is an artificial neural network with multiple layers between the input and output layers. There are different types of neural networks. We use a fully connected audit network, which always consists of the same components: neurons, weights, biases, and activation functions. These components functioning similar to the human brains and can be trained like any other machine learning algorithm. DNNs have great potential over traditional methods because they can model arbitrary non-linear relationships by data. Because of the advantages of the model generated by DNNs, the lower layer combination, a small number of units and activation functions can be used to build a nonlinear model for complex data.

In DNNs, we take the coordinates in the domain as the input and the solution u of the equation as the output. The ultimate goal is to use DNNs to approximate the solution of the equations. In Fig. 2, we represent M-layer deep neural networks by notation as follow:

$$u_{DNN}(\boldsymbol{x}) = \mathcal{N}_M \mathcal{N}_{M-1} \mathcal{N}_{M-2} \cdots \mathcal{N}_1 \mathcal{N}_0(\boldsymbol{x}), \tag{2.2}$$

where $\boldsymbol{x} = (x, y, z)^T$, $u_{DNN}(\boldsymbol{x}) : \mathbb{R}^3 \to \mathbb{R}$, $M > 0$ and

$$\begin{cases} \mathcal{N}_0(\boldsymbol{x}) = \boldsymbol{x} \in \mathbb{R}^{d_{in}}, \\ \mathcal{N}_m(\boldsymbol{x}) = \sigma(W_m \mathcal{N}_{m-1}(\boldsymbol{x}) + b_m), & for\ 1 \le m \le M-1, \\ \mathcal{N}_M(\boldsymbol{x}) = W_M \mathcal{N}_{M-1}(\boldsymbol{x}) + b_M \in \mathbb{R}^{d_{out}}. \end{cases} \tag{2.3}$$

$W_m, b_m (1 \le m \le M)$ are parameters to be trained, σ is given activation function. There are many commonly used activation functions. We introduce how to select the appropriate activation function in the next section.

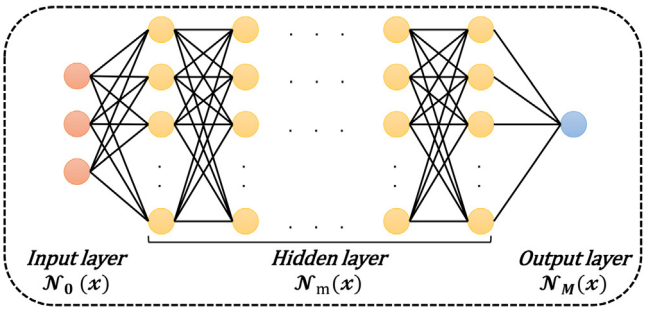

Fig. 2. M-layer deep neural networks.

Automatic differentiation: In mathematics and computer algebra, automatic differentiation is sometimes called differential calculus, which is a method to calculate the derivative of a function with the help of computer program. Two traditional differential methods are as follows:

(i) Differentiate the expression of a function on the sign, and calculate its value at a certain point.
(ii) Using the difference method.

The main disadvantages of using symbolic differentiation are its slow speed and difficulty in converting computer programs into expressions. The two important disadvantages of using difference are discarding error and cancellation error. These two traditional methods have the problems of complexity and error increase when calculating higher order differential. AD can solve the above problems.

AD uses the fact that a computer program for designing a vector function $y = F(x)$ can, in general, be decomposed into a sequence of basic operations, each of which can be differentiated by a basic function. According to the derivative rule of compound function, the differential information of F (such as gradient, tangent, Jacobian matrix, etc.) can be obtained by combining the differential of these basic functions. This process produces real (numerically accurate) derivatives. Because only at the most basic level to do symbol conversion, AD avoids the problem of complex symbolic operation.

The basis of AD is to combine the differential values according to the derivative rule of compound function. Taking $f(x) = g(h(x))$ as an example, according to the derivation rule of composite function, we have:

$$\frac{df}{dx} = \frac{dg}{dx} = \frac{dg}{dh}\frac{dh}{dx}. \tag{2.4}$$

There are usually two different modes: Forward Mode and Reverse Mode. Forward Mode uses the derivative rule of composite function from right to left, that is, $\frac{dh}{dx}$ is calculated first, then $\frac{dg}{dh}$ is calculated. Reverse Mode is from left to right.

In the neural network, there are basic expressions among the connected neurons. When using AD to calculate various differential operators (∇, Δ, etc.) in PDEs, it is just like a customized tool. Therefore, we add this crucial step in the design of the computational graph of CDRE.

Optimization: An optimization problem consists of maximizing or minimizing a real function by systematically choosing input values from within an allowed set and computing the value of the function. The generalization of optimization theory and techniques to other formulations constitutes a large area of applied mathematics. More generally, optimization includes finding "best available" values of some objective function given a defined domain, including a variety of different types of objective functions and different types of domains. In this step, by establishing a unique loss function as a guide, we use

the optimizer to find the optimal approximation function. The establishment of loss function and the selection of optimizer are described in detail in the next section.

3. Implementation details

In this section, we introduce some details of the algorithm implementation in detail to supplement the previous section. First of all, we explain the advantages and disadvantages of various activation functions σ in detail, and choose an appropriate σ for our method. Next, we design a core of our method: loss function. We design different loss functions according to the characteristics of high-dimensional CDRE. Then, we introduce a main optimization algorithms (OA) which are also called optimizers and analyze their advantages. Finally, we analyze the error sources of the proposed algorithm.

3.1. Activation function

Activation function is very important for neural network model to learn and understand very complex and nonlinear functions. If σ is not used, the output of each layer is a linear function of the input. Therefore, no matter how many layers the neural network has, the output is a linear combination of inputs, which is the most primitive perceptron. If used, σ introduces a nonlinear factor to the neuron, so that the neural network can approximate any nonlinear function, so that the neural network can be applied to many nonlinear models.

First, we introduce one of the most commonly used activation function in traditional neural networks. **Sigmoid function** is also called logistic function.

$$\sigma_S(x) = \frac{1}{1+e^{-x}}. \tag{3.1}$$

The output mapping of $\sigma_S(x)$ is between $(0, 1)$, which is monotone continuous and easy to derive. However, when derivative of $\sigma_S(x)$ is less than 1, the gradient disappear easily. Specifically, when x is very small or very large, this is a case where the derivative is very small. In addition, the main training method of neural network is back propagation algorithm, which is based on the chain rule of derivatives. The maximum derivative of $\sigma_S(x)$ is 0.25, and the result of multiplication is very small. With the deepening of neural network layers, when the gradient propagates backward to the shallow part of the network, it cannot cause parameter disturbance, that is, it does not transfer the loss information to the shallow part of the network, so the network cannot be trained. This is called gradient disappearance. In addition, the output of $\sigma_S(x)$ function is not zero centered, which leads to slow convergence of neural network.

Aiming at the shortcomings of $\sigma_S(x)$, a **rectified linear unit function** (ReLU) is proposed.

$$\sigma_R(x) = max\{x, 0\}. \tag{3.2}$$

$\sigma_R(x)$ effectively solves the problem of gradient disappearance, but it still fails to overcome the difficulty that the output is not zero centered. Although $\sigma_R(x)$ solves the problem of gradient disappearance, it is not suitable for computational graph in this paper. Because in the second section of the AD step, the second derivative of the $\sigma_R(x)$ function disappear.

In order to fit the computational graph in this paper, we use **hyperbolic tangent function**.

$$\sigma_T(x) = tanh(x). \tag{3.3}$$

$\sigma_T(x)$ whose output center is 0 and interval is $[-1, 1]$. $\sigma_T(x)$ can be imagined as two $\sigma_S(x)$ put together, and its performance is higher than that of $\sigma_S(x)$. Although the gradient of $\sigma_T(x)$ also disappears, it is suitable for AD and can effectively calculate the second derivative. In the future research, we continue to find and design better σ for solving PDEs.

3.2. Loss function

The loss function is a function in which the values of random variables are mapped to nonnegative real numbers to represent the risk or loss of random events. In application, the loss function is usually used as a learning criterion to solve and evaluate the model by minimizing the loss function. For different tasks, we need to establish different loss functions. In this section, we establish different loss functions for high-dimensional convection–diffusion–reaction equation and their physical properties.

We divide the loss function into three parts: one comes from the equation itself: $J_f(\boldsymbol{x}; W, b)$ and $J_{f,\Gamma}(\boldsymbol{x}; W, b)$, the other comes from the boundary conditions: $J_{BC}(\boldsymbol{x}; W, b)$, and the rest comes from physical conditions: $J_{PPC}(\boldsymbol{x}; W, b)$.

$$\begin{aligned}
J_f(\boldsymbol{x}; W, b) &= \frac{1}{N}\sum_{i=1}^{N}\big(-\Delta u(\boldsymbol{x}_i; W, b) + (\beta\cdot\nabla)u(\boldsymbol{x}_i; W, b) \\
&\quad + \gamma u(\boldsymbol{x}_i; W, b) - f_i\big)^2, \\
J_{f,\Gamma}(\boldsymbol{x}; W, b) &= \frac{1}{N}\sum_{i=1}^{N}\big(-\Delta_\Gamma u(\boldsymbol{x}_i; W, b) + (\beta\cdot\nabla_\Gamma)u(\boldsymbol{x}_i; W, b) \\
&\quad + \gamma u(\boldsymbol{x}_i; W, b) - f_i\big)^2, \\
J_{BC}(\boldsymbol{x}; W, b) &= \frac{1}{M_1}\sum_{j=1}^{M_1}(u(\boldsymbol{x}_j; W, b) - g_{d,j})^2_{\partial\Omega} \\
&\quad + \frac{1}{M_2}\sum_{j=1}^{M_2}(\nabla\cdot u(\boldsymbol{x}_j; W, b) - g_{n,j})^2_{\partial\Omega}, \\
J_{PPC}(\boldsymbol{x}; W, b) &= \frac{1}{N+M_2}\sum_{k=1}^{N+M_2}\max\{0, -(u(\boldsymbol{x}_k) + c)\} \\
&\quad + \frac{1}{N+M_2}\sum_{k=1}^{N+M_2}\max\{0, u(\boldsymbol{x}_k) - d\},
\end{aligned} \tag{3.4}$$

where $J_{PPC}(\boldsymbol{x}; W, b)$ represents that solution is restrained in $[c, d]$.

3.2.1. Loss function for 3D convection–diffusion–reaction equation

(I) 3D convection–diffusion–reaction equation:

$$J_{3d}(\boldsymbol{x}; W, b) = \lambda_1 J_f(\boldsymbol{x}; W, b) + \lambda_2 J_{BC}(\boldsymbol{x}; W, b). \tag{3.5}$$

(II) Positivity-preserving 3D convection–diffusion–reaction equation:

$$J_{3d,p}(\boldsymbol{x}; W, b) = \lambda_1 J_f(\boldsymbol{x}; W, b) + \lambda_2 J_{BC}(\boldsymbol{x}; W, b) + \lambda_3 J_{PPC}(\boldsymbol{x}; W, b). \tag{3.6}$$

3.2.2. Loss function for surface convection–diffusion–reaction equation

(III) Convection–diffusion–reaction equation on closed surface Γ:

$$J_s(\boldsymbol{x}; W, b) = J_{f,\Gamma}(\boldsymbol{x}; W, b). \tag{3.7}$$

(IV) Positivity-preserving convection–diffusion–reaction equation on closed surface Γ:

$$J_{s,p}(\boldsymbol{x}; W, b) = \lambda_1 J_{f,\Gamma}(\boldsymbol{x}; W, b) + \lambda_2 J_{PPC}(\boldsymbol{x}; W, b). \tag{3.8}$$

Combined with the above two types of loss functions and method, we get DLM_{3d} and $DLM_{3d,p}$ for 3D problem, DLM_s and $DLM_{s,p}$ for surface problem. In different tasks, we have different requirements for the numerical solution of the equation. Therefore, we can adjust parameters λ_1, λ_2, and λ_3 flexibly. In subsequent numerical examples, we usually choose a tenfold relationship between λ_1, λ_2, and λ_3.

3.3. Optimization algorithm

In the world of deep learning, we usually find that there are many problems with no analytical solution, or it takes a lot of computation to calculate the optimal solution. Facing this kind of problem, the general method is to use iterative method to approach the optimal solution of the problem as much as possible. We call the method to solve this kind of optimization problem optimization algorithm, which is essentially a mathematical method. Common optimization algorithms include Gradient descent (GD), Momentum, Adagrad, Adaptive Moment Estimation (Adam) and so on.

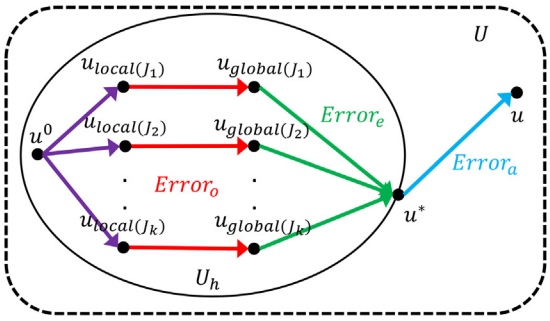

Fig. 3. Decomposition of DLM errors. $Error_a$, $Error_e$ and $Error_o$ are approximation error, estimation error and optimization error, respectively. $u_{local(j)}$ is local optimal solution obtained by the OA, $u_{global(j)}$ is global optimal solution under J_i, u^0 is the starting point for optimization, u^* is optimal approximation solution generated by DNNs, u is a solution to the CDRE.

Adam: Adam is a random gradient descent optimization method based on the idea of SGD and Momentum. It calculates the first-order and second-order moments of the gradient before iteration, and calculates the moving average value, which is used to update the current parameters. The default parameter value are $\delta = 1e{-}3$, $\rho_1 = 0.9$, $\rho_2 = 0.999$ and minimal constant $\varrho = 1e-8$.

$$\begin{aligned} g_t(\boldsymbol{x}) &= \tfrac{1}{n}\textstyle\sum_{i=1}^{n} \nabla_\theta J_i(\boldsymbol{x}_i;\theta),\\ m_t(m_{t-1}, g_t) &= \rho_1 \cdot m_{t-1} + (1-\rho_1)\cdot g_t,\\ v_t(v_{t-1}, g_t) &= \rho_2 \cdot v_{t-1} + (1-\rho_2) g_t^T g_t,\\ \hat{m}_t(m_t) &= \tfrac{m_t}{1-\rho_1},\\ \hat{v}_t(v_t) &= \tfrac{v_t}{1-\rho_2},\\ \theta_{t+1}(\theta_t, \hat{v}_t, \hat{m}_t) &= \theta_t - \tfrac{\delta}{\varrho+\sqrt{\hat{v}_t}}\cdot \hat{m}_t. \end{aligned} \tag{3.9}$$

After the bias correction of Adam's gradient, the learning rate of each iteration has a fixed range, which makes the parameters more stable. It calculates different adaptive learning rates for different parameters. It is also suitable for most nonconvex optimization problems (large data sets and high-dimensional space). In this paper, we choose Adam to optimize the loss function and solve CRDE.

3.4. Error source analysis

There is a Theorem shows that any function and its partial derivatives can be uniformly approximated by DNNs in [34]. However, this is not possible due to the limitation of computing resources. Therefore, we analyze the source of the error to provide the idea of reducing the error.

The error of DLM can be decomposed into three parts which are shown in Fig. 3: approximation error: $Error_a$, estimation error: $Error_e$ and optimization error: $Error_o$. After d_{in}, N_{hidden}, N_{neuron}, d_{out} and σ_{DNN} are selected, DNNs can constitute the approximation finite dimensional space U_h. Assume that the optimal approximation solution in U_h is u^*. Therefore, $Error_a$ is generated by u^* in U_h approaching u in an infinite dimensional space U. When we set up different loss functions, we get different global optimal solution $u_{global(J_i)}$. So there is a $Error_e$ which is caused by $u_{global(J_i)}$ and u^*. At present, in the field of deep learning, the loss functions established are highly non-convex. Therefore, it is difficult for us to get the global optimal $u_{global(J_i)}$. Even if we start with the same u^0, we always get a different $u_{local(J_i)}$. Obviously, the third part of the $Error_o$ is created.

4. Numerical examples

In this section, we present five numerical experiments to verify the accuracy and effectiveness of the deep learning method (DLM) for solving steady CDRE. Firstly, we use DLM to solve problem with exact solution and compare the results with finite element method (FEM). In addition, the performance of the proposed method in solving anisotropic region problems is explored. Furthermore, under the different coefficients ϵ, DLM is used to deal with convection-dominated problems. And then, we also study the numerical results of the algorithm in the non-convex region (L-type). In the end, DLM is extended to surface and compared the numerical results with the traditional surface FEM. We relied on the deepxde library for our experiments and uploaded the code to GitHub.[1]

Table 4.1
Deep neural network parameters for Example 4.1.

$N_{hidden} * N_{neuron}$	σ	OA	η	N_{epoch}
$4 * 50$	σ_T	Adam	$5e-4$	50000

Table 4.2
Compare the error of FEM and DLM_{3d} on prediction points: $\epsilon = 1e-1$.

N_{FEM}	E_{FEM}	$N_{DLM_{3d}}$	$E_{DLM_{3d}}$
125	1.7727e2	150	3.9893e1
1000	5.8275e3	1000	3.5878e2
3375	4.3728e3	3400	1.3899e2
8000	7.3974e3	8000	5.6793e3
15625	9.7051e3	15600	1.9720e3
27000	1.5744e2	27000	1.6103e3

Table 4.3
Compare the error of FEM and DLM_{3d} on prediction points: $\epsilon = 1e-3$.

N_{FEM}	E_{FEM}	$N_{DLM_{3d}}$	$E_{DLM_{3d}}$
125	2.0440e1	150	3.3386e0
1000	1.3710e1	1000	4.7183e2
3375	7.9074e2	3400	6.4395e3
8000	4.6726e2	8000	3.8301e3
15625	3.2712e2	15600	2.6859e3
27000	2.0759e2	27000	1.9880e3

In the following example, we use the following relative error:

$$E = \sum_{i=1}^{K} \frac{(u_{Exact}(\boldsymbol{x}_i) - u_{DLM}(\boldsymbol{x}_i))^2}{u_{Exact}(\boldsymbol{x}_i)^2}. \tag{4.1}$$

4.1. A numerical example with the exact solution

We firstly consider the test the accuracy of the proposed DLM_{3d} for (1.1). Consider exact solution:

$$u(x, y, z) = \frac{xy}{\pi}\arctan(\frac{z}{\sqrt{\epsilon}}), \tag{4.2}$$

on the unit cube domain ($\Omega = [0,1]^3$). In this test, the velocity field of diffusion and convection coefficient is chosen to be $\epsilon = 1e-1, 1e-3$, $\beta = (0, 0, 0.5)$ and $\gamma = 1$. And set $\lambda_1 = 1$, $\lambda_2 = 10$, $K = 8000$. Other deep neural network parameters were selected as shown in Table 4.1, and there are 7901 parameters to be trained.

In Table 4.2 and Table 4.3, we respectively compare the errors of FEM and DLM_{3d} predictions at the same K points selected when $\epsilon = 1e{-}1, 1e{-}3$. We can find that DLM_{3d} performs better. And DLM_{3d} can be an order of magnitude lower than FEM when using the same data information.

In Fig. 4 and Fig. 5, the calculation results and the absolute error $|u_{true} - u_{DLM}|$ and $|u_{true} - u_{FEM}|$ are given respectively. The true solution and the change curve of the loss function in the training process on the training set and the test set are also given.

Next, we analyzed size of the data set $\mathcal{D}$, and selected $\epsilon = 1e-3$, $N_{hidden} = 5$, $N_{neuron} = 50$, $\eta = 1e-3$ and $N_{epoch} = 20000$. The influence of results with different $|\mathcal{D}_f|$ and $|\mathcal{D}_g|$ are compared. From Table 4.4,

[1] https://github.com/Poker-Pan/PINN-CDREs

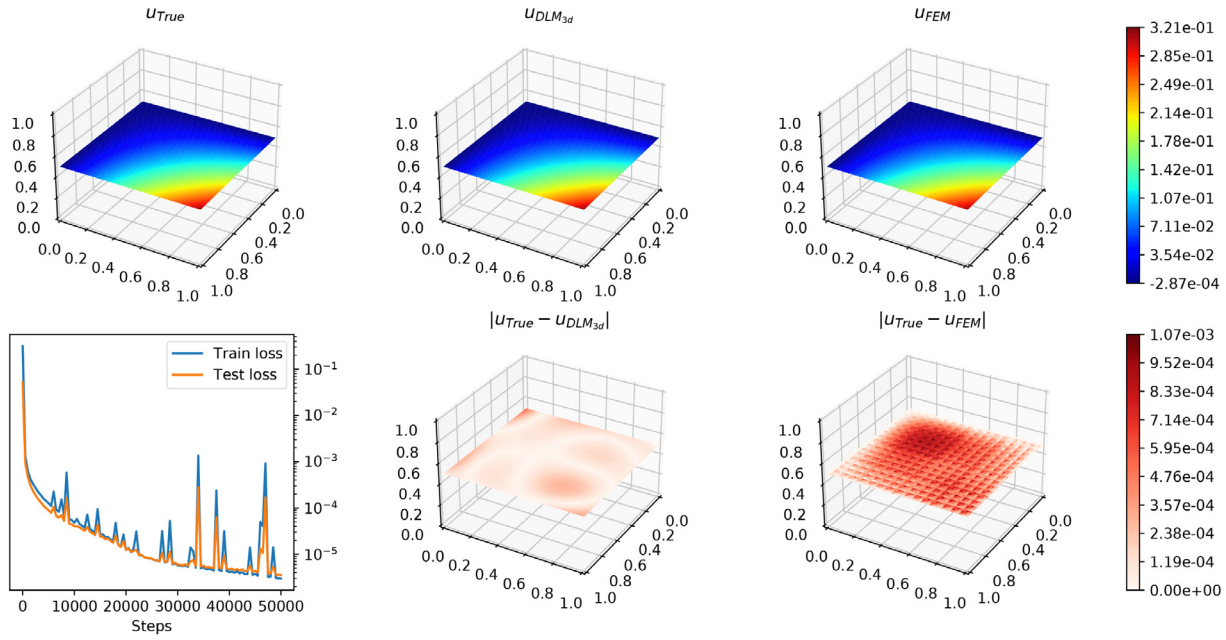

Fig. 4. Comparison of the DLM_{3d} solution with the FEM solution: $\epsilon = 1e-1$. (Top left) the true solution: u_{true}, (Top middle) the DLM_{3d} solution: u_{DLM}, (Top right) the FEM solution: u_{FEM}, (Bottom left) training process of DLM, (Bottom middle) the absolute error: $|u_{true} - u_{DLM}|$, (Bottom right) the absolute error: $|u_{true} - u_{FEM}|$.

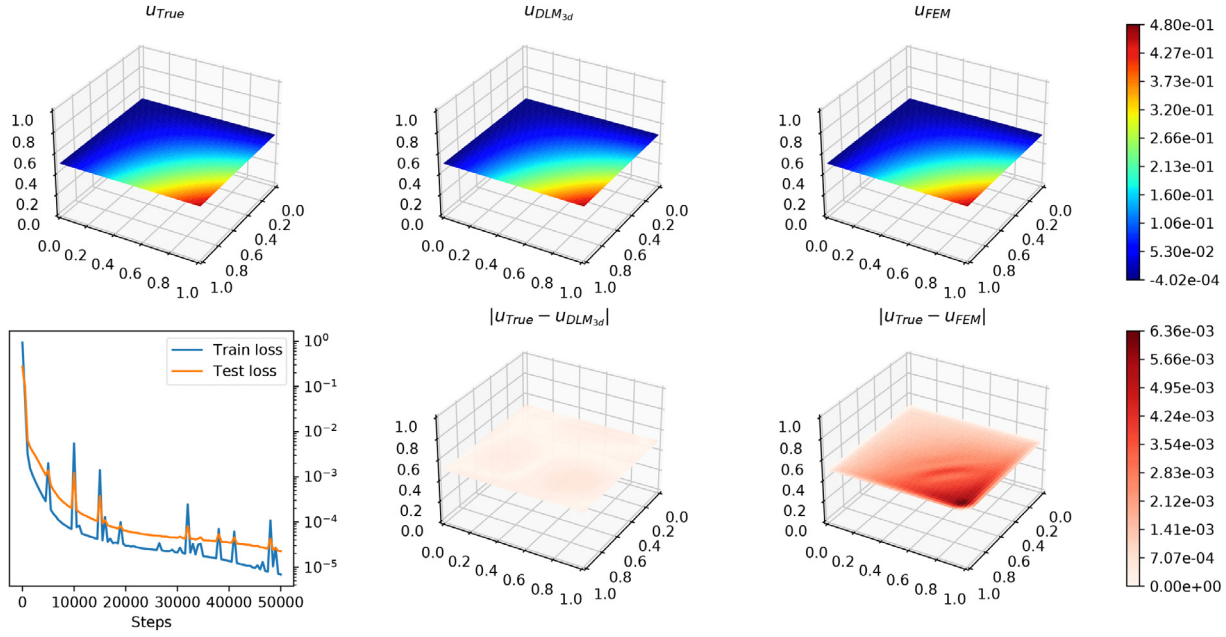

Fig. 5. Comparison of the DLM_{3d} solution with the FEM solution: $\epsilon = 1e-3$. (Top left) the true solution: u_{true}, (Top middle) the DLM_{3d} solution: u_{DLM}, (Top right) the FEM solution: u_{FEM}, (Bottom left) training process of DLM, (Bottom middle) the absolute error: $|u_{true} - u_{DLM}|$, (Bottom right) the absolute error: $|u_{true} - u_{FEM}|$.

we can find that the relative error of prediction points decreases with the increase of $|D_f|$ and $|D_g|$. And, we also analyzed the relationship between N_{hidden} and N_{neuron}. It is not difficult to find in Table 4.5 that there is a similar relationship between them. With the increase of N_{hidden} and N_{neuron}, the relative error of prediction points decreases.

In addition, we propose to adopt different optimization algorithms in algorithm . Next, we compare the error and calculation time of SGD, Moment and Adam under the same data and network configuration.

It is not difficult to find from Table 4.6 that the calculation time difference of the three optimization algorithms is not large, but Adam has higher accuracy compared with the other two algorithms. Therefore, we adopted this better optimization algorithm in the following calculation examples.

4.2. Anisotropic region

In this part, we consider anisotropic region for convection-domina ted diffusion problem (1.1). We research the problem in $\Omega = [0,1] \times [0,2] \times [0,0.1]$, and choose $\epsilon = 1e-6$, $\gamma = 1$, and boundary condition as follows:

$$g_n(x,y,z) = 0, \quad (x,y,z) \in [0,1] \times \{2\} \times [0,0.1],$$
$$g_d(x,y,z) = \begin{cases} z(0.1-z), & (x,y,z) \in [0,1] \times \{0\} \times [0,0.1], \\ 0, & \text{otherwise.} \end{cases} \tag{4.3}$$

Choose the proposed DLM_{3d}, and set $\lambda_1 = 1$, $\lambda_2 = 10$. Other deep neural network parameters were selected as shown in Table 4.7, and there are 7851 parameters to be trained.

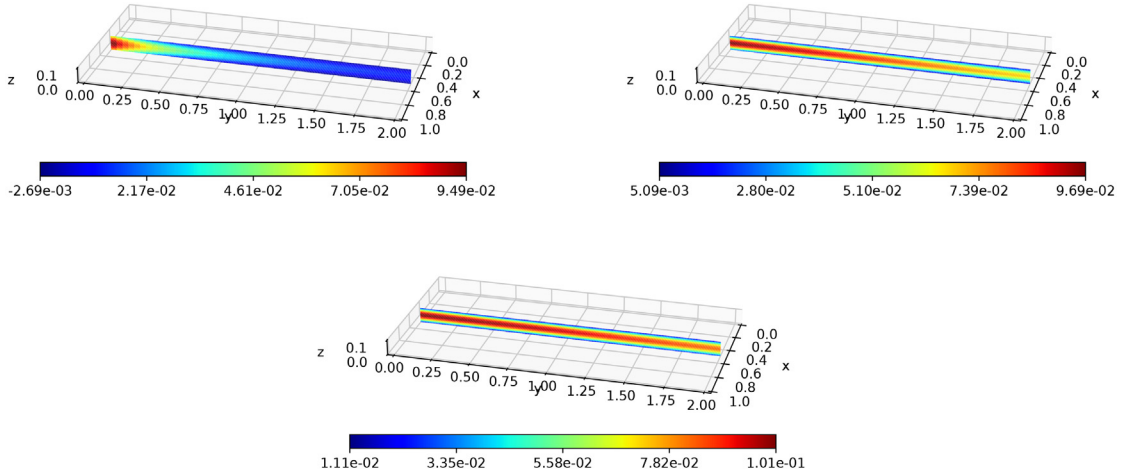

Fig. 6. Numerical solutions for Anisotropic region by DLM_{3d}: (Top Left) $\beta = (0, 1, 0)$, (Top Right) $\beta = (0, 5, 0)$, (Bottom Middle) $\beta = (0, 10, 0)$.

Table 4.4
The selection of boundary points and comparison points.

D_g \ D_f	2000	4000	6000	8000	10000
20	$1.1597e-2$	$2.3391e-2$	$1.7169e-2$	$2.1063e-2$	$2.1435e-2$
40	$8.9781e-3$	$3.5047e-2$	$6.0156e-3$	$7.4503e-3$	$6.0518e-3$
60	$1.1280e-2$	$3.9668e-2$	$7.3418e-3$	$7.3990e-3$	$2.2805e-3$
80	$7.0926e-3$	$5.0861e-3$	$1.9062e-2$	$9.0909e-3$	$1.3420e-3$
100	$8.8055e-3$	$3.4333e-3$	$1.5803e-2$	$4.8301e-3$	$2.8189e-3$

Table 4.5
The relationship between the number of comparative layers and the number of neurons.

N_{hidden} \ N_{neuron}	20	40	60	80	100
2	$1.0536e-2$	$1.0262e-2$	$1.1451e-2$	$9.3551e-3$	$7.1317e-3$
4	$1.0437e-2$	$1.2791e-2$	$7.7580e-3$	$5.0831e-3$	$5.1284e-3$
6	$6.1561e-3$	$8.1737e-3$	$8.4630e-3$	$4.3426e-3$	$1.2759e-4$
8	$1.0631e-2$	$1.2372e-2$	$3.4112e-3$	$7.3728e-3$	$1.3747e-4$
10	$9.6658e-3$	$2.3576e-2$	$2.5003e-2$	$5.3468e-4$	$1.1991e-4$

Table 4.6
Comparison of three different optimization algorithms.

	$E_{DLM_{3d}}$	$t(s)$
SGD	$1.2662e1$	$1.6329e3$
Momentum	$9.9630e2$	$1.3480e3$
Adam	$5.6793e3$	$1.1580e3$

Table 4.7
Deep neural network parameters for Example 4.2.

$N_{hidden} * N_{neuron}$	$\mathcal{N}_{train}$	σ	OA	η	N_{epoch}
$4 * 50$	1320	σ_T	Adam	$1e-3$	10000

From Fig. 6, we can observe that as convection coefficient β increases, the fluid moves farther to the right. The velocity of the fluid is greatest at the center of the region due to the greater coefficient of friction near the upper and lower boundaries. These can show that numerical results confirm the validness of our theoretical analysis and demonstrate the effectiveness of the given method.

4.3. Convection-dominated diffusion problem

In this section, we mainly consider convection-dominated diffusion problem (1.1) which has no exact solution. We research the problem in $\Omega = [0, 1]^3$, and choose $\beta = (1-z, 1-z, 1)$, $\gamma = 1$ and boundary condition as follows:

$$g_d(x, y, z) = \begin{cases} 1, & \text{if } 0 \le x, y \le 0.5, z = 0, \\ 0, & \text{otherwise.} \end{cases} \tag{4.4}$$

Table 4.8
Deep neural network parameters for Example 4.3.

$N_{hidden} * N_{neuron}$	$\mathcal{N}_{train}$	σ	OA	η	N_{epoch}
$5 * 45$	7600	σ_T	Adam	$7e-6$	50000

In addition, we consider strong convection-dominated problem with the small diffusion coefficient $\epsilon = 1e{-}6, 1e{-}9, 1e{-}12$, respectively. Since we want the solution u_{DLM} to be positive, we use the proposed $DLM_{3d,p}$ with $c = 0$, $d = +\infty$, and choose $\lambda_1 = 1$, $\lambda_2 = 1$, $\lambda_3 = 10$. The parameters related to DNN are selected as follows in Table 4.8, and there are 8506 parameters to be trained.

The numerical solutions along its main diagonal are presented in Fig. 7. We can observe that when the fluid enters the cube, due to the $\beta = (1 - z, 1 - z, 1)$, the fluid deflects and flows out of the top. Compared with [35,36], we can find that our proposed $DLM_{3d,p}$ can better capture the phenomenon with $\epsilon = 1e-6, 1e-9, 1e-12$ close to the Z axis.

4.4. L-shape region

In this section, we consider the performance of the $DLM_{3d,p}$ with $c = 0$, $d = +\infty$ on non-convex regions $\Omega = [0, 1]\times[0, 2]\times[0, 0.5] \setminus [0, 1]\times[0, 0.5]\times[0, 0.3]$, and choose $\beta = (0, 3, -y)$, $\gamma = 1$ and boundary condition as follows:

$$g_n(x, y, z) = 0, \quad (x, y, z) \in [0, 1]\times\{2\}\times[0, 0.5],$$
$$g_d(x, y, z) = \begin{cases} \kappa(0.5 - z)(z - 0.3), & (x, y, z) \in [0, 1]\times\{0\}\times[0.3, 0.5], \\ 0, & \text{otherwise.} \end{cases}$$

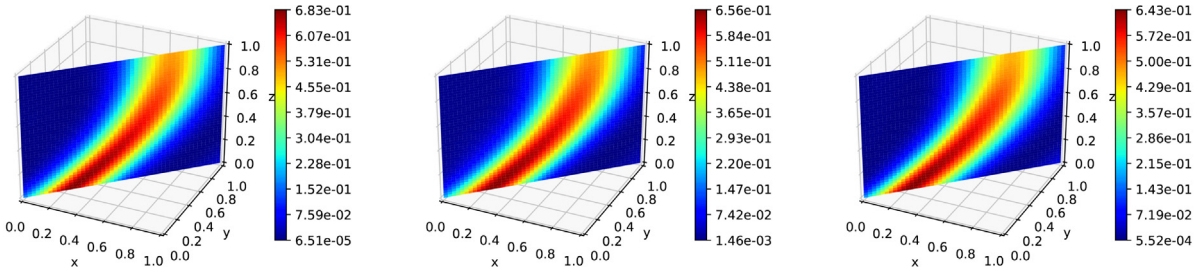

Fig. 7. Numerical solutions for strong convection-dominated problem by $DLM_{3d,p}$. (Left) $\epsilon = 1e-6$, (Middle) $\epsilon = 1e-9$, (Right) $\epsilon = 1e-12$.

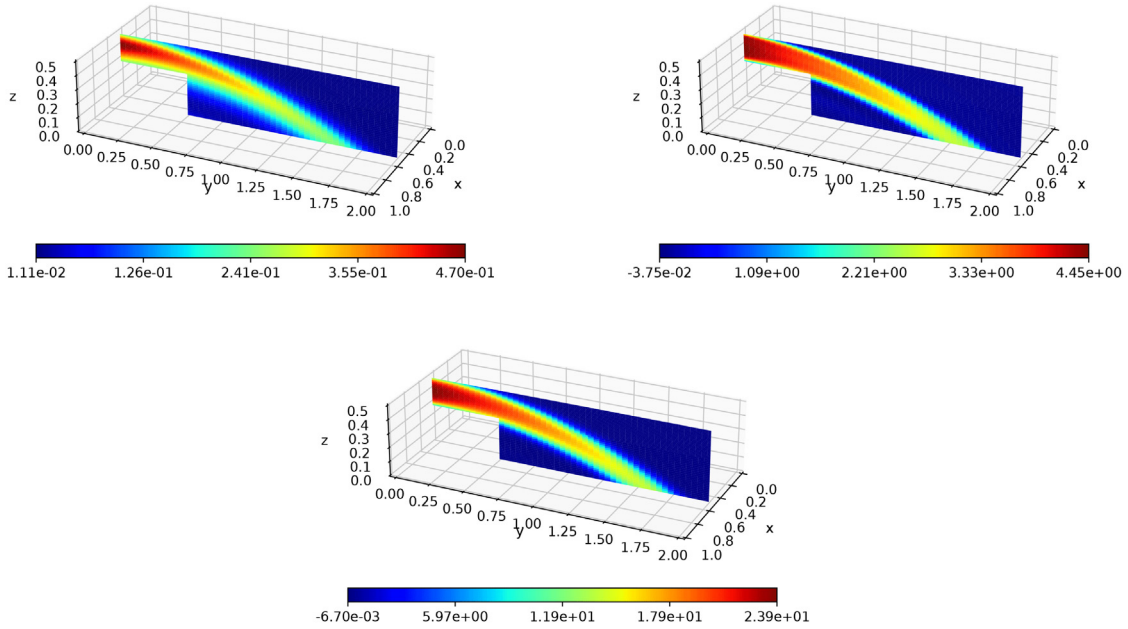

Fig. 8. Numerical solutions for problem in L-shape region by $DLM_{3d,p}$: (Top Left) $\kappa = 1$, (Top Right) $\kappa = 10$, (Bottom Middle) $\kappa = 50$.

Table 4.9
Deep neural network parameters for Example 4.4.

$N_{hidden} * N_{neuron}$	$\mathcal{N}_{train}$	σ	OA	η	N_{epoch}
$5 * 50$	6600	σ_T	Adam	$1e-4$	50000

Table 4.10
Deep neural network parameters for Example 4.4.

$N_{hidden} * N_{neuron}$	$\mathcal{N}_{train}$	σ	OA	η	N_{epoch}
$6 * 50$	12900	σ_T	Adam	$5e-4$	20000

(4.5)

Choose loss function: $J_{3d,p}$, and set $\lambda_1 = 1$, $\lambda_2 = 10$, $\lambda_3 = 50$. Other deep neural network parameters were selected as shown in Table 4.9, and there are 7851 parameters to be trained.

By comparing the results Fig. 8 with [37], fluid motion is almost the same. When κ increases, the fluid enters faster and travels farther. And because of gravity, the direction of fluid movement deflects downward, which is consistent with the actual situation.

4.5. Complex 3D region

In this section, we consider the capabilities of DLM_{3d} algorithm in complex three-dimensional regions. We study the following region:

$$\Omega : \phi(x, y, z) = \tanh(\sqrt{P(x, y, z) + 51} - \sqrt{12}) \leq 0, \tag{4.6}$$

where

$$P(x, y, z) = 75x^4+75y^4+75z^4+50x^2y^2+50x^2z^2+50y^2z^2-90x^2-90y^2-90z^2. \tag{4.7}$$

Then choose $\epsilon = 1$, $\beta = (1, 1, 1)$, $\gamma = 1$. Moreover, we consider the exact solution on all surfaces:

$$u(x, y, z) = x^2y^2z^2. \tag{4.8}$$

Choose loss function: J_{3d}, and set $\lambda_1 = 1$, $\lambda_2 = 10$. Other deep neural network parameters were selected as shown in Table 4.10, and there are 13001 parameters to be trained. Through the analysis of Fig. 9, it can be found that the proposed method DLM_{3d} is also well applicable to complex 3D regions.

4.6. Surface PDEs

In this part, we mainly consider extending the domain to the surface. We consider (1.1) on a sphere:

$$\Gamma : x^4 + y^4 + z^4 - x^2 - y^2 - z^2 = 0 \tag{4.9}$$

with the exact solution:

$$u(x, y, z) = \arctan(\frac{x}{\epsilon})\arctan(\frac{y}{\epsilon})\arctan(\frac{z}{\epsilon}). \tag{4.10}$$

The convection–diffusion equation has a smooth exact solution when ϵ is large. When a small ϵ is employed, it is a convection-dominated

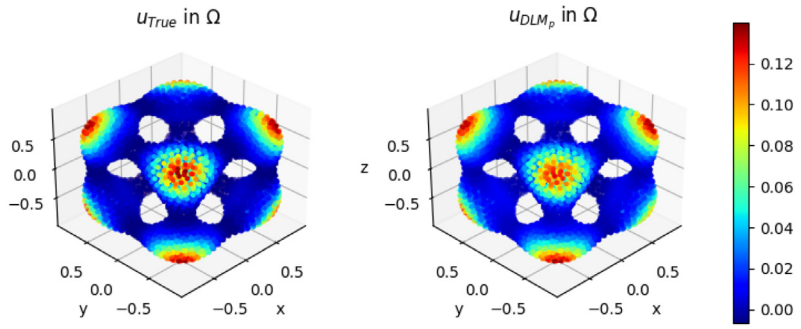

Fig. 9. Numerical solutions for problem in complex 3D region by DLM_{3d}.

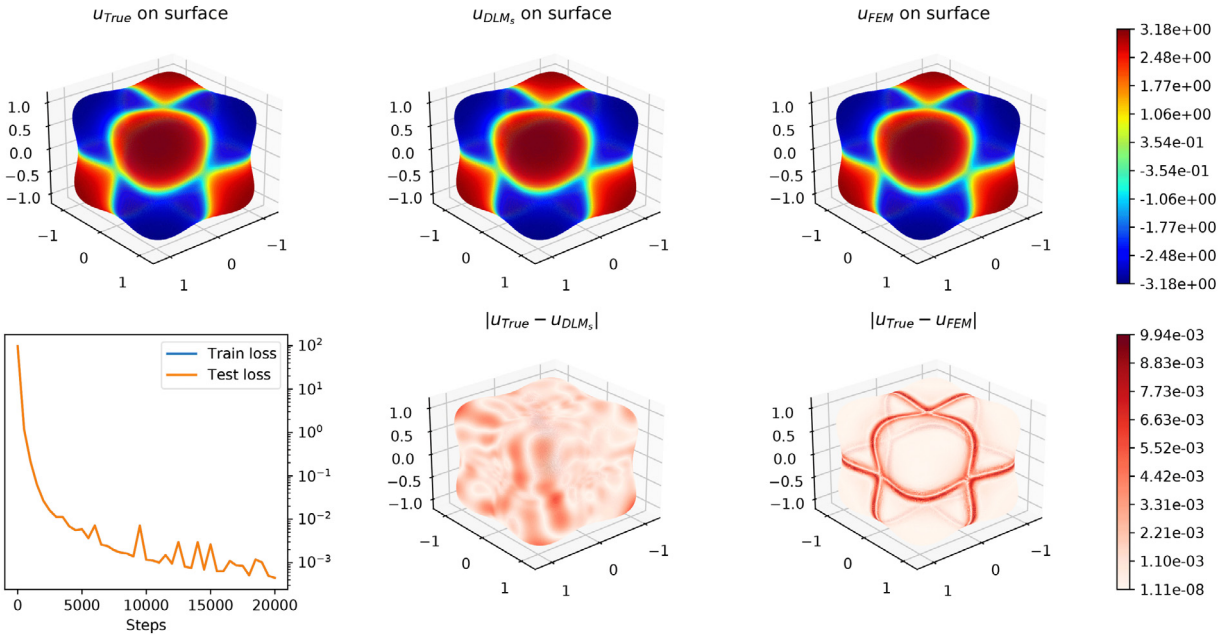

Fig. 10. Comparison of the DLM_s solution with the FEM solution: $\epsilon = 1e-1$. (Top left) the true solution: u_{true}, (Top middle) the DLM_s solution: u_{DLM}, (Top right) the FEM solution: u_{FEM}, (Bottom left) training process of DLM, (Bottom middle) the absolute error: $|u_{true} - u_{DLM}|$, (Bottom right) the absolute error: $|u_{true} - u_{FEM}|$.

Table 4.11
Deep neural network parameters for Example 4.5.

$N_{hidden} * N_{neuron}$	σ	OA	η	N_{epoch}
$5 * 50$	σ_T	Adam	$1e-4$	20000

Table 4.12
Compare the error of FEM and DLM_s on prediction points: $\epsilon = 1e-1$.

N_{FEM}	E_{FEM}	$t(s)$	N_{DLM_s}	E_{DLM_s}	$t(s)$
698	7.0620e2	1.2039e2	700	1.3397e2	8.2380e2
3482	1.3604e2	1.1283e3	3500	2.0893e3	1.3720e3
14042	3.1677e3	2.5400e3	14000	8.6542e4	1.7326e3
55658	7.7885e4	5.9289e3	55650	1.3548e4	4.0472e3

diffusion problem with an exact solution which has large variations of function values and gradients. We set $\epsilon = 1e{-}1, \beta = (1, 1, 1), \gamma = 1$. Other deep neural network parameters were selected as shown in Table 4.11, and there are 10 451 parameters to be trained.

It can be found in the Table 4.12 that when using the same data points, our DLM_s algorithm has higher accuracy than the traditional surface finite element method. And it saves more time than traditional methods. These times include generating data points and mesh.

In Fig. 10, we present the calculation results when 55 650 points are used. Through the analysis of the experimental results, the algorithm can effectively reduce the numerical oscillation. And due to the increase of data points, there is little difference between the test set and the training set, so in the bottom left of Fig. 10, the loss function values of the training set and the test set are almost the same.

Moreover, we give the data points and mesh used by DLM_s and FEM respectively. Our algorithm DLM_s only needs random collection of data points, as shown in Fig. 11(a), which only takes a little time. However, generating the mesh required for FEM, as shown in Fig. 11(b), requires a high time cost.

In addition, we compare the proposed method with deep learning methods presented in literatures [38,39] on solving partial differential equations. Literature [38] uses the test functions in the weak solution and the weak equation as parameters for the original network and the adversarial network (WAN), respectively, and updates them alternately to approximate the optimal network parameter settings. Literature [39] uses a combination of Bayesian neural networks and PINNs (B-PINNs) as a prior, while Hamiltonian Monte Carlo or variational inference can be used as a posterior estimate. The B-PINNs utilize both laws of physics and scattering noise measurements to provide predictions and to quantify the uncertainty arising from the noisy data within a Bayesian framework.

From Table 4.13, we can find that the B-PINNs, combined with Bayesian networks, takes the longest time to compute and has relatively good accuracy. The WAN gives the smallest error in the results, but takes longer than the ours PINNs. Our PINNs method, while does not

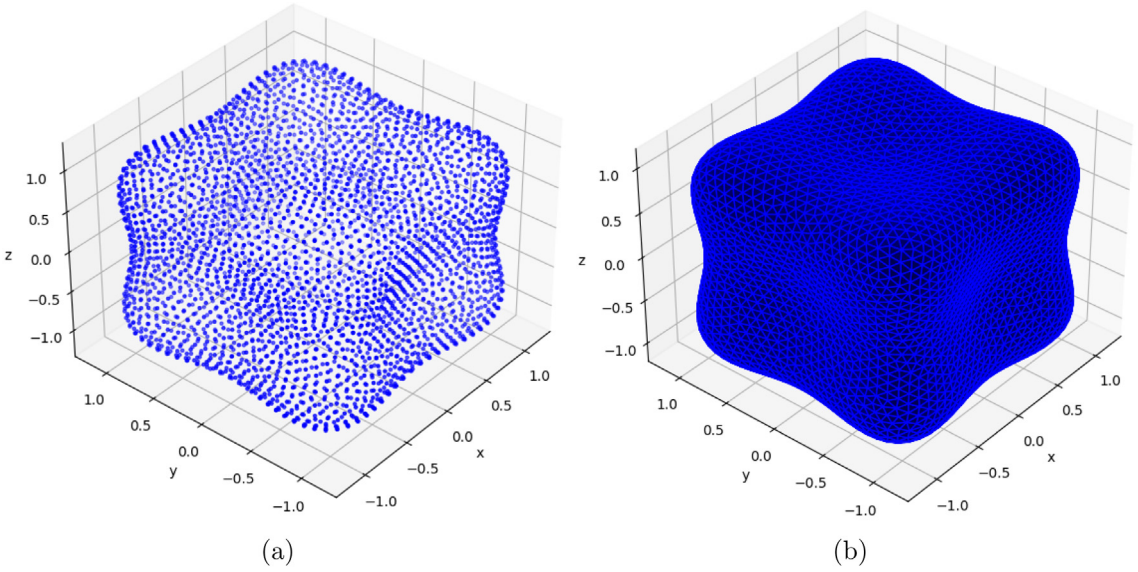

Fig. 11. Select data points and construct mesh on the sphere.

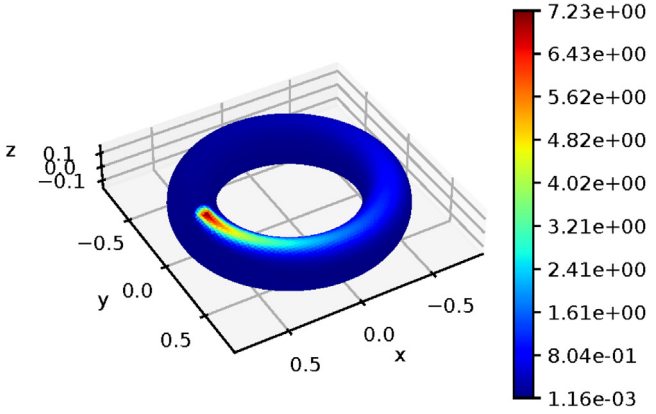

Fig. 12. Numerical results for the discontinuous source problem with $\alpha = 1e-3$.

Table 4.13
Comparison of three different optimization algorithms.

	N_{DLM_s}	E_{DLM_s}	$t(s)$
WAN	55650	5.5631e5	5.5972e3
B-PINNs	55650	8.8493e5	7.3081e3
PINNs(ours)	55650	1.3548e4	4.0472e3

have the best accuracy, is comparable to the other two in terms of error and has the least computational time. This also shows that our approach has room for improvement.

4.7. Discontinuous source experiment

In this section, we solve the problem of discontinuous sources with convection-occupied diffusion on a torus,

$$\Gamma_1 : (0.6 - \sqrt{x^2 + y^2})^2 + z^2 - 0.15^2 = 0. \tag{4.11}$$

We choose $\alpha = 1e-3$, $\boldsymbol{\beta} = (-y, x, 0)$, $\gamma = 0.5$. The term on the right side of the equation is chosen as follows:

$$f(x, y, z) = \begin{cases} 50, & (x-0.6)^2 + y^2 + (z-0.15)^2 < 0.05^2, \\ 0, & \text{otherwise.} \end{cases} \tag{4.12}$$

Table 4.14
Deep neural network parameters for Example 4.6.

$N_{hidden} * N_{neuron}$	σ	OA	η	N_{epoch}
$6 * 50$	σ_T	Adam	$7e-5$	25000

Table 4.15
Error results on each surface for Example 4.7.

	Γ_2	Γ_3	Γ_4	Γ_5
E	2.0378e3	1.5327e2	8.3826e3	3.2367e2

Choose loss function: $J_{s,p}$, and set $\lambda_1 = 1$, $\lambda_2 = 10$. Other deep neural network parameters were selected as shown in Table 4.14, and there are 13 001 parameters to be trained.

Since the exact solution of the equation is not known, we can only compare it with the numerical results in the same case. Therefore, we compare the numerical results with [40], and we can find that our numerical results (see Fig. 12) are consistent. This shows the success of the simulation and the reliability of the method.

4.8. Surface with different topologies

In the last section, in order to verify that the proposed method can be widely applied to all kinds of surfaces. We apply the proposed method to the following classes of surfaces, all of which are topologically different. Spherical (Γ_2), torus (Γ_3), the surface of bottle opener (Γ_4), the wheel surface (Γ_5):

$$\begin{aligned}
&\Gamma_2 : x^2 + y^2 + z^2 - 1 = 0. \\
&\Gamma_3 : (0.6 - \sqrt{x^2 + y^2})^2 + z^2 - 0.15^2 = 0. \\
&\Gamma_4 : (x^2(1-x^2) - y^2)^2 + \frac{1}{2}z^2 - \frac{1}{40} = 0. \\
&\Gamma_5 : ((x^2 + y^2 - 1)^2 + z^2)((x^2 + z^2 - 1)^2 + y^2)((y^2 + z^2 - 1)^2 + x^2) \\
&\quad - (0.075)^2(1 + 3(x^2 + y^2 + z^2)) = 0.
\end{aligned} \tag{4.13}$$

During the test, the equation of the parameter selection of $\alpha = 1$, $\boldsymbol{\beta} = (1, 1, 1)$, $\gamma = 1$, $\lambda_1 = 1$, $\lambda_2 = 10$. Moreover, we consider the exact solution on all surfaces:

$$u(x, y, z) = x^2 y^2 z^2. \tag{4.14}$$

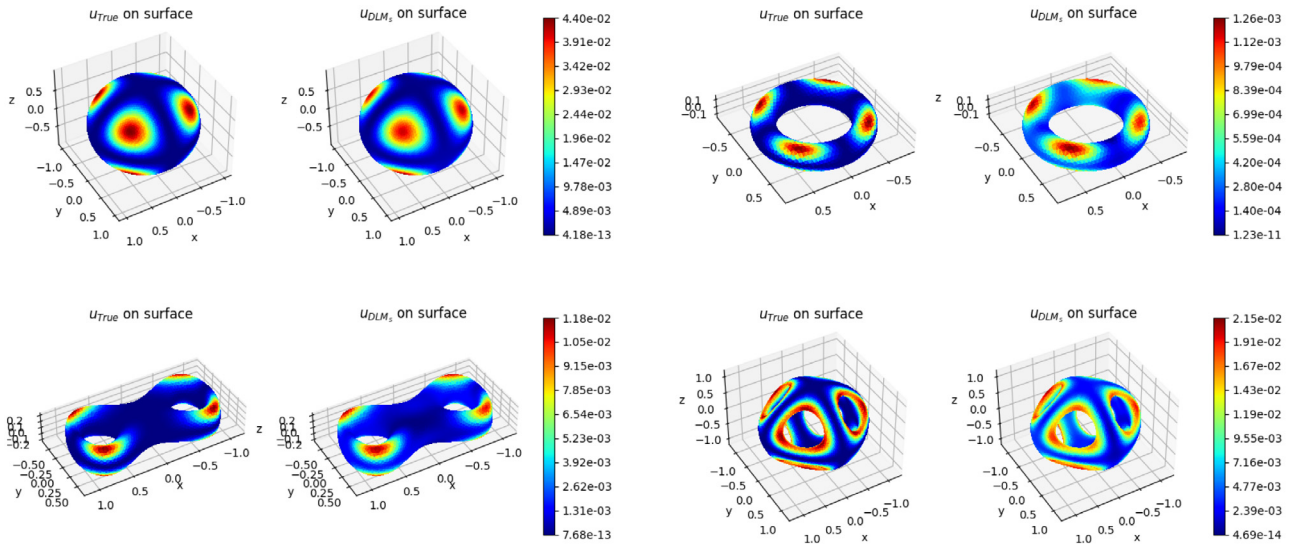

Fig. 13. Numerical results of the algorithm on various types of surfaces.

Through the analysis of Table 4.15, it can be seen that the proposed method is suitable for different surfaces. All the above surfaces belong to different topological structures, which also reflects the effectiveness of the algorithm in this paper. We give numerical results on various surfaces in Fig. 13. Unfortunately, the algorithm oscillates on Γ_3.

5. Conclusions

We propose a novel data-driven deep learning algorithm for solving high-dimensional convection–diffusion–reaction equations. In terms of method, this method avoids the high cost of mesh generation, and only needs to collect data points randomly according to the given distribution. In addition, in order to satisfy the physical properties such as positivity and maximum principle, We design two types of loss functions and obtain four different algorithms. Numerical examples verify the accuracy and effectiveness of our algorithm, which show that DLM_{3d} and DLM_s has better accuracy than the traditional FEM in high-dimensional domain. In addition, $DLM_{3d,p}$ performs well in anisotropic and non-convex regions. Besides, this novel algorithm can be extended to other manifolds and fluid dynamics models.

On the theoretical side, while the current work does not include a complete theoretical analysis, in our future work, we plan to leverage the neural tangent kernel technique (NTK) to aid us in proving the convergence [41–43], error estimation, and other key properties of our algorithm. This theoretical investigation will provide deeper insights into the behavior and guarantees of our method, enhancing its reliability and applicability. We have conducted extensive numerical experiments both in three dimensions and on surfaces to demonstrate the effectiveness and advantages of our proposed approach. The numerical experiments reveal that our method consistently produces highly accurate numerical solutions. Especially for problems with complex computational regions, it outperform standard finite element methods in both 3D space and on surfaces. Moreover, in numerical experiments, by comparing with some improved methods based on PINNs, we find that our method is inferior to the improved PINNs method. This finding is enlightening for our subsequent work. We can modify the network structure and loss function to meet different needs.

In managerial aspects, we recognize that there are several avenues for future research and improvement. First, extending the scope of our method to handle more diverse and challenging PDE problems will be pursued. Second, we aim to investigate the potential of adaptively refining the neural network architecture to improve the efficiency and accuracy of our approach. Third, exploring the integration of domain knowledge into the PINN framework could lead to further enhancements in specific application domains. In conclusion, this study contributes to bridging the gap between traditional finite element methods and the emerging field of PINN-based approaches for solving PDEs.

Funding

This work is in parts supported by the Natural Science Foundation of Xinjiang Province (No. 2023D01C164, No. 2022TSYCTD0019 and No. 2022D01D32), the National Natural Science Foundation of China (No. 12361090 and No. 12001466), and the Graduate Student Research Innovation Program of Xinjiang (No. XJ2020G024).

CRediT authorship contribution statement

Jiangong Pan: Investigation, Software, Validation, Writing – original draft. **Xufeng Xiao:** Conceptualization, Methodology, Formal analysis, Software, Writing – review & editing. **Lei Guo:** Formal analysis, Writing – review & editing. **Xinlong Feng:** Formal analysis, Writing – review & editing.

Declaration of competing interest

No conflict of interest exists in the submission of this manuscript, and manuscript is approved by all authors for publication. I would like to declare on behalf of my co-authors that the work described was original research that has not been published previously, and not under consideration for publication elsewhere, in whole or in part.

Data availability

Data will be made available on request.

Acknowledgments

The authors would like to thank the editors and referees for their valuable comments and suggestions which helped us to improve the results of this paper.

Appendix

We will add the specific experimental procedures and relevant experimental codes in the appendix. You can find the full code at https://github.com/Poker-Pan/PINN-CDREs. The main implementation process is divided into the following steps and our experiment is based on Deepxde library:

1 Acquisition data point

In the first step, a specified number of data points are collected as input to the neural network in a planned calculation area (internal and boundary) according to a uniform or normal distribution. The relevant code is as follows:

```
    def train_points(self):
        X = super().train_points()
        if self.num_initial > 0:
            if self.train_distribution == "uniform":
                tmp = self.geom.uniform_initial_points(self.num_initial)
            else:
                tmp = self.geom.random_initial_points(
                    self.num_initial, random=self.train_distribution
                )
            if self.exclusions is not None:

                def is_not_excluded(x):
                    return not np.any([np.allclose(x, y) for y in self.exclusions])

                tmp = np.array(list(filter(is_not_excluded, tmp)))
            X = np.vstack((tmp, X))
        self.train_x_all = X
        return X
```

2 Build the PDE function

The second step is to build the correlation function based on the PDE to be computed. The return value of the function is the residual value of the equation.

```
    def pde(x, y):
        X, Y, Z = x[:, 0:1], x[:, 1:2], x[:, 2:]
        dy_x = tf.gradients(y, x)[0]
        dy_x, dy_y, dy_z = dy_x[:, 0:1], dy_x[:, 1:2], dy_x[:, 2:]
        dy_xx = tf.gradients(dy_x, x)[0][:, 0:1]
        dy_yy = tf.gradients(dy_y, x)[0][:, 1:2]
        dy_zz = tf.gradients(dy_z, x)[0][:, 2:]
        beta_1, beta_2, beta_3 = 0.0, 0.0, 0.5
        epsilon = 1e-1
        alpha = 2.5
        return -epsilon*(dy_xx + dy_yy + dy_zz) + (beta_1*dy_x + beta_2*dy_y + beta_3*dy_z) - \
            (2/(np.sqrt(epsilon)*np.pi)*X*Y*Z*tf.pow(1/(1+Z*Z/epsilon),2) + (beta_1*(Y/np.pi*tf.
                                                                              atan(Z/np.sqrt(epsilon))) + beta_2*(X
                                                                              /np.pi*tf.atan(Z/np.sqrt(epsilon))) +
                                                                               beta_3*(X*Y/(np.sqrt(epsilon)*np.pi*
                                                                              (1+Z*Z/epsilon)))))
```

3 Build boundary function

The third step is to create a function for the boundary conditions of PDE. Again, the return value of this function is the residual value of the boundary.

```
    def func(x):    # boundary condition
        X, Y, Z = x[:, 0:1], x[:, 1:2], x[:, 2:]
        epsilon = 1e-1
        value = X * Y / (np.pi) * np.arctan(Z / np.sqrt(epsilon))
        return value
```

4 Build neural network

The fourth step is to construct the neural network as the approximation function of the PDE solution according to the set parameter values.

```
class FNN(NN):
    """Fully-connected neural network."""
    def __init__(self, layer_sizes, activation, kernel_initializer):
```

```
        super().__init__()
        if isinstance(activation, list):
            if not (len(layer_sizes) - 1) == len(activation):
                raise ValueError(
                    "Total number of activation functions do not match with sum of hidden layers
                                                                        and output layer!"
                )
            self.activation = list(map(activations.get, activation))
        else:
            self.activation = activations.get(activation)
        initializer = initializers.get(kernel_initializer)
        initializer_zero = initializers.get("zeros")

        self.linears = torch.nn.ModuleList()
        for i in range(1, len(layer_sizes)):
            self.linears.append(
                torch.nn.Linear(
                    layer_sizes[i - 1], layer_sizes[i], dtype=config.real(torch)
                )
            )
            initializer(self.linears[-1].weight)
            initializer_zero(self.linears[-1].bias)

    def forward(self, inputs):
        x = inputs
        if self._input_transform is not None:
            x = self._input_transform(x)
        for j, linear in enumerate(self.linears[:-1]):
            x = (
                self.activation[j](linear(x))
                if isinstance(self.activation, list)
                else self.activation(linear(x))
            )
        x = self.linears[-1](x)
        if self._output_transform is not None:
            x = self._output_transform(inputs, x)
        return x
```

5 Training neural network

The optimization algorithms such as SGD, Momentum and Adam mentioned in the article are used to train the neural network and complete the whole calculation task.

```
model = dde.Model(data, net)
model.compile("adam", lr=eta)
losshistory, train_state = model.train(epochs=500)
```

References

[1] L. Yann, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.

[2] J. Berg, K. Nyström, A unified deep artificial neural network approach to partial differential equations in complex geometries, Neurocomputing 317 (2018) 28–41.

[3] M.A. Olshanskii, A. Reusken, X. Xu, A stabilized finite element method for advection-diffusion equations on surfaces, IMA J. Numer. Anal. 34 (2) (2014) 732–758.

[4] T. Fries, Higher-order surface FEM for incompressible Navier-Stokes flows on manifolds, Internat. J. Numer. Methods Fluids 88 (2) (2018) 55–78.

[5] I. Nitschke, A. Voigt, J. Wensch, A finite element approach to incompressible two-phase flow on manifolds, J. Fluid Mech. 708 (2012) 418.

[6] S. Reuther, A. Voigt, Solving the incompressible surface Navier-Stokes equation by surface finite elements, Phys. Fluids 30 (1) (2018).

[7] C.M. Elliott, B. Stinner, V. Styles, R. Welford, Numerical computation of advection and diffusion on evolving diffuse interfaces, IMA J. Numer. Anal. 31 (3) (2011) 786–812.

[8] P. Hansbo, M.G. Larson, S. Zahedi, Characteristic cut finite element methods for convection-diffusion problems on time dependent surfaces, Comput. Methods Appl. Mech. Engrg. 293 (2015) 431–461.

[9] X. Xiao, Z. Dai, X. Feng, A positivity preserving characteristic finite element method for solving the transport and convection-diffusion-reaction equations on general surfaces, Comput. Phys. Comm. 247 (2020).

[10] G. Dziuk, C.M. Elliott, Finite element methods for surface PDEs, Acta Numer. 22 (2013) 289.

[11] T. Lee, D. Mateescu, Experimental and numerical investigation of 2-D backward-facing step flow, J. Fluids Struct. 12 (6) (1998) 703–716.

[12] M. Dissanayake, N. Phan-Thien, Neural-network-based approximations for solving partial differential equations, Commun. Numer. Methods Eng. 10 (3) (1994) 195–201.

[13] I.E. Lagaris, A. Likas, D.I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (5) (1998) 987–1000.

[14] I.E. Lagaris, A.C. Likas, D.G. Papageorgiou, Neural-network methods for boundary value problems with irregular boundaries, IEEE Trans. Neural Netw. 11 (5) (2000) 1041–1049.

[15] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.

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

[17] G. Pang, L. Lu, G.E. Karniadakis, fPINNs: Fractional physics-informed neural networks, SIAM J. Sci. Comput. 41 (4) (2019) A2603–A2626.

[18] H.C. Elman, G.H. Golub, Iterative methods for cyclically reduced nonselfadjoint linear systems, Math. Comp. 54 (190) (1990) 671–700.

[19] H. Roos, H. Zarin, The streamline-diffusion method for a convection-diffusion problem with a point source, J. Comput. Appl. Math. 150 (1) (2003) 109–128.

[20] X. Guo, W. Li, F. Iorio, Convolutional neural networks for steady flow approximation, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 481–490.

[21] Y. Zhu, N. Zabaras, Bayesian deep convolutional encoder-decoder networks for surrogate modeling and uncertainty quantification, J. Comput. Phys. 366 (2018) 415–447.

[22] J. Adler, O. Öktem, Solving ill-posed inverse problems using iterative deep neural networks, Inverse Problems 33 (12) (2017).

[23] S. Bhatnagar, Y. Afshar, S. Pan, K. Duraisamy, S. Kaushik, Prediction of aerodynamic flow fields using convolutional neural networks, Comput. Mech. 64 (2) (2019) 525–545.

[24] Y. Chen, L. Lu, G.E. Karniadakis, L.D. Negro, Physics-informed neural networks for inverse problems in nano-optics and metamaterials, Opt. Express 28 (8) (2020) 11618–11633.

[25] H. Gao, J. Wang, M.J. Zahr, Non-intrusive model reduction of large-scale, nonlinear dynamical systems using deep learning, Physica D 412 (2020).

[26] G. Liu, X. Chen, Y. Hu, Anime sketch coloring with swish-gated residual U-net, in: International Symposium on Intelligence Computation and Applications, Springer, 2018, pp. 190–204.

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

[28] M.R. Chen, G.Q. Zeng, K.D. Lu, J. Weng, A two-layer nonlinear combination method for short-term wind speed prediction based on ELM, ENN and LSTM, IEEE Internet Things J. (2019) 1.

[29] H. Tu, Y. Xia, C.K. Tse, X. Chen, A hybrid cyber attack model for cyber-physical power systems, IEEE Access PP (99) (2020) 1.

[30] B. Cai, K. Hao, Z. Wang, C. Yang, X. Kong, Z. Liu, R. Ji, Y. Liu, Data-driven early fault diagnostic methodology of permanent magnet synchronous motor, Expert Syst. Appl. 177 (2021) 115000.

[31] B. Cai, Z. Wang, H. Zhu, Y. Liu, K. Hao, Z. Yang, Y. Ren, Q. Feng, Z. Liu, Artificial intelligence enhanced two-stage hybrid fault prognosis methodology of PMSM, IEEE Trans. Ind. Inform. 18 (10) (2021) 7262–7273.

[32] B. Cai, H. Fan, X. Shao, Y. Liu, G. Liu, Z. Liu, R. Ji, Remaining useful life re-prediction methodology based on Wiener process: Subsea Christmas tree system as a case study, Comput. Ind. Eng. 151 (2021) 106983.

[33] X. Kong, B. Cai, Y. Liu, H. Zhu, Y. Liu, H. Shao, C. Yang, H. Li, T. Mo, Optimal sensor placement methodology of hydraulic control system for fault diagnosis, Mech. Syst. Signal Process. 174 (2022) 109069.

[34] A. Pinkus, Approximation theory of the MLP model in neural networks, Acta Numer. 8 (1) (1999) 143–195.

[35] J. Bey, G. Wittum, Downwind numbering: Robust multigrid for convection-diffusion problems, Appl. Numer. Math. 23 (1) (1997) 177–192.

[36] G. Peng, Z. Gao, X. Feng, A novel cell-centered finite volume scheme with positivity-preserving property for the anisotropic diffusion problems on general polyhedral meshes, Appl. Math. Lett. 104 (2020).

[37] L. Qian, H. Cai, R. Guo, X. Feng, The characteristic variational multiscale method for convection-dominated convection-diffusion-reaction problems, Int. J. Heat Mass Transfer 72 (2014) 461–469.

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

[39] L. Yang, X. Meng, G.E. Karniadakis, B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data, J. Comput. Phys. 425 (2021) 109913.

[40] X. Xiao, X. Feng, Z. Li, A gradient recovery-based adaptive finite element method for convection-diffusion-reaction equations on surfaces, Internat. J. Numer. Methods Engrg. 120 (3) (2019).

[41] A. Jacot, F. Gabriel, C. Hongler, Neural tangent kernel: Convergence and generalization in neural networks, Adv. Neural Inf. Process. Syst. 31 (2018).

[42] J. Lee, J.Y. Choi, E.K. Ryu, A. No, Neural tangent kernel analysis of deep narrow neural networks, in: International Conference on Machine Learning, PMLR, 2022, pp. 12282–12351.

[43] M. Saadat, B. Gjorgiev, L. Das, G. Sansavini, Neural tangent kernel analysis of PINN for advection-diffusion equation, 2022, arXiv preprint arXiv:2211.11716.