Multi-Scale Neural Networks Based on Runge-Kutta Method for Solving Unsteady Partial Differential Equations*

CHEN Zebin, FENG Xinlong†

(*School of Mathematics and System Sciences, Xinjiang University, Urumqi Xinjiang 830017, China*)

Abstract: This paper proposes the multi-scale neural networks method based on Runge-Kutta to solve unsteady partial differential equations. The method uses q-order Runge-Kutta to construct the time iteratione scheme, and further establishes the total loss function of multiple time steps, which is to realize the parameter sharing of neural networks with multiple time steps, and to predict the function value at any moment in the time domain. Besides, the m-scaling factor is adopted to speed up the convergence of the loss function and improve the accuracy of the numerical solution. Finally, several numerical experiments are presented to demonstrate the effectiveness of the proposed method.

Key words: unsteady partial differential equations; q-order Runge-Kutta method; multi-scale neural networks; m-scaling factor; high accuracy

DOI: 10.13568/j.cnki.651094.651316.2022.06.25.0001

CLC number: O175 **Document Code**: A **Article ID**: 2096-7675(2023)02-0142-08

引文格式：陈泽斌, 冯新龙. Runge-Kutta 型多尺度神经网络求解非定常偏微分方程[J]. 新疆大学学报(自然科学版)(中英文), 2023, 40(2): 142-149.

英文引文格式：CHEN Zebin, FENG Xinlong. Multi-scale neural networks based on Runge-Kutta method for solving unsteady partial differential equations[J]. Journal of Xinjiang University(Natural Science Edition in Chinese and English), 2023, 40(2): 142-149.

Runge-Kutta 型多尺度神经网络求解非定常偏微分方程

陈泽斌, 冯新龙

(新疆大学 数学与系统科学学院, 新疆 乌鲁木齐 830017)

摘 要： 提出了基于 Runge-Kutta 的多尺度神经网络方法求解非定常偏微分方程. 利用 q 阶 Runge-Kutta 构造时间迭代格式, 通过建立多时间步的总损失函数, 实现多时间步的神经网络参数共享, 并预测时域内任意时刻的函数值. 同时采用 m-缩放因子加快损失函数收敛, 提高数值解精度. 最后, 给出了若干数值实验验证所提方法的有效性.

关键词： 非定常偏微分方程; q 阶 Runge-Kutta 法; 多尺度神经网络; m-缩放因子; 高精度

0 Introduction

Deep learning has achieved satisfactory results in searching technology, natural language processing, image processing, recommendation system, personalization technology, etc. In recent years, deep learning has been successfully applied to solve partial differential equations and has been deeply promoted. Compared with the finite element method[1] and the finite difference method[2], deep learning as a meshless method, can mitigate the curse of dimensionality when solving high-dimensional partial differential equations. So it is more convenient to establish a solution framework for high-dimensional partial differential equations. E etc[3] proposed the Deep-Ritz method based on deep neural networks for numerically solving variational problems, which was insensitive to the dimension of the problem and could be used to solve high-dimensional pr-

* **Received Date**: 2022-06-25

Foundation Item: This work was supported by Open Project of Key Laboratory of Xinjiang "Machine learning for incompressible magnetohydrodynamics models" (2020D04002).

Biography: CHEN Zebin(1995-), male, master student, research fields: deep learning to solve partial differential equations.

† **Corresponding author**: FENG Xinlong(1976-), male, professor, research field: numerical solutions of partial differential equations, E-mail: fxlmath@xju.edu.cn.

oblems. Physics-Informed Neural Networks(PINNs)[4] used the automatic differentiation technique[5] for the first time to embed the residual of the equation into the loss function of the neural networks, and obtained the numerical solution of the equation by minimizing the loss function. PINNs is a new numerical method for solving partial differential equations, which makes full use of the physical information contained in the PDEs. PINNs have attracted the attention of many scholars, and literature[6−7] shows theoretical convergence of PINNs for certain classes of PDEs. Multi-scale DNN[8] proposed the idea of radial scaling in the frequency domain, which had the ability to approximate high-frequency and high-dimensional functions, and could accelerate the convergence speed of the loss function.

Recently, the research on deep learning algorithms for nonlinear unsteady partial differential equations have attracted the attention of many scholars. The time-discrete model of PINNs can still guarantee the stability and high precision of numerical solutions when using large time steps, but it needs high computational costs. DeLISA added the physical information of the governing equations into the time iteration scheme, and introduced time-dependent input to achieve continuous-time prediction without a large number of interior points[9]. On this basis, this paper proposes a multi-scale neural networks algorithm integrating Runge-Kutta method. On the one hand, the algorithm constructs a time iteration scheme to build the total loss function of multiple time steps, thus realizes the sharing of neural networks parameters in multiple time steps to save computational costs. It can also predict the function value at any time in the time domain after training. On the other hand, the algorithm can not only speed up the convergence speed of the loss function, but also improve the accuracy of the solution. In the numerical example, we compare the stability of the time stepping scheme and the time iteration scheme in multi-time-step solutions. Then, through the sensitivity analysis of boundary points and initial points, we find that the solution accuracy does not increase with the increase of points. Therefore, we can still achieve the same calculation accuracy by selecting an appropriately small number of points, and achieving the purpose of reducing the amount of calculation.

The rest of this paper is organized as follows: In section 1, we describe neural networks combined with Runge-Kutta method and automatic differentiation in detail. Section 2 shows time iteration scheme based on Runge-Kutta multi-scale neural networks. We then present several numerical experiments, including Convection-Diffusion equation and Burgers equation in section 3. Finally, we conclude with the key ideas raised in this work.

1 Preliminaries

In this section, we introduce the neural networks integrates the Runge-Kutta[10] method to solve unsteady partial differential equations in detail. Automatic differentiation is briefly introduced, and the computational steps of the automatic differentiation are understood by calculating the derivatives of the output of a simple structured neural networks with respect to the input.

1.1 Fusion of Neural Networks and Runge-Kutta Method

To understand the key idea clearly, we consider a class of unsteady partial differential equations defined on the bounded set $\Omega \subset \mathbf{R}^n$

$$\begin{aligned} &u_t = \mathcal{N}(u(\mathbf{x},t)), && (\mathbf{x},t) \in \Omega \times (0,T] \\ &u(\mathbf{x},0) = u^0(\mathbf{x}), && \mathbf{x} \in \Omega \\ &u(\mathbf{x},t) = h(\mathbf{x},t), && (\mathbf{x},t) \in \partial\Omega \times [0,T] \end{aligned} \tag{1}$$

where $\mathbf{x} \in \Omega$ represents the space vector, and $\mathcal{N}$ represents the differential operator with respect to function u. We give the space discrete of equation (1), and then integrate it in the interval $[t_n, t_{n+1}]$ to get

$$u^{n+1} - u^n = \int_{t_n}^{t_{n+1}} \mathcal{N}[u(\mathbf{x}_k,t)]\mathrm{d}t, \qquad k = 1,2,\cdots,N_0,$$

where $t_n = n\Delta t$. We use the q-order implicit Runge-Kutta method to approximate the right-hand integral term in the above equation

$$\begin{aligned} u^{n+1} &= u^n + \Delta t \sum_{j=1}^{q} b_j \mathcal{N}[u^{n+c_j}] \\ u^{n+c_i} &= u^n + \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}[u^{n+c_j}], \quad i = 1,2,\cdots,q \end{aligned} \tag{2}$$

where $u^{n+c_i}(x) = u(t^n + c_i\Delta t, x)$, $0 < a_{ij}, b_j, c_j < 1$. As shown in Fig 1, we give the fusion frame diagram of the neural networks and the Runge-Kutta method when $\mathbf{x} = (x_1, x_2) \in \mathbf{R}^2$, $q = 5$.

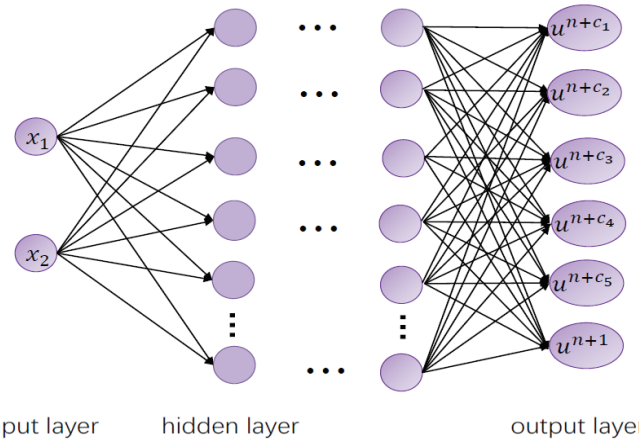

Fig 1 Fusion frame diagram of neural networks and Runge-Kutta method

From equation (2), we could include the indirect label of the output layer of the neural networks

$$u_{q+1}^n \triangleq u^{n+1} - \Delta t \sum_{j=1}^{q} b_j \mathcal{N}[u^{n+c_j}],$$

$$u_i^n \triangleq u^{n+c_i} - \Delta t \sum_{j=1}^{q} a_{ij} \mathcal{N}[u^{n+c_j}], \quad i = 1, 2, \cdots, q.$$

In the following, we establish the loss function for the n-th time layer

$$SSE_{0,n} = \sum_{k=1}^{N_0} \sum_{i=1}^{q+1} \left[u_i^n - u^n(\mathbf{x}_k)\right]^2,$$

where $u^n(\mathbf{x}_k)$ is the known function value of the n time step. Similarly, we set the loss function on the boundary

$$SSE_{b,n}(\theta) = \sum_{k=1}^{N_0} \sum_{i=1}^{q+1} \left[u^{n+c_i} - h(\mathbf{x}_k, t_n + c_j\Delta t)\right]^2,$$

where $u^{n+c_{q+1}} = u^{n+1}$. Then the total loss function of equation (1) consists of the above two parts

$$Loss_n = SSE_{0,n} + SSE_{b,n}.$$

1.2 Automatic Differentiation

PINNs embed the residual of the equation into the loss function of the neural networks for the first time, which provided an effective means to numerically solve partial differential equations. By calculating the derivative of the networks output with respect to the input, we can obtain the residuals of the equation. The numerical solution u_{ANN} constructed by the neural networks have a specific functional expression, so the derivative can be calculated using finite difference method, symbolic differentiation or automatic differentiation. However, the limitations of the finite difference method lie in truncation error and method error. Symbolic differentiation is computationally expensive and time-consuming. Automatic differentiation can overcome the limitations of the above two methods, so this paper uses the automatic differentiation technique to calculate the derivative.

Automatic differentiation(AD) computes derivatives using the chain rule. AD calculation process can be divided into two steps: calculating the function value in the forward mode and calculating the derivative value in the reverse mode. We learn about AD by computing the derivative of the output of a multi-layer feedforward neural networks with respect to the input. The neural networks consists of an input layer with two neurons x_1 and x_2, a hidden layer with one neuron and an output layer with one neuron y.

$$y = 2\tanh(0.1x_1 - 0.3x_2 + 0.6) - 0.3.$$

The calculation of the derivative of y with respect to (x_1, x_2) at (2,3) is shown in Table 1.

Table 1 Automatic differentiation technique derivation process

Forward Pass	Backward Pass
$x_1 = 2$	$\frac{\partial y}{\partial y} = 1$
$x_2 = 3$	
$z = 0.1x_1 - 0.3x_2 + 0.6 = -0.1$	$\frac{\partial y}{\partial \tilde{z}} = \frac{\partial(2\tilde{z}-0.3)}{\partial \tilde{z}} = 2$
$\tilde{z} = \tanh(z) \approx -0.099\ 7$	$\frac{\partial y}{\partial z} = \frac{\partial y}{\partial \tilde{z}}\frac{\partial \tilde{z}}{\partial z} = \frac{\partial y}{\partial \tilde{z}}\operatorname{sech}^2(\tilde{z}) \approx 1.980\ 3$
$y = 2\tilde{z} - 0.3 = -0.499\ 3$	$\frac{\partial y}{\partial x_1} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial x_1} = \frac{\partial y}{\partial z} * 0.1 \approx 0.198\ 0$
	$\frac{\partial y}{\partial x_2} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial x_2} = \frac{\partial y}{\partial z} * (-0.3) = -0.594\ 1$

Iri etc[11] give the proof of the computational complexity of AD which proves that the computation of the gradient is at most 5 times that of the function, regardless of the dimension of the independent variable. For an optimization problem with a 100-dimensional smooth objective function, we use symbolic differentiation or numerical differentiation to compute the gradient, and then the gradient is at least 100 times more computable than the function value, not to mention the use of second-order derivative information or higher-dimensional functions. If AD is used, regardless of the dimension of the independent variable, the computation amount of the gradient is at most 4 times than the value of the function with machine precision.

2 Multi-Scale Neural Networks Integrating Runge-Kutta Method

According to the universal approximation theorem[12], as long as a hidden layer contains enough neurons, the neural networks can approximate a continuous function defined on a compact set with arbitrary accuracy. It should be noted that the neural networks obeys the frequency principle[13] when fitting the function: it tends to preferentially fit the low-frequency part of the objective function. Based on the understanding of the frequency principle, in order to solve the characteristics of slow learning at high-frequency, we can make some special designs for the neural networks. From the point of view of function space, the basis function of neural networks is composed of activation functions. The basis functions of different scales construct a feasible function space, which can approximate the objective function faster. The same basis function can be scaled to generate different scales basis functions. In order to describe the multi-scale neural network conveniently, we only divide the neurons into h parts in the first hidden layer, as shown in Fig 2. The input of the i-th neuron is ix, and the corresponding output is $\sigma(iwx+b)$. Then the multi-scale neural network with L-1 hidden layers is defined as

$$\mathbf{u}_{\mathrm{ANN}} = \mathbf{W}_L\sigma\circ(\mathbf{W}_{L-1}\sigma\circ(\cdots\sigma(\mathbf{W}_1\sigma\circ(\mathbf{M}\odot\mathbf{W_0}\mathbf{x}+\mathbf{b_0})+\mathbf{b}_1)\cdots)+\mathbf{b}_{L-1}),$$

where $\odot$ is the Hadamard product, m-scaling factor is $\mathbf{M} = m\times(1,1,\cdots,1,2,\cdots,2,\cdots i,\cdots,i,\cdots,h,\cdots,h)$.

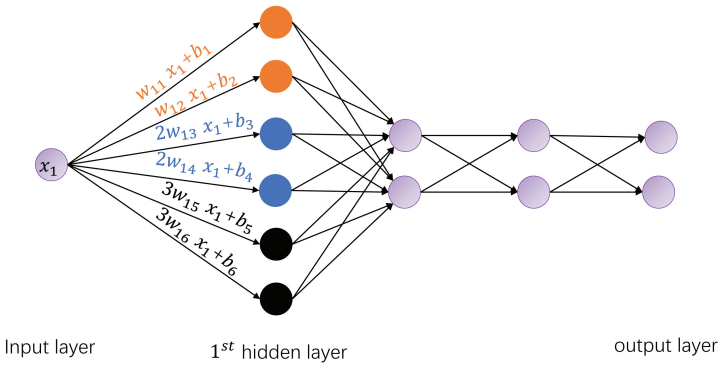

Fig 2 Multi-scale neural network example of h=3

The m-scaling factor of multi-scale neural networks can not only speed up the convergence speed of the loss function, but also improve the accuracy of the solution. In Fig 3, we present a schematic diagram of the framework of time iteration scheme based on multi-scale neural networks. The input-output mapping of a multi-scale neural network is

$$(\mathbf{x}, t_n) \to (u^{n+c_1}, u^{n+c_2}, \cdots, u^{n+c_q}, u^{n+1}).$$

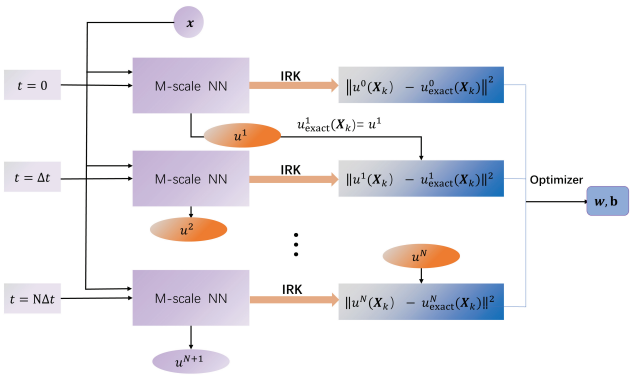

Fig 3 Time iteration scheme of neural networks combined with Runge-Kutta method

In Fig 4, we give a schematic diagram of the time step scheme of the neural networks and implicit Runge-Kutta(IRK) fusion. It is worth noting that the iterative solution of each time step in this framework requires a neural network to fit. The iterative solution of multiple time steps needs to optimize the parameters of multiple neural networks. Time iteration scheme makes full use of the approximation ability of the neural networks by adjusting the input-output mapping relationship of the multi-scale neural networks, so that the iterative solutions of multiple time steps share the parameters. At n-th time step, we set the loss function as follows

$$loss_n = \sum_{k=1}^{N_0} \sum_{i=1}^{q+1} \left[u_i^n(\mathbf{x}_k) - u_{\text{exact}}^{n,i}(\mathbf{x}_k) \right]^2.$$

We set the loss function for boundary conditions as follows

$$loss_b = \sum_{n=1}^{N} \sum_{k=1}^{N_b} \sum_{i=1}^{q+1} \left[u_i^n(\mathbf{x}_k) - h^{n,i}(\mathbf{x}_k) \right]^2.$$

Similarly, we set the loss function for the initial condition as follows

$$loss_0 = \sum_{k=1}^{N_b} \sum_{i=1}^{q+1} \left[u_i^0(\mathbf{x}_k) - u^{0,i}(\mathbf{x}_k) \right]^2.$$

So the total loss function can be written as

$$loss = \sum_{n=1}^{N} loss_n + loss_b + loss_0.$$

The multi-scale neural networks algorithm incorporating the Runge-Kutta method requires only a small number of initial value data pairs, constructs its time iteration scheme by IRK method, and then builds the total loss function for multiple time steps to obtain the function value at any time in the time domain by only one time optimization.

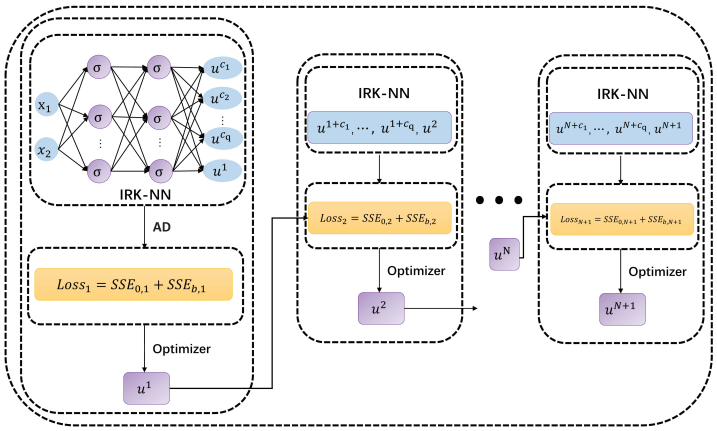

Fig 4 Time stepping scheme of neural networks combined with implicit Runge-Kutta method

3 Numerical Experiments

In this section, in order to verify the effectiveness of the time iteration scheme algorithm based on multi-scale neural networks, we give numerical experiments to solve the Convection-Diffusion equation and Burgers equation respectively, and define the relative L_2 error as

$$\text{Error}_{(L_2)} = \frac{\left[\sum_{i=1}^{N}\left(u_{\text{exact}}(\mathbf{x}_i)-u_{\text{pred}}(\mathbf{x}_i)\right)^2\right]^{1/2}}{\left[\sum_{i=1}^{N}\left(u_{\text{exact}}(\mathbf{x}_i)\right)^2\right]^{1/2}}.$$

3.1 Convection-Diffusion Equation

Convection-Diffusion equation[14] is widely used to describe fluid models and many physical phenomena. The unknown function u can be used to express the contaminant concentration transported through the fluid. Of course, it can also be expressed as the temperature of the fluid moving along the hot wall or the electron concentration in a semiconductor device model. To verify the stability of time iteration scheme, let's consider a two-dimensional Convection-Diffusion equation.

$$u_t = \triangle u - \beta\cdot\nabla u, \qquad (\mathbf{x},t)\in\Omega\times(0,T] \tag{3}$$

where the solution area of the equation is defined as $\Omega=[-1,3]^2$, $T=1$, and $\beta=(a,b)$. The initial boundary value conditions for this problem are given by the following true solutions

$$u(x,y,t)=\frac{10}{4t+1}\exp\left[-\frac{(x-at-0.5)^2}{4t+1}-\frac{(y-bt-0.5)^2}{4t+1}\right],$$

where $a=b=4/5$. Specifically we choose 5 hidden layers neural network with 60 neurons in each hidden layer, and tanh as activation function. The training set is composed of $N_0=16\times16$ initial value condition data pairs and $N_b=50\times4$ boundary points data pairs. We use implicit Runge-Kutta method with $q=4$, and take the time step $\Delta t=0.2$. After establishing the loss function through time iteration scheme, we employ the L-BFGS-B algorithm to optimize. In Table 2, we apply time step scheme(TSS) and time iteration scheme(TIS) to solve the Convection-Diffusion equation, respectively, and then give a comparison of the relative L_2 error at different times by strictly controlling the variables. Compared with the traditional discrete method, the relative L_2 error of the TSS numerical solution method increases with the iteration of the time step. The numerical accuracy of the iterative solution of TIS at multiple time steps is significantly better than that of TTS.

Table 2 Comparison of the relative L_2 error of TSS and TIS in solving the Convection-Diffusion equation at different times

	$t=0.2$	$t=0.4$	$t=0.6$	$t=0.8$	$t=1.0$
TSS	1.560×10^{-4}	6.904×10^{-2}	1.556×10^{-1}	2.432×10^{-1}	3.248×10^{-1}
TIS	3.196×10^{-4}	2.453×10^{-4}	1.855×10^{-4}	1.332×10^{-4}	1.764×10^{-4}

3.2 2D Burgers Equation

Burgers equation is a kind of basic partial differential equation that often appears in many fields of applied mathematics. It is widely used in fluid mechanics, nonlinear acoustics, aerodynamics and traffic flow. It is the result of the combination of nonlinear waves and linear diffusions and is the simplest model to analyze the combined effects of linear advection and diffusion. Next, we consider the following two-dimensional Burgers equation[15] to compare the difference in the convergence speed and solution accuracy of the loss function using a multi-scale neural network and a fully connected neural network.

$$u_t = 0.1\cdot\triangle u - \beta\cdot\nabla u, \qquad (\mathbf{x},t)\in\Omega\times(0,T] \tag{4}$$

where $\beta=(u(x,y),1)^T$, $\Omega=[0,1]\times[0,1]$, and $T=2$. The initial boundary value condition of equation (4) is given by the following analytical solution

$$u(x,y,t)=1/\left(1+\exp\left(\frac{(x+y-t)}{0.2}\right)\right).$$

In this example, the multi-scale neural network is structured with 6 hidden layers and 50 neurons in each hidden layer. At the same time, we select the tanh function as the activation function, and the m-scaling factor of the multi-scale neural

network is $\mathbf{M}=m\times(1,1,2,2,\cdots,i,i,\cdots,25,25)$, then the mathematical expression of the multi-scale neural network is

$$\mathbf{u}_{\mathrm{ANN}}=\mathbf{W}_L\sigma\circ(\mathbf{M}\odot\mathbf{W}_{L-1}\sigma\circ(\cdots\sigma(\mathbf{M}\odot\mathbf{W}_1\sigma\circ(\mathbf{M}\odot\mathbf{W_0}\mathbf{x}+\mathbf{b_0})+\mathbf{b}_1)\cdots)+\mathbf{b}_{L-1}).$$

We set the hyperparameter $m=0.06$, $q=4$, the time step is $\Delta t=0.2$. The training set consists of $N_0=30\times30$ initial value condition data pairs and $N_b=50\times4$ boundary data pairs. In Fig 5, we give the comparison of the convergence speed of the loss function when time iteration scheme selects the fully connected neural network(FNN) and the multi-scale neural network(NN) as the approximator respectively. We found that its loss function has a faster convergence rate when using a multi-scale NN.

In Table 3, we give the comparison of the relative L_2 error at different times when the FNN and the multi-scale NN are selected as the approximator in time iteration scheme(TIS). We can see that when using the multi-scale NN, the accuracy of the solution is higher and is improved by 64.7% at the final moment $T=2$. In Table 4, we perform sensitivity analysis on the number of boundary points and initial points, and give the relative L_2 error at the last moment $T=2$. We see that the accuracy of the solution is not significantly improved with the increase in the number of points. So we can select an appropriately small number of points to reduce the amount of calculation and achieve the required accuracy.

Table 3 Comparison of relative L_2 error at different times for solving equations with two network structures

	$t=0.2$	$t=0.5$	$t=0.8$	$t=1.5$	$t=1.8$	$t=2.0$
TIS	3.030×10^{-4}	2.011×10^{-4}	1.621×10^{-4}	9.934×10^{-5}	9.834×10^{-5}	9.719×10^{-5}
TIS-M	2.393×10^{-4}	1.182×10^{-4}	1.154×10^{-4}	7.417×10^{-5}	4.829×10^{-5}	3.429×10^{-5}

Table 4 Sensitivity analysis of the number of interior points and boundary points

N_b	N_0					
	5×5	10×10	20×20	30×30	50×50	70×70
104	4.493×10^{-5}	2.674×10^{-5}	4.126×10^{-5}	4.989×10^{-5}	7.794×10^{-5}	8.085×10^{-5}
204	3.508×10^{-5}	1.862×10^{-5}	4.901×10^{-5}	3.430×10^{-5}	1.130×10^{-4}	8.390×10^{-5}
404	3.449×10^{-5}	3.668×10^{-5}	3.659×10^{-5}	3.112×10^{-5}	6.336×10^{-5}	3.839×10^{-5}

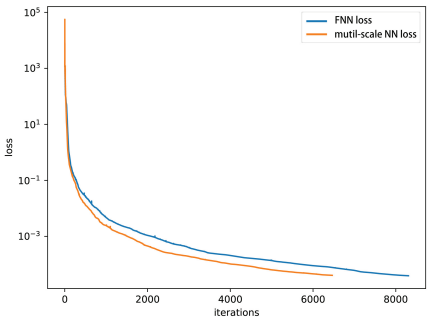

Fig 5 Comparison of convergence speed of loss function between FNN and multi-scale NN

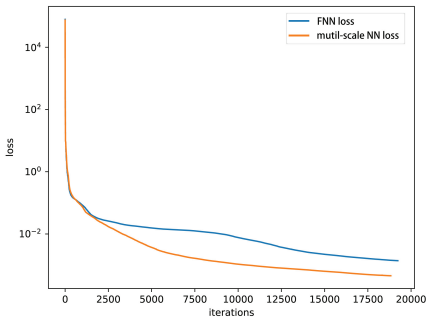

Fig 6 Comparison of convergence speed of loss function between FNN and multi-scale NN

3.3 3D Burgers Equation

we consider the following three-dimensional Burgers equation

$$\mathbf{u}_t=\Delta\mathbf{u}-(\mathbf{u}\cdot\nabla)\mathbf{u},\quad(\mathbf{x},t)\in\Omega\times(0,T] \tag{5}$$

where $\Omega=[0,1]^3$, $T=0.1$. The initial boundary value condition of the equation is given by the following analytical solution

$$\mathbf{u}=[u,v,w]^T=-\frac{2\epsilon}{\phi}\left[1+e^{-t}\cos(x)\sin(y)\sin(z),1+e^{-t}\sin(x)\cos(y)\sin(z),1+e^{-t}\sin(x)\sin(y)\cos(z)\right]^T.$$

where $\phi(x,y,z,t)=1+x+e^{-t}\sin(x)\sin(y)\sin(z)$, $\epsilon=0.05$. In this example, we set the structure of the multi-scale NN with 2 hidden layers, each layers with 100 neurons. At the same time, we choose tanh as the activation function, and set the time step to $\Delta t=0.005$, q=2. The m-scaling factor of the multi-scale NN is $\mathbf{M}=m\times(1,1,2,2,\cdots,i,i,\cdots,25,25)$, where the

hyperparameter $m = 0.06$. The training set consists of $N_0 = 5\times5\times5$ initial moment data pairs and $N_b = 21\times21\times6$ boundary data pairs. In Fig 6, we give a comparison of the convergence speed of the loss function when TIS selects the FNN and the multi-scale NN as the approximator respectively. When we use a multi-scale NN, its loss function has a faster convergence rate as a whole. In Table 5, we give a comparison of the relative L_2 error at different times when TIS selects the FNN and the multi-scale NN as the approximator respectively. We see that when using a multi-scale NN, its solution accuracy is higher, and the accuracy of u, v, w at the final moment $T = 0.1$ is increased by $45.3\%, 51.3\%, 61.5\%$, respectively.

Table 5 Comparison of relative L_2 error at different times with two network structures

t	FNN			multi-scale NN		
	L_u^2 error	L_v^2 error	L_w^2 error	L_u^2 error	L_v^2 error	L_w^2 error
0.005	1.388×10^{-3}	1.312×10^{-3}	2.059×10^{-3}	8.022×10^{-4}	6.448×10^{-4}	7.339×10^{-4}
0.03	1.262×10^{-3}	1.202×10^{-3}	1.914×10^{-3}	7.253×10^{-4}	5.737×10^{-4}	6.758×10^{-4}
0.05	1.229×10^{-3}	1.144×10^{-3}	1.828×10^{-3}	6.980×10^{-4}	5.475×10^{-4}	6.376×10^{-4}
0.08	1.278×10^{-3}	1.139×10^{-3}	1.737×10^{-3}	7.052×10^{-4}	5.544×10^{-4}	6.147×10^{-4}
0.10	1.388×10^{-3}	1.216×10^{-3}	1.700×10^{-3}	7.594×10^{-4}	5.917×10^{-4}	6.553×10^{-4}

4 Conclusion

With the wide application of deep learning in many fields, our research on solving partial differential equations with deep learning becomes more important. We combine the neural networks with the traditional Runge-Kutta method, and establish a time iteration scheme approximation algorithm based on the multi-scale NNs, which alleviates the degradation of the accuracy of the numerical solution with time iteration to a certain extent. The introduction of the multi-scale NNs ensures that the convergence speed of the loss function is accelerated, and the accuracy of the numerical solution is also improved. In this way, we organically combine classical mathematical methods with NNs to provide new ideas for the numerical solution of partial differential equations. Since the time iteration scheme requires setting time-dependent input, sharing the parameters of the neural networks among multiple time step solutions. In future work, we continue to consider long short term memory networks and convolutional neural networks.

References:

[1] 金孟晴, 冯新龙, 何银年. 曲面上对流-扩散-反应方程的两种稳定化混合有限元方法的数值比较[J]. 新疆大学学报(自然科学版)(中英文), 2022, 39(3): 266-274+282.

[2] FENG X L, HE Y N. H^1-Super-convergence of center finite difference method based on P_1-element for the elliptic equation[J]. Applied Mathematical Modelling, 2014, 38(23): 5439-5455.

[3] E W N, YU B. The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems[J]. Communications in Mathematics and Statistics, 2018, 6(1): 1-12.

[4] RAISSI M, PERDIKARIS P, KARNIADAKIS G E. Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations[J]. Journal of Computational Physics, 2019, 378: 686-707.

[5] HOFFMANN P H W. A hitchhiker's guide to automatic differentiation[J]. Numerical Algorithms, 2016, 72(3): 775-811.

[6] MISHRA S, MOLINARO R. Estimates on the generalization error of physics-informed neural networks for approximating PDEs[J]. IMA Journal of Numerical Analysis, 2022, 42(2): 981-1022.

[7] DE R T, MISHRA S. Error analysis for physics informed neural networks (PINNs) approximating Kolmogorov PDEs[J]. Advances in Computational Mathmatics, 2022, 48(6):1-40.

[8] LIU Z, CAI W, XU Z. Multi-scale deep neural network (MscaleDNN) for solving Poisson-Boltzmann equation in complex domains[J]. Communications in Computational Physics, 2020, 28(5): 1970-2001.

[9] LI Y, ZHOU Z J, YING S H. DeLISA: deep learning based iteration scheme approximation for solving PDEs[J]. Journal of Computational Physics, 2022, 451: 110884.

[10] ISERLES A. A first course in the numerical analysis of differential equations[M]. Cambridge: Cambridge University Press, 2009.

[11] IRI M, TANABE K. Mathematical programming: recent developments and applications[J]. Mathematics of Computation, 1990, 55(192): 262-279.

[12] HORNIK K, STINCHCOMBE M, WHITE H. Multilayer feedforward networks are universal approximators[J]. Neural Networks, 1989, 2(5): 359-366.

[13] XU Z Q, ZHANG Y Y, LUO T, et al. Frequency principle: fourier analysis sheds light on deep neural networks[J]. IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences, 2020, 28(5): 1746-1767.

[14] ZHAI S Y, GUI D W, HUANG P Z, et al. A novel high-order ADI method for 3D fractional convection-diffusion equations[J]. International Communications in Heat and Mass Transfer, 2015, 66: 212-217.

[15] YANG X, GE Y, ZHANG L. A class of high-order compact difference schemes for solving the Burgers equations[J]. Applied Mathematics and Computation, 2019, 358: 394-417.

责任编辑： 张自强 刘敏