OXFORD

Journal of Geophysics and Engineering

Journal of Geophysics and Engineering (2024) **21**, 1194–1215

https://doi.org/10.1093/jge/gxae062
Advance access publication 11 June 2024

A deep learning operator-based numerical scheme method for solving 1D wave equations

Yunfan Chang [1], **Dinghui Yang**[1,*] **and Xijun He** [2]

[1] Department of Mathematical Sciences, Tsinghua University, 100084 Beijing, China
[2] School of Mathematics and Statistics, Beijing Technology and Business University (BTBU), 100048 Beijing, China

*Corresponding author. E-mail: ydh@tsinghua.edu.cn

Received: January 16, 2024. **Revised:** April 28, 2024. **Accepted:** June 10, 2024

Abstract

In this paper, we introduce the deep numerical technique DeepNM, which is designed for solving one-dimensional (1D) hyperbolic conservation laws, particularly wave equations. By creatively integrating traditional numerical schemes with deep learning techniques, the method yields improvements over conventional approaches. Specifically, we compare this approach against two established classical numerical methods: the discontinuous Galerkin method (DG) and the Lax–Wendroff correction method (LWC). While maintaining a comparable level of accuracy, DeepNM significantly improves computational speed, surpassing conventional numerical methods in this aspect by more than tenfold, and reducing storage requirements by over 1000 times. Furthermore, DeepNM facilitates the utilization of higher-order numerical schemes and allows for an increased number of grid points, thereby enhancing precision. In contrast to the more prevalent PINN method, DeepNM optimally combines the strengths of conventional mathematical techniques with deep learning, resulting in heightened accuracy and expedited computations for solving partial differential equations. Notably, DeepNM introduces a novel research paradigm for numerical equation-solving that can be seamlessly integrated with various traditional numerical methods.

Keywords: Deep learning; wave equations; discontinuous Galerkin method; lax–Wendroff correction method

1. Introduction

Numerical techniques have proven their indispensability in the solution of wave equations, particularly within the realm of exploration geophysics. Within the scope of forward modeling, the wave equation serves as a vital interpretive tool for deciphering complex geological structures and serves as a benchmark for testing processing algorithms. In the context of waveform inversion, which aims to elucidate the Earth's subsurface structure, the accurate and efficient computation of synthetic seismograms assumes paramount importance. Among the commonly employed approximate methods for wave equation solving, finite-difference (FD) and finite-element (FE) methods stand out. This paper employs two representative techniques, namely the Lax–Wendroff correction method (LWC) and the discontinuous Galerkin method (DG), which are categorized within the FD and FE families, respectively.

The LWC, an extension of the FD method for solving hyperbolic partial differential equations (PDEs), was initially proposed by Peter Lax and Burton Wendroff in 1960 (Lax 1959). This method was born from the need to enhance the accuracy and stability of numerical techniques for PDE solution, particularly in computational fluid dynamics and related fields (Burstein 1966, Jameson 1985, Dablain 1986). DG, known for its accuracy, has gained significant prominence in the realm

of computational geophysics. DG was first introduced in 1973 by Reed and Hill in the context of solving the neutron transport equation (Reed and Hill 1973). Over the course of several decades, various variants of DG have surfaced (Cockburn and Shu 1989, 1991, 1998, Cockburn *et al.* 1989, 1990). DG has found rapid applications across a wide spectrum of fields, including aeroacoustics (Atkins and Atkins 1997, Dumbser and Munz 2005), gas dynamics (Van der Vegt and Van der Ven 2002, Engsig-Karup *et al.* 2008), and magneto-hydrodynamics (Hesthaven and Warburton 2002, Lu *et al.* 2004).

Notwithstanding their precision, traditional numerical methods encounter challenges related to computational efficiency and storage demands when applied to large-scale computations. While accurate solutions for seismic wave equations in a forward modeling context have become attainable, realistic conditions and inverse problems present formidable computational hurdles. In large-scale seismic wavefield simulations, wave equation-based seismic migration, and full waveform inversion, traditional methods exhibit extremely low computational efficiency. The substantial cost associated with classical numerical methods can be primarily attributed to the stringent Courant–Friedrichs–Lewy (CFL) conditions (Courant *et al.* 1928, Shu 1988). Both FD and FE methods necessitate the use of small time-steps, resulting in increased iterations and extended computational time. Additionally, traditional methods entail substantial storage requirements, as they must retain coefficients for all elements at each time layer, resulting in an inefficient utilization of storage resources.

Deep neural networks have demonstrated remarkable success in the field of artificial intelligence in recent years (LeCun *et al.* 2015, Goodfellow *et al.* 2016, Silver *et al.* 2016). These research breakthroughs have sparked high expectations for the application of deep learning in solving PDEs. Numerous deep learning-based approaches have been developed for the resolution of PDEs, particularly in higher dimensions (Han *et al.* 2018, Sirignano and Spiliopoulos 2018, Raissi *et al.* 2019, Meng *et al.* 2020). Some of these methods excel in both accuracy and speed. From a mathematical perspective, universal approximation theorems affirm that a single hidden-layer neural network can effectively approximate a wide range of functions on compact subsets (Pinkus 1999). This underpins the neural network's capacity for approximation. Nonetheless, many deep learning-based methods devise loss functions directly from equations, bypassing the need for traditional numerical frameworks (Han *et al.* 2017, Asem 2020, Raissi *et al.* 2020). While some of these techniques have exhibited promise in addressing specific problems, they might not be sufficiently sophisticated for tackling wave equations, especially those encountered in exploration geophysics. The majority of existing research into deep learning-based PDE methods concentrates on higher-dimensional challenges rather than the domain of seismic wave equations.

In this paper, we introduce an approach to solving wave equations by employing deep learning-based numerical techniques. This innovative algorithm combines deep neural networks with specific numerical schemes, such as the deep discontinuous Galerkin method (DeepDG) and deep Lax–Wendroff correction method (DeepLWC), collectively referred to as deep numerical methods (DeepNM). This nomenclature arises from the incorporation of neural network function representations within the traditional numerical method context. The DeepNM method amalgamates the loss function from deep learning with the numerical iteration scheme found in traditional methods. When the loss function converges to zero, the results obtained through our method align with those of conventional techniques. In our approach, a deep neural network is trained to adhere to the numerical iterative scheme, utilizing the stochastic gradient descent technique across randomly sampled spatial and temporal points. The efficiency of our training process is enhanced by introducing random space and time sampling in each training epoch. Furthermore, we concurrently train the coefficients of basis functions of different orders. Our investigation demonstrates the promising potential of DeepNM for more efficient and precise solutions to hyperbolic conservation law equations.

2. The deep numerical methods

In this work, we consider the wave equations of the general form:

$$\mathcal{L}u = f(t,x), \qquad x \in \Omega, \quad 0 \le t \le T, \tag{1}$$

with initial and boundary conditions

$$\begin{aligned} u(0,x) &= g(x), && x \in \Omega, \\ u(t,x) &= h(t,x), && x \in \partial\Omega, \quad 0 \le t \le T, \end{aligned} \tag{2}$$

where $\mathcal{L}$ is a nonlinear differential operator. The Burgers' equations and acoustic wave equations studied in this study are both of this type of equation (for Burgers' equations $\mathcal{L}u = u_t + uu_x$, and for acoustic wave equations $\mathcal{L}u = u_{tt} - c^2 u_{xx}$).

We take the acoustic equation as an example to introduce the deep numerical method. In an homogeneous isotropic medium, the 1D acoustic wave equation can be described by the following equation:

$$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = f(t,x), \qquad x \in \Omega, \quad 0 \le t \le T, \tag{3}$$

where u is pressure, c is the acoustic velocity in a medium, f is an external source, Ω is the computational domain, and T is termination time.

DeepNM has to be combined with the specified traditional numerical methods. This work showcases its compatibility with two widely used methods—LWC (a finite difference method) and DG (a finite element method)—by implementing DeepDG and DeepLWC, respectively. DeepNM can incorporate various numerical formats by utilizing these examples seamlessly. For convenience, we use the DG as an example to introduce the DeepNM.

2.1. *Deep neural network*

The deep neural network (DNN) is commonly depicted as the composition of multiple distinct functions, as described in (Goodfellow *et al.* 2016). A DNN comprises a sequence of layers, with each layer housing numerous neurons interconnected with both preceding and succeeding layers. Neurons are linked through affine transformations and nonlinear activation functions. To be more specific, in the context of this study, the DNN input is represented as $\mathbf{z}^0 = (t, \mathbf{x})$, and the output is denoted as $\mathcal{N}(t, \mathbf{x})$. The recursive transformations during the propagation phase can be expressed as follows:

$$\begin{aligned} &\mathbf{z}^0 = (t, \mathbf{x}), \quad \text{(input)}, \\ &\mathbf{z}^{l+1} = a_l\left(\mathbf{w}^{l+1} \cdot \mathbf{z}^l + \mathbf{b}^l\right), \quad l = 0, 1, \ldots L-1 \\ &\mathcal{N}(t, \mathbf{x}) = \mathbf{w}^{L+1}\mathbf{z}^L, \quad \text{(output)}, \end{aligned} \tag{4}$$

where L is the depth of the model, $\mathbf{w}^{l+1} \in \mathbb{R}^{m_{l+1} \times m_l}$ is the weight of layer $l+1$, m_l is the width of layer l, $\mathbf{b}^l \in \mathbb{R}^{m_{l+1}}$ is the bias of layer l, a_l is the activation function, and $\mathbf{w}^{L+1} \in \mathbb{R}^{m_L}$ is the weight of output layer. Our experience has suggested that the choice of activation function a_l plays a crucial role in the accuracy of algorithm. To balance simplicity and accuracy, we have decided to use the swish activation function given explicitly by

$$a_l(x) = \text{swish}(x) := x\,\text{sigmoid}(x) = \frac{x}{1+\exp(-x)}, \quad l = 0, 1, \ldots, L-1. \tag{5}$$

Presently, the most prevalent and effective activation function in use is the Rectified Linear Unit (ReLU), defined as $\max(0, x)$. However, in our numerical experiments, we cannot employ this specific activation function due to its tendency to introduce oscillations in the computed first-order derivatives and to cause second-order derivatives with respect to the inputs to become null, as noted in Raissi *et al.* (2020). Let θ encompass all the parameters $\mathbf{w}^L$ and $\mathbf{b}^l$, which are to be optimized by minimizing the loss function to align the DNN solution with the target function $u(t, \mathbf{x})$.

It is worth noting that, according to the universal approximation theorem (Pinkus 1999), even a network with just a single hidden layer is sufficient to fit the training set. Deeper networks often require fewer units per layer and fewer parameters. Montufar *et al.* (2014) demonstrated that functions that can be represented by a deep rectifier network may demand an exponential number of hidden units when using a shallow network with just one hidden layer. However, training deeper neural networks can be more challenging. To address this, He *et al.* (2016) proposed a residual learning framework (ResNet) to mitigate the degradation problem. In our study, we incorporate residual connections to enhance the network's ease of training and to circumvent the vanishing gradient issue. The network comprises two residual blocks, as depicted in Fig. 1.

2.2. *Discontinuous Galerkin method*

Following He *et al.* (2015), we rewrite equation (3) by introducting variables p which satisfy $\partial p/\partial t = \partial u/\partial x$. Then, after an integration in time, equation (3) can be rewritten as follows:

$$\frac{\partial}{\partial t}\begin{pmatrix} u \\ p \end{pmatrix} + \frac{\partial}{\partial x}\begin{pmatrix} -c^2 p \\ -u \end{pmatrix} = \begin{pmatrix} \hat{f} \\ 0 \end{pmatrix}, \tag{6}$$

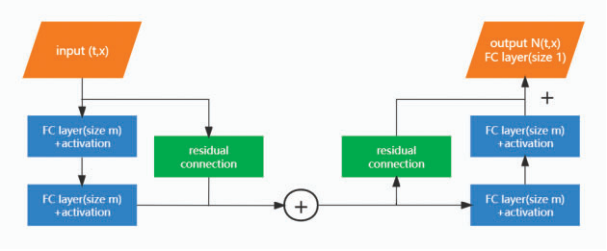


Figure 1. A network with two blocks and an output fully connected layer. Each block consists of two fully connected (FC) layers and a skip connection. The plus sign inside the straight line stands for the addition of the fully connected layer and the residual connection.

where $\hat{f}$ is the integral of f in time. Let

$$\mathbf{W} = \begin{pmatrix} u \\ p \end{pmatrix}, \quad \mathbf{F}(\mathbf{W}) = -\begin{pmatrix} -c^2 p \\ -u \end{pmatrix}, \quad \mathbf{f} = \begin{pmatrix} \hat{f} \\ 0 \end{pmatrix}, \tag{7}$$

then equation (6) can be presented as the following hyperbolic conservation law:

$$\frac{\partial \mathbf{W}}{\partial t} + \nabla \cdot \mathbf{F}(\mathbf{W}) = \mathbf{f}, \tag{8}$$

where $\mathbf{F}(\mathbf{W})$ is called the flux vector. It should be noted that the Deep DG method works for any hyperbolic conservation law equation, although we are using the acoustic wave equation as an example.

Assume the computational domain $\Omega \in \mathbb{R}^2$ is divided into non-overlapping and conforming sub-elements $\{\Omega_i\}$. The scalar test function space of the DGM is $V_h = \{v \in L^1(\Omega) : v|_{\Omega_i} \in P^k(\Omega_i)\}$, where $P^k(\Omega_i)$ is a polynomial space defined on Ω_i of orders no more than k. Multiplying equation (8) by a time-independent scalar test function v and integrating over Ω_i gives

$$\int_{\Omega_i} \left(v \frac{\partial \mathbf{W}}{\partial t} + v \nabla \cdot \mathbf{F}(\mathbf{W}) \right) \mathrm{d}V = \int_{\Omega_i} \mathbf{f} v \, \mathrm{d}V. \tag{9}$$

Using Green's formula, equation (9) is given by

$$\int_{\Omega_i} \left(v \frac{\partial \mathbf{W}}{\partial t} - \mathbf{F}(\mathbf{W}) \cdot \nabla v \right) \mathrm{d}V + \int_{\partial \Omega_i} v \mathbf{F}(\mathbf{W}) \cdot \mathbf{n} \mathrm{d}S = \int_{\Omega_i} \mathbf{f} v \, \mathrm{d}V, \tag{10}$$

where $\partial\Omega_i$ is the boundary of Ω_i and $\mathbf{n}$ is the outward unit vector normal to $\partial\Omega_i$. Since $\mathbf{W}$ may be discontinuous across Ω_i, we need to replace the flux by a numerical flux $\hat{\mathbf{F}}(\mathbf{W}_i, \mathbf{W}_e, \mathbf{n})$, where $\mathbf{W}_i$ and $\mathbf{W}_e$ represent the approximations of $\mathbf{W}$ on $\partial\Omega_i$ inside and outside Ω_i, respectively. In order to keep the stability and convergence of the numerical scheme, the numerical flux should have the properties of Lipschitz, consistency, conservation and monotone (Toro and Solvers 1999, LeVeque *et al.* 2002). In this study, we focus on the Godunov flux and the local Lax–Fredrichs flux (Cockburn and Shu 1989, 1998), which reads:

- Godunov flux:

$$\hat{\mathbf{F}}^G(\mathbf{a}, \mathbf{b}, \mathbf{n}) = \begin{cases} \min_{\mathbf{a} \le \mathbf{W} \le \mathbf{b}} \mathbf{F}(\mathbf{W}) \cdot \mathbf{n}, & \text{if} \quad \mathbf{a} \le \mathbf{b}, \\ \max_{\mathbf{b} \le \mathbf{W} \le \mathbf{a}} \mathbf{F}(\mathbf{W}) \cdot \mathbf{n}, & \text{otherwise.} \end{cases}, \tag{11}$$

- Local Lax–Friedrichs flux:

$$\hat{\mathbf{F}}^{LLF}(\mathbf{a}, \mathbf{b}, \mathbf{n}) = \frac{1}{2}(\mathbf{F}(\mathbf{a}) + \mathbf{F}(\mathbf{b})) \cdot \mathbf{n} - \frac{C}{2}(\mathbf{b} - \mathbf{a}), \tag{12}$$

where constant C is the biggest eigenvalue (in absolute value) of $(\frac{\partial}{\partial \mathbf{W}})\mathbf{F}(\mathbf{W}_i) \cdot \mathbf{n}$ and $(\frac{\partial}{\partial \mathbf{W}})\mathbf{F}(\mathbf{W}_e) \cdot \mathbf{n}$.

We choose the basis function φ_i^j to be the orthogonal Legendre polynomial of order j on element Ω_i (0 on other elements). We can write the DG approximation as

$$\mathbf{W}_h(t,\mathbf{x}) = \sum_{j=0}^{k}\sum_{i=1}^{M} C_i^j(t)\varphi_i^j(\mathbf{x}), \tag{13}$$

where M is the number of elements and $C_i^j(t)$ is the time-dependent coefficient. Using equation (13), we replace the test function v with φ_i^j in equation (10), which can be rewritten as follows:

$$\begin{aligned}&\frac{\partial C_i^j(t)}{\partial t}\int_{\Omega_i}(\varphi_i^j)^2\,\mathrm{d}V - \int_{\Omega_i}\mathbf{F}\left(\mathbf{W}_h(t,\mathbf{x})\right)\cdot\nabla\varphi_i^j\,\mathrm{d}V\\ &+\sum_l\int_{\Omega_l\cap\Omega_i}\varphi_i^j\hat{\mathbf{F}}\left(\left(\sum_{j=0}^{k}C_i^j(t)\varphi_i^j\right),\left(\sum_{j=0}^{k}C_l^j(t)\varphi_l^j\right),\mathbf{n}\right)\mathrm{d}S = \int_{\Omega_i}\mathbf{f}\varphi_i^j\,\mathrm{d}V.\end{aligned} \tag{14}$$

2.3. *Loss functions of deep discontinuous Galerkin method*

We use neural networks to approximate the coefficient $C_i^j(t)$ in equation (13). More precisely, we use DNN to rewrite equation (13) as follows (Chen *et al.* 2021):

$$\mathbf{W}_{h,\theta}(t,\mathbf{x}) = \sum_{j=0}^{k}\sum_{i=1}^{M}\mathcal{N}_\theta^j\left(t,\mathbf{x}_{i+1/2}\right)\varphi_i^j(\mathbf{x}), \tag{15}$$

where $\mathbf{x}_{i+1/2}$ represents the center of the element Ω_i. Here we use a total of $k+1$ neural networks $\mathcal{N}_\theta^0, \mathcal{N}_\theta^1, \ldots, \mathcal{N}_\theta^k$. The inputs to each neural network are temporal variable t and spatial variable $\mathbf{x}$. Our goal is to make the output of the neural network satisfy $\mathcal{N}_\theta^j\left(t,\mathbf{x}_{i+1/2}\right) = C_i^j(t)$.

From equation (15), we can rewrite equation (14) as

$$\begin{aligned}&\frac{\partial\mathcal{N}_\theta^j\left(t,\mathbf{x}_{i+1/2}\right)}{\partial t}\int_{\Omega_i}(\varphi_i^j)^2\,\mathrm{d}V - \int_{\Omega_i}\mathbf{F}\left(\mathbf{W}_{h,\theta}(t,\mathbf{x})\right)\cdot\nabla\varphi_i^j\,\mathrm{d}V\\ &+\sum_l\int_{\Omega_l\cap\Omega_i}\varphi_i^j\hat{\mathbf{F}}\left(\left(\sum_{j=0}^{k}\mathcal{N}_\theta^j(t,\mathbf{x}_{i+1/2})\varphi_i^j\right),\left(\sum_{j=0}^{k}\mathcal{N}_\theta^j(t,\mathbf{x}_{l+1/2})\varphi_l^j\right),\mathbf{n}\right)\mathrm{d}S\\ &=\int_{\Omega_i}\mathbf{f}\varphi_i^j\,\mathrm{d}V.\end{aligned} \tag{16}$$

We use $\mathcal{N}_\theta$ to represent all of the neural networks' outputs $\mathcal{N}_\theta^0, \mathcal{N}_\theta^1, \ldots, \mathcal{N}_\theta^k$. Moreover, we let L denote the operator related to the spatial discretization and $\bar{\mathbf{F}}$ represent the term related to the source f; then equation (16) can be written in the following form:

$$\frac{\partial\mathcal{N}_\theta^j\left(t,\mathbf{x}_{i+1/2}\right)}{\partial t} = L\left(\mathcal{N}_\theta(t,\mathbf{x})\right) + \bar{\mathbf{F}}. \tag{17}$$

For the fully discrete numerical analysis, we employ the 2-order Runge–Kutta-type (RK) total-variation-diminishing (TVD) temporal discretizations (Shu 1988). The 2nd-order TVD RK temporal discretization reads as

$$\begin{aligned}C_{i,j}^{(0)} &= \mathcal{N}_\theta^j\left(t_n,\mathbf{x}_{i+1/2}\right)\\ C_{i,j}^{(1)} &= C_{i,j}^{(0)} + \Delta t L\left(C_{i,j}^{(0)}\right) + \Delta t\bar{\mathbf{F}}(t_n)\end{aligned}$$

$$C_{i,j}^{(2)} = \frac{1}{2}C_{i,j}^{(0)} + \frac{1}{2}C_{i,j}^{(1)} + \frac{1}{2}\Delta t L\left(C_{i,j}^{(1)}\right) + \frac{1}{2}\Delta t\bar{\mathbf{F}}(t_n + \Delta t)$$
$$C_{i,j}^{(n+1)} = C_{i,j}^{(2)}, \tag{18}$$

where t_n denotes the time $n\Delta t$ and Δt is the time step. According to the traditional RKDG method, $C_{i,j}^{(n+1)}$ is the coefficient of the time $(n+1)\Delta t$; hence, in our method, we should have $C_{i,j}^{(n+1)} = \mathcal{N}_\theta^j(t_{n+1}, \mathbf{x}_{i+1/2})$. To describe the difference between $C_{i,j}^{(n+1)}$ and $\mathcal{N}_\theta^j(t_{n+1}, \mathbf{x}_{i+1/2})$, we define

$$L_{i,j,n}^{DG} = C_{i,j}^{(n+1)} - \mathcal{N}_\theta^j\left(t_{n+1}, \mathbf{x}_{i+1/2}\right). \tag{19}$$

Therefore, the loss functions for the DNNs are defined as the residual error of equation (19) in the L^2 sense:

$$\mathcal{L}_j^{DG}(\theta_j) = \left(\sum_{i,n}(L_{i,j,n}^{DG})^2\right)^{1/2} + \lambda\|\theta_j\|_2, \quad j = 0, \ldots, k, \tag{20}$$

where the second term is the regularization, which is designed to limit the capacity of neural networks and prevent the overfitting of our method (Goodfellow *et al.* 2016).

2.4. *Initial and boundary condition*

We take equation (2) as an example to introduce the treatment of initial and boundary conditions. We can handle the general initial and boundary conditions in a completely similar way.

Following the work of Raissi *et al.* (2019), we straightforwardly add two penalty terms which are the initial residual $\mathcal{L}_{u_i}(\theta)$ and the boundary residual $\mathcal{L}_{u_b}(\theta)$. The residuals can be defined as follows:

$$\mathcal{L}_{u_i}(\theta) = \|u_\theta(0,x) - g(x)\|_2 = \left(\int_\Omega (u_\theta(0,x) - g(x))^2 \mathrm{d}x\right)^{1/2}$$
$$\mathcal{L}_{u_b}(\theta) = \|u_\theta(t,x) - h(t,x)\|_2$$
$$= \left(\int_{[0,T]\times\partial\Omega} (u_\theta(t,x) - h(t,x))^2 \mathrm{d}t\mathrm{d}x\right)^{1/2}. \tag{21}$$

Thus our loss function of DeepDG can be rewritten as

$$\mathcal{L}(\theta) = \sum_{j=0}^{k} \mathcal{L}_j^{DG}(\theta_j) + \lambda_i\mathcal{L}_{u_i}(\theta) + \lambda_b\mathcal{L}_{u_b}(\theta). \tag{22}$$

Then we are left with the optimization problem:

$$\min_\theta \mathcal{L}(\theta), \quad \mathcal{L}(\theta) = \sum_{j=0}^{k} \mathcal{L}_j^{DG}(\theta_j) + \lambda_i\mathcal{L}_{u_i}(\theta) + \lambda_b\mathcal{L}_{u_b}(\theta), \tag{23}$$

where θ denotes $(\theta_0, \ldots, \theta_k)$.

2.5. *The stochastic gradient descent with momentum algorithm*

To finish describing the DeepDG, we need to furnish how to solve the optimization problems equation (23) efficiently and effectively.

Notice that the loss function $\mathcal{L}_j^{DG}(\theta_j)$ of each of our neural networks is basically irrelevant, hence when we take the partial derivative θ_j of equation (22) we obtain:

$$\nabla_{\theta_j}\mathcal{L}(\theta) = \nabla_{\theta_j}\mathcal{L}_j^{DG}(\theta_j) + \lambda_i\nabla_{\theta_j}\mathcal{L}_{u_i}(\theta) + \lambda_b\nabla_{\theta_j}\mathcal{L}_{u_b}(\theta), \quad j = 0, 1, \ldots, k. \tag{24}$$

Therefore, we can train the neural networks corresponding to different orders of basic functions relatively independently. Such parallel computation makes it possible to increase the Legendre polynomial order k without affecting our calculation speed.

From the mathematical derivation, we can also find that, in the DG method, whether using the n-order orthogonal polynomial approximation or the $(n+1)$-order orthogonal polynomial approximation, the coefficients of the first n-order are the same. This means that the coefficients of orthogonal Legendre polynomials of different orders in the DG format are independent, and our method can approximate these coefficients independently and simultaneously.

Equation (20) is computationally expensive because i and n are typically very large. i represents the element Ω_i, which can be very large in higher dimensions. For this problem, the algorithm of choice is the stochastic gradient descent with a momentum method, which can be described as follows:

$$\begin{aligned}
\nabla_{\theta_j}\mathcal{L}^{I_m}(\theta) &= \nabla_{\theta_j}\left(\sum_{(i,n)\in I_k}(L^{DG}_{i,j,n})^2\right)^{1/2} + \lambda\frac{\theta_j}{\|\theta_j\|_2} + \lambda_i\nabla_{\theta_j}\mathcal{L}_{u_i}(\theta) + \lambda_b\nabla_{\theta_j}\mathcal{L}_{u_b}(\theta)\\
\Delta_j^{m+1} &= \beta\Delta_j^m + \nabla_{\theta_j}\mathcal{L}^{I_m}(\theta)\\
\theta_j^{m+1} &= \theta_j^m - \eta\Delta_j^{m+1}, \qquad j = 0, 1, \ldots, k,
\end{aligned} \tag{25}$$

where for each iterative step m, I_m is a set of points in $\Omega\times[0,T]$ that are randomly sampled with uniform distribution and β and η are both hyperparameters. In practice, we choose a "mini-batch" of terms at each step, which means $|I_m|$ = mini-batch ($|I_m|$ = the number of elements in the set I_m). For example, a mini-batch can be selected as 10^4 or 10^5. To accelerate the training of the neural network, we use the Adam optimizer version of the SGD with momentum (Kingma and Ba 2014).

2.6. *Deep Lax–Wendroff correction method*

The traditional format for the LWC methods of equation (3) is

$$\begin{aligned}
u_i^{n+1} &= 2u_i^n - u_i^{n-1} + (\Delta t)^2\left(\frac{\partial^2 u}{\partial t^2}\right)_i^n + \frac{(\Delta t)^4}{12}\left(\frac{\partial^4 u}{\partial t^4}\right)_i^n\\
&= 2u_i^n - u_i^{n-1} + (\Delta t)^2\left(c^2\frac{\partial^2 u}{\partial x^2} + f(t,x)\right)_i^n + \frac{(\Delta t)^4}{12}\left(c^4\frac{\partial^4 u}{\partial x^4} + c^2 f_{xx} + f_{tt}\right)_i^n.
\end{aligned} \tag{26}$$

The calculation for high-order derivative terms is

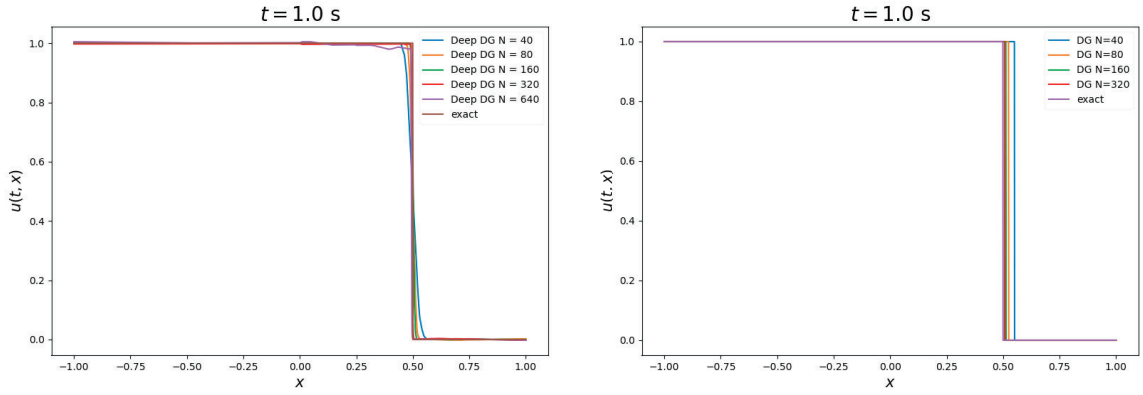
$$\left(\frac{\partial^2 u}{\partial x^2}\right)_j^n = \frac{1}{\Delta x^2}\left[w_0 u_j^n + \sum_{i=1}^{M} w_i\left(u_{j+i}^n + u_{j-i}^n\right)\right], \tag{27}$$

where $M = 2$, 3, and 4 denote the methods with accuracies of $O(\Delta t^4, \Delta x^4)$, $O(\Delta t^4, \Delta x^6)$, and $O(\Delta t^4, \Delta x^8)$.

Similar to the approach of DeepDG, we aim to make the output $\mathcal{N}_\theta(t_n, x_i)$ of the neural network approximate the output u_i^n of traditional methods. Hence, the loss function can be constructed as

$$\begin{aligned}
L_{i,n}^{LWC} &= \mathcal{N}_\theta(t_{n+1}, x_i) - 2\mathcal{N}_\theta(t_n, x_i) + \mathcal{N}_\theta(t_{n-1}, x_i)\\
&\quad -(\Delta t)^2\left(c^2\frac{\partial^2\mathcal{N}_\theta}{\partial x^2} + f(t,x)\right)_i^n - \frac{(\Delta t)^4}{12}\left(c^4\frac{\partial^4\mathcal{N}_\theta}{\partial x^4} + c^2 f_{xx} + f_{tt}\right)_i^n.
\end{aligned} \tag{28}$$

The remaining procedures, such as handling boundary conditions and solving the optimization process, are the same as DeepDG.



(a) Deep discontinuous Galerkin method (b) Discontinuous Galerkin method

Figure 2. Numerical solutions for the Burgers' equation, using two different methods at $t = 1$ s. (a) DeepDG method; (b) DG method.

3. Numerical results

3.1. *Burgers' equation*

Consider the Burgers' equation

$$u_t + \left(\frac{u^2}{2}\right)_x = 0, \tag{29}$$

with initial condition

$$u(0, x) = \begin{cases} 1, & x < 0, \\ 0, & x > 0. \end{cases} \tag{30}$$

Burgers' equation is not an acoustic equation, but we can apply our method without difficulty since it is a hyperbolic conservation law equation.

We select the Burgers' equation with discontinuity. On the one hand, it can verify the effectiveness of the DG method for discontinuous problems. On the other hand, it can also verify whether the combination of DG and deep learning can still have a good effect. Equation (29) has a discontinuous exact solution:

$$u(t, x) = \begin{cases} 1, & x < t/2, \\ 0, & x > t/2. \end{cases} \tag{31}$$

The network we used to solve this problem is same as in Fig. 1, which has two blocks and the size $m = 20$. The model has a total of 1341 parameters.

In order to compute the wave field at time $t = 1$ s, we divide the interval $[-1, 1]$ into N parts and choose time step $\Delta t = 1/N$ s for convenience. We take the Godunov numerical flux and the orthogonal Legendre basis functions. The results from the DeepDG method are shown in Fig. 2.

To compare the performance of the DeepDG and DG methods more effectively in terms of relative error and running time, we present the error curve during the training process of the DeepDG in Fig. 3. Each solid line corresponds to the relative error trend of the DeepDG with different grid numbers N over time. In contrast, since the traditional DG method only yields results at the final time layer $t = 1$ s, its results are displayed as solid points in Fig. 3. The figure shows that the DeepDG method rapidly reduces the relative error during the initial training phase, enabling it to achieve the desired accuracy quickly. Interestingly, for small grid numbers N (e.g., $N = 40,\ 80,\ 160$), the DeepDG method takes more time than the traditional DG method. However, at convergence, the relative error of the DeepDG is significantly lower than that of the traditional DG method. Conversely, for large N (i.e., $N = 320,\ 640$), the DeepDG method achieves the same level of relative error as the traditional DG method in significantly less time.

The specific comparisons of time cost, relative error, and wavefield storage space are shown in Tables 1 and 2. Our results demonstrate that the DeepDG method provides a more accurate solution than the traditional DG method. As a naturally

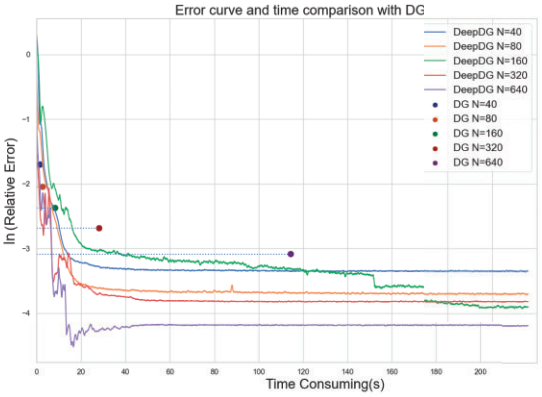


Figure 3. Numerical errors for solving the Burgers' equation; relative error during the training process. Each of the solid lines is the trend of the relative error of the DeepDG with different grid numbers N over time. Each of the solid points is the relative error and the time cost of DG with different grid numbers N over time.

Table 1. Burgers' equation: Comparison of the time cost, relative error, and wavefield storage space between the DG with $N = 320$ and the DeepDG with $N = 40,\ 80,\ 160,\ 320,\ 640$. The final relative L_2 error is the averaged L_2 relative error in the last 1000 steps.

DG			
Element No. (N)	Time cost (s)	Relative L_2 error (%)	Storage space (kb)
320	28.2	6.8	1280
DeepDG			
Element No. (N)	Time cost when the relative error reaches 6.8% (s)	Final relative L_2 error (%)	Storage space (kb)
40	13.3	3.5	14
80	12.3	2.5	14
160	19.4	2.0	14
320	8.7	2.1	14
640	8.6	1.2	14

Table 2. Burgers' equation: Comparison of the time cost, relative error, and wavefield storage space between the DG with $N = 640$ and the DeepDG with $N = 40,\ 80,\ 160,\ 320,\ 640$. The final relative L_2 error is the averaged L_2 relative error in the last 1000 steps.

DG			
Element No. (N)	Time cost (s)	Relative L_2 error (%)	Storage space (kb)
640	114.5	4.6	3840
DeepDG			
Element No. (N)	Time cost when the relative error reaches 4.6% (s)	Final relative L_2 error (%)	Storage space (kb)
40	17.2	3.5	14
80	15.4	2.5	14
160	40.3	2.0	14
320	11.7	2.1	14
640	10.9	1.2	14

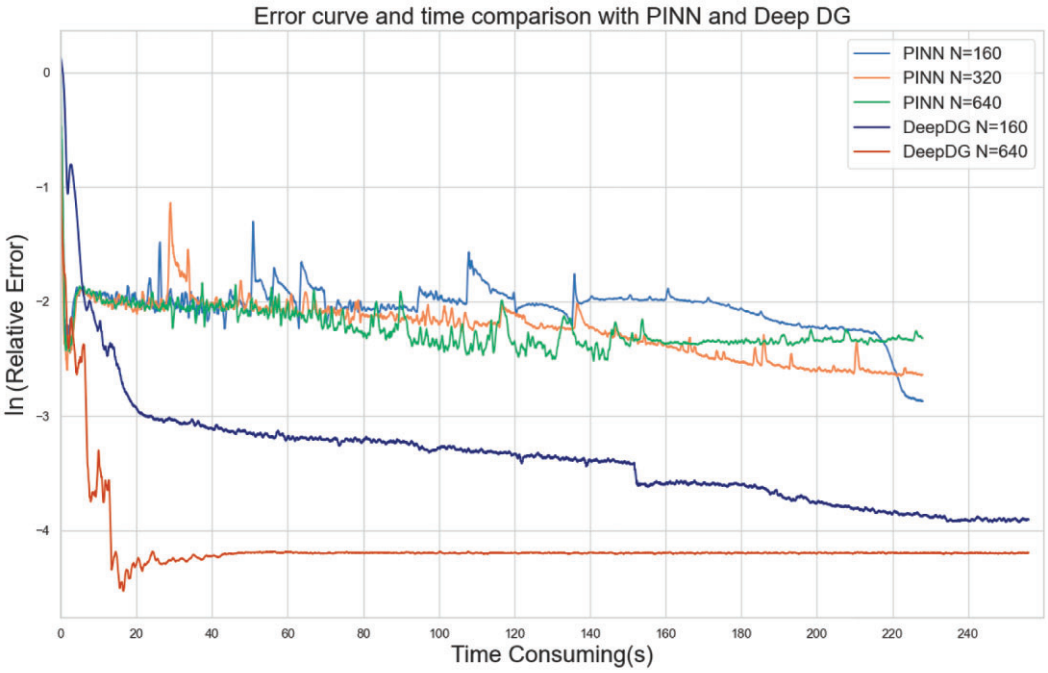


Figure 4. Burgers' equation: Relative error during the training process of PINN and DeepDG.

nonlinear variational method, the DeepDG is also inherently adaptive, contributing to its improved accuracy. Additionally, our analysis reveals that the DeepDG method is much faster than the traditional DG method when achieving the same level of accuracy. For instance, when $N = 320$, the time cost of the traditional DG method is 3.3 times that of the DeepDG method when $N = 640$. Similarly, when $N = 640$, the time cost of the traditional DG method is 10.5 times that of the DeepDG method when $N = 640$. Finally, we also examine the storage space required for each method in Tables 1 and 2. Notably, while the traditional DG method requires the storage of wavefields at all times, the DeepDG method only stores the structure and parameters of the neural network. As a result, the storage space of the DeepDG method is independent of the number of elements (N) and time steps (N_t) and is significantly smaller than that of the traditional DG method. Specifically, for $N = 640$, the storage space of the traditional DG method is 3840 kb, while that of the DeepDG method is only 14 kb. This means that the storage of the DeepDG is 14/3840 of that of the DG method for this case.

Comparison with physics-informed neural networks. To facilitate a more convenient comparison between our method and the recently popular physics-informed neural network (PINN) approach (Raissi *et al.* 2019), we conducted numerical experiments using identical network architectures for both methods. In Fig. 4, we illustrate the relative error curves for both PINN and DeepDG. It is evident that after an initial rapid reduction in relative error, PINN exhibits relatively poor convergence, with the relative error remaining unchanged as N increases, indicating that increasing N does not enhance accuracy. In contrast, our approach demonstrates a lower convergence error compared to PINN, with a more favorable convergence pattern. Moreover, our method can achieve greater accuracy by increasing N, implying that in practical applications the desired level of precision can be attained simply by adjusting N.

In fact, we can theoretically elucidate the distinctions between DeepNM and PINN. Our approach leverages traditional numerical discretization techniques to compute these partial derivatives, incurring no additional computational burden with increasing network complexity. Conversely, PINN utilizes established automatic differentiation techniques from deep learning to compute partial derivatives, making it easier to implement in programming but significantly increasing memory and computational requirements. Detailed information can be found in Appendix A.

3.2. *1D acoustic equation*

In this section, we consider the 1D acoustic equation as follows:

$$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = 0, \tag{32}$$

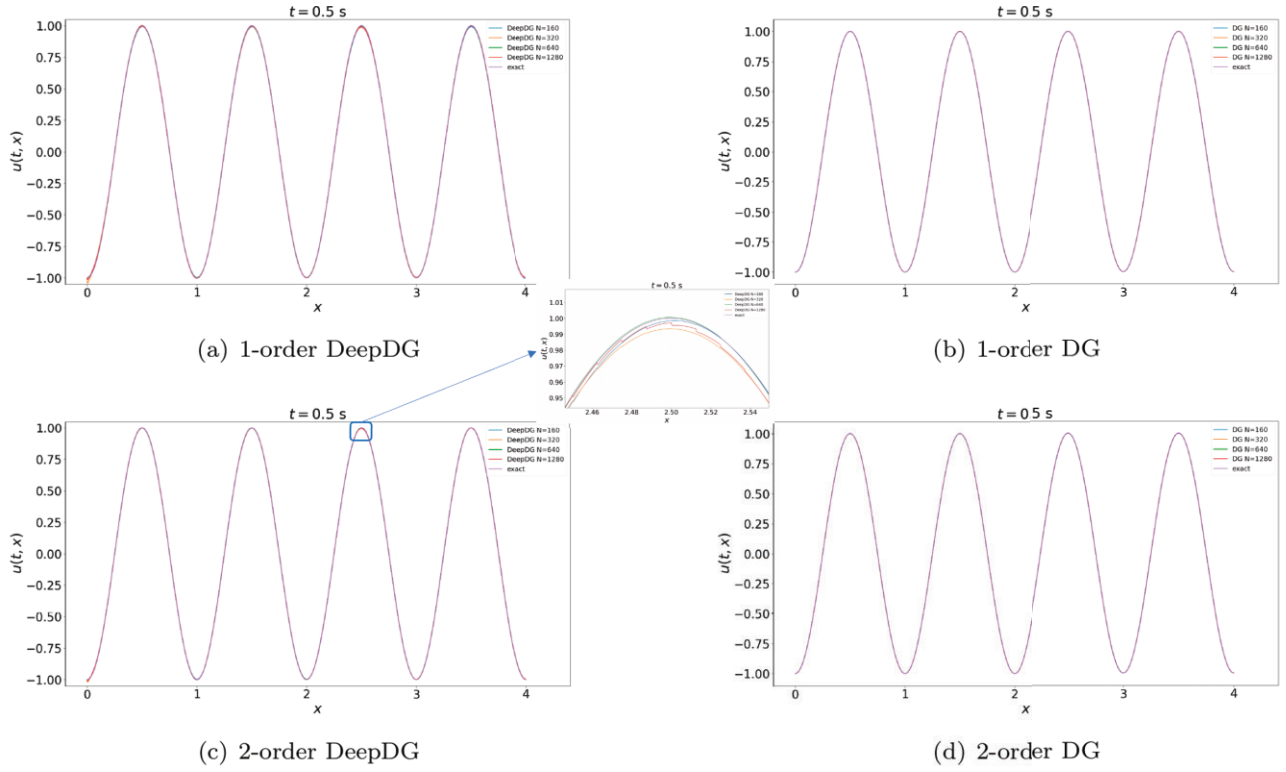


Figure 5. Numerical solutions for the 1D acoustic equation: The waveforms of the acoustic wavefields at time $t = 0.5$ s generated by DeepDG and DG in different orders. Panels (a) and (c) are 1- and 2-order DeepDG, respectively; (b) and (d) are 1- and 2-order DG, respectively.

with the initial conditions

$$\begin{cases} u(x,0) = \cos\left(-\frac{2\pi f_0}{c}x\right), \\ \frac{\partial u(x,0)}{\partial t} = -2\pi f_0 \sin\left(-\frac{2\pi f_0}{c}x\right), \end{cases} \tag{33}$$

where c is the acoustic velocity and f_0 is the frequency. The equation has the following analytical solution:

$$u(t,x) = \cos 1\left(2\pi f_0 t - \frac{2\pi f_0}{c}x\right). \tag{34}$$

We set our computational domain $(t, x) \in [0, 0.5\ \text{s}] \times [0, 4\ \text{km}]$. The network we used in this problem is a stack of four blocks (nine fully connected layers) and the width of the network $m = 20$. The total number of the parameters is 2181 in the model. In order to compute the wave field at time $t = 0.5$ s, we divide the interval $[0, 4\ \text{km}]$ into N elements of the same size and choose time step $\Delta t = 1/(2N)$ s. We use the local Lax–Friedrichs flux and the k-order orthogonal Legendre basis functions $(k = 1, 2)$ in this experiment. Figure 5 shows the numerical solutions compute by DeepDG and DG at $t = 0.5$ s. The left-hand column shows the results of DeepDG of different orders ($k = 1, 2$, respectively) and the right-hand column shows the results of traditional DG. It can be seen that the accuracy of the DeepDG method can reach the expected accuracy of solving seismic wave equations.

In order to compare the relative error and running time of the DeepDG and DG methods more intuitively, we give the error curve during the training process of the DeepDG in Figs 6 and 7. We can see that the DeepDG method declines rapidly at the beginning of the training, which allows the method to achieve the desired accuracy in the application quickly. It can be seen from Fig. 6 that when N is small $(N = 160,\ 320,\ 640)$, the relative error at convergence of DeepDG is significantly lower than the traditional DG. We can also see that when N is large $(N = 320,\ 640,\ 1280)$, the time taken by the DeepDG method to achieve the relative error of the traditional DG method is much shorter than that of the traditional method.

Nevertheless, we also see that for the traditional DG of the fine grid$(N = 1280)$, the accuracy of DeepDG with $N = 160,\ 320,\ 640$ is not as high as the traditional method. Only DeepDG with $N = 1280$ can achieve the accuracy of the fine grid, and the time is only about 1/3 of the fine grid DG. However, to improve the accuracy of our method, we use the DeepDG with

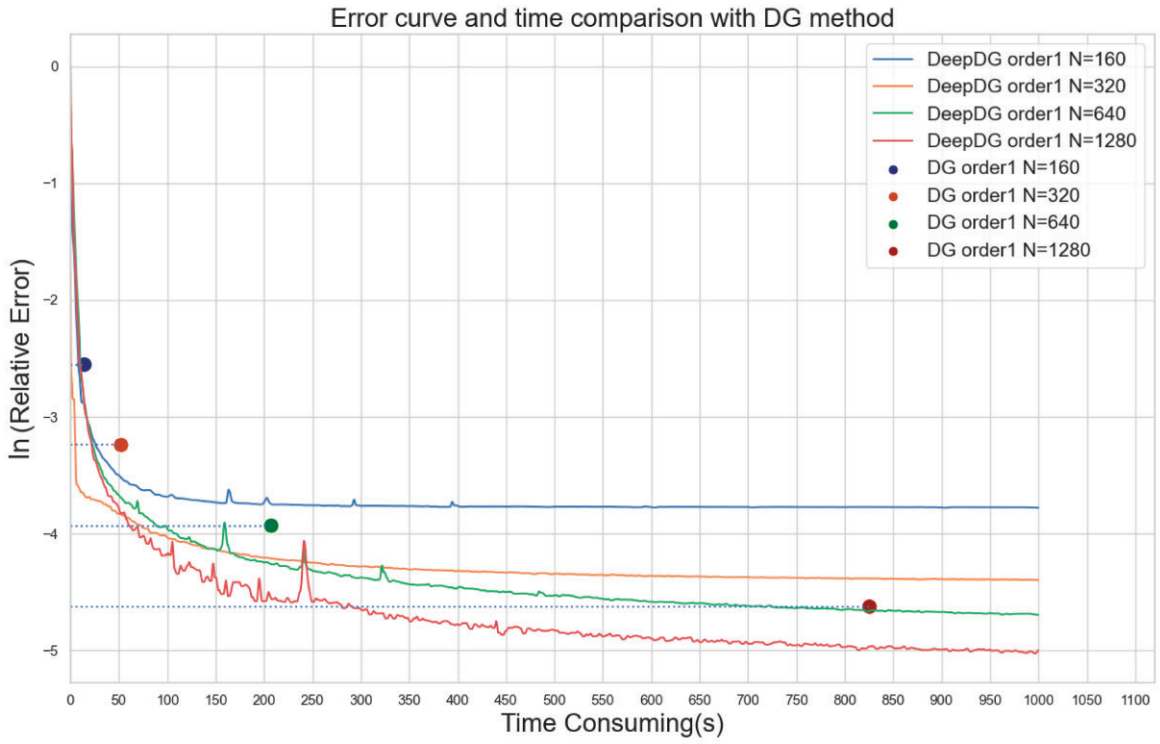


Figure 6. Numerical errors for solving the 1D acoustic equation: Relative error during the training process. Each of the solid lines is the trend of the relative error of the 1-order DeepDG with different grid numbers N over time. Each of the solid points is the relative error and the time cost of DG with different grid numbers N over time.

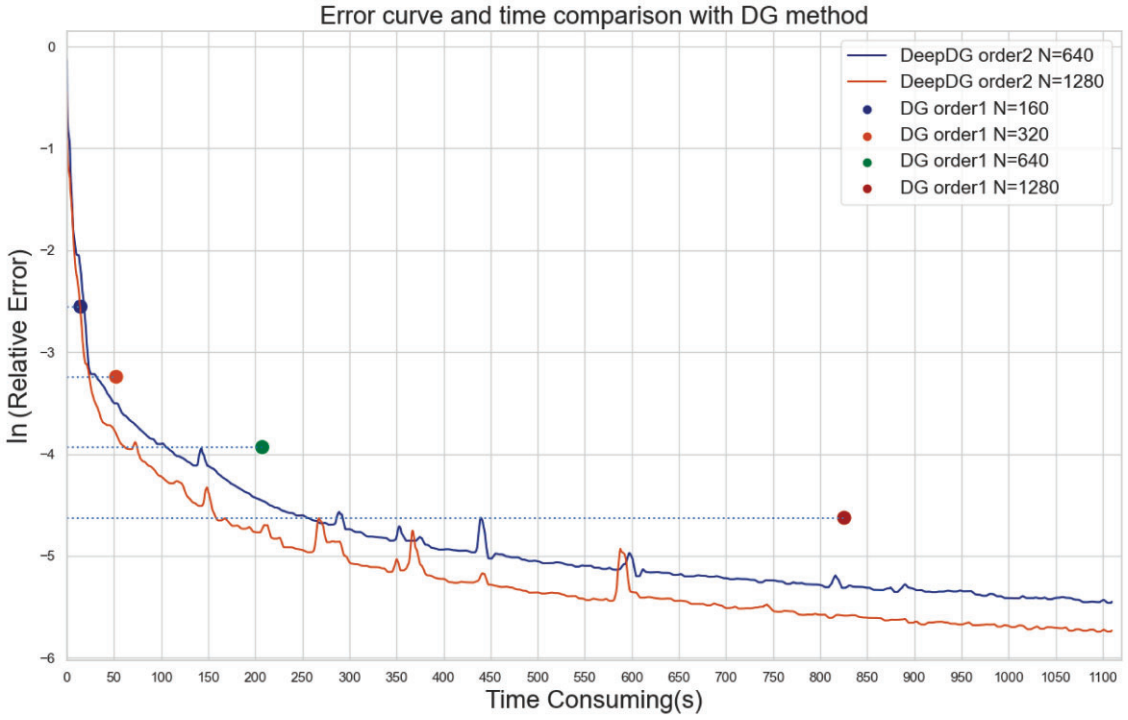


Figure 7. Numerical errors for solving the 1D acoustic equation: Relative error during the training process. Each of the solid lines is the trend of the relative error of the 2-order DeepDG with different grid numbers N over time. Each of the solid points is the relative error and the time cost of DG with different grid numbers N over time.

Table 3. 1D acoustic equation: Comparison of the time cost, relative error, and wavefield storage space between the DG($k = 1$) with $N = 640$ and the DeepDG($k = 1$) with $N = 320,\ 640,\ 1280$ and DeepDG($k = 2$) with $N = 640,\ 1280$. k is the order of DG format. The final relative L_2 error is the averaged L_2 relative error in the last 1000 steps.

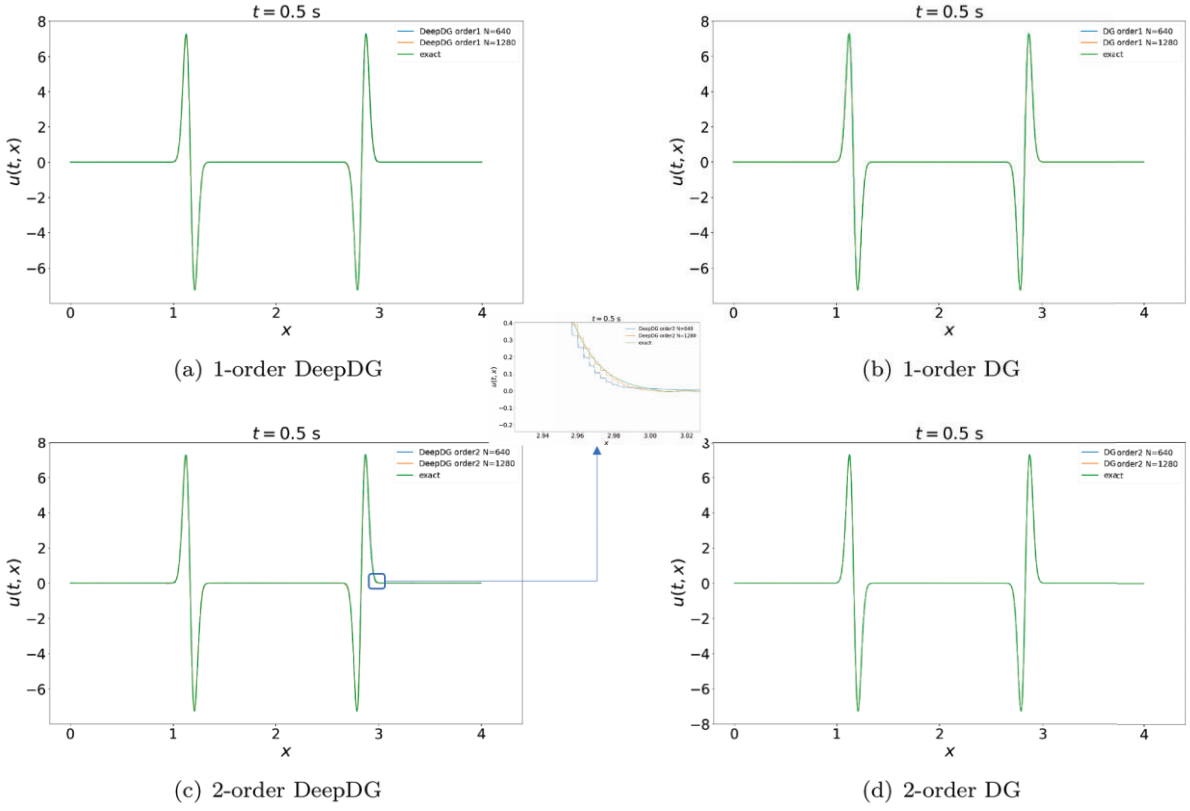
DG(k=1)			
Element No. (N)	Time cost (s)	Relative L_2 error (%)	Storage space (kb)
640	207.4	1.96	10240
DeepDG(k=1)			
Element No. (N)	Time cost when the relative error reaches 1.96% (s)	Final relative L_2 error (%)	Storage space (kb)
320	74.4	1.2	42
640	91.4	0.91	42
1280	63.4	0.65	42
DeepDG(k=2)			
Element No. (N)	Time cost when the relative error reaches 1.96% (s)	Final relative L_2 error (%)	Storage space (kb)
640	109.5	0.42	63
1280	65.8	0.32	63

Table 4. 1D acoustic equation: Comparison of the time cost, relative error, and wavefield storage space between the DG($k = 1$) with $N = 1280$ and the DeepDG($k = 2$) with $N = 640,\ 1280$. k is the order of DG format. The final relative L_2 error is the averaged L_2 relative error in the last 1000 steps.

DG(k=1)			
Element No. (N)	Time cost (s)	Relative L_2 error (%)	Storage space (kb)
1280	825.5	0.98	39680
DeepDG(k=2)			
Element No. (N)	Time cost when the relative error reaches 0.98% (s)	Final relative L_2 error (%)	Storage space (kb)
640	259.4	0.42	63
1280	163.5	0.32	63

the second-order DG format in Fig. 7 to compare it with the traditional DG. We can achieve the accuracy of the traditional method with a fine grid by increasing the order of the numerical format of the deep learning method. At the same time, our method can be about five times faster than the traditional method with the same precision.

Table 3 shows the computational time cost, relative error, and storage space comparison of DeepDG and DG. We can see that the DeepDG gives a more accurate solution than the traditional DG. Being a naturally nonlinear variational method, the DeepDG is also naturally adaptive. We believe that this contributes to the better accuracy of the DeepDG. From Tables 3 and 4, we can also see that the computational speed of DeepDG is much faster than that of DG when the accuracy is the same. For example, the time cost of DG ($k = 1$) with $N = 640$ is 3.3 times that of the DeepDG ($k = 1$) with $N = 1280$, and DG ($k = 1$) with $N = 1280$ takes 5.0 times that of the DeepDG ($k = 2$) with $N = 1280$. In addition, from Table 3 we notice that DeepDG can employ higher-order DG formats or increase the number of grid points to achieve higher precision. We also show the storage space comparison in Tables 3 and 4. If the traditional DG needs to record the wavefields at all times, it must store the coefficients of all basis functions at all times on all elements. Therefore, the storage space of the traditional DG should be proportional to $N \cdot N_t$, where N represents the element number, and N_t means the time steps. Nevertheless, the DeepDG only stores the structure and parameters of the neural network. Thus the storage space of DeepDG is only dependent on the



(a) 1-order DeepDG (b) 1-order DG

(c) 2-order DeepDG (d) 2-order DG

Figure 8. Numerical solutions for the 1D explosive source acoustic equation. The waveforms of the acoustic wavefields at time $t = 0.5$ s generated by DeepDG and DG in different orders. Panels (a) and (c) are 1- and 2-order DeepDG, respectively; (b) and (d) are 1-, 2-order DG, respectively.

network we specify in advance and has nothing to do with the number of elements (N) or time steps (N_t). Therefore, our method is on a different order of magnitude than the traditional method in terms of storage capacity (for $N = 1280$, DG's storage space is 39680 kb and DeepDG's is 63 kb).

3.3. *1D explosive source acoustic equation*

Next we consider the 1D explosive source acoustic equation as follows:

$$\frac{\partial^2 u}{\partial t^2} - c^2 \frac{\partial^2 u}{\partial x^2} = f, \tag{35}$$

where c is the wave velocity and f is the force source. In this numerical experiment, the computational domain is $0 \leq x \leq 4$ km. The source is an explosive source that is located at the center of the computational domain and the expression is

$$f(t, x) = -9.6 f_0 (0.6 f_0 t - 1) \exp(-8(0.6 f_0 t - 1)^2) \delta(x - x_0), \tag{36}$$

where $f_0 = 15$ Hz is the frequency, $x_0 = 2$ km is the source location.

In order to compute the wave field at time $t = 0.5$ s, we divide the interval $[0, 4\text{ km}]$ into N elements of the same size and choose time step $\Delta t = 1/(2N)$ s. We adopted two representative numerical schemes (DG and LWC) to solve the 1D explosive source acoustic equation. In the following, we present the numerical results separately.

3.3.1 DeepDG. The network we used in this problem is a stack of four blocks (nine fully connected layers) and the width of the network is $m = 40$. The total number of parameters in the model is 8361. We use the local Lax–Friedrichs flux and the k-order orthogonal Legendre basis functions $(k = 1, 2)$ in this experiment. Figure 8 shows the numerical solutions computed by DeepDG and DG at $t = 0.5$ s. The left-hand column shows the results of DeepDG of different orders ($k = 1, 2$, respectively) and the right-hand column shows the results of traditional DG. It can be seen that the accuracy of the DeepDG method can reach the expected accuracy of solving seismic wave equations.

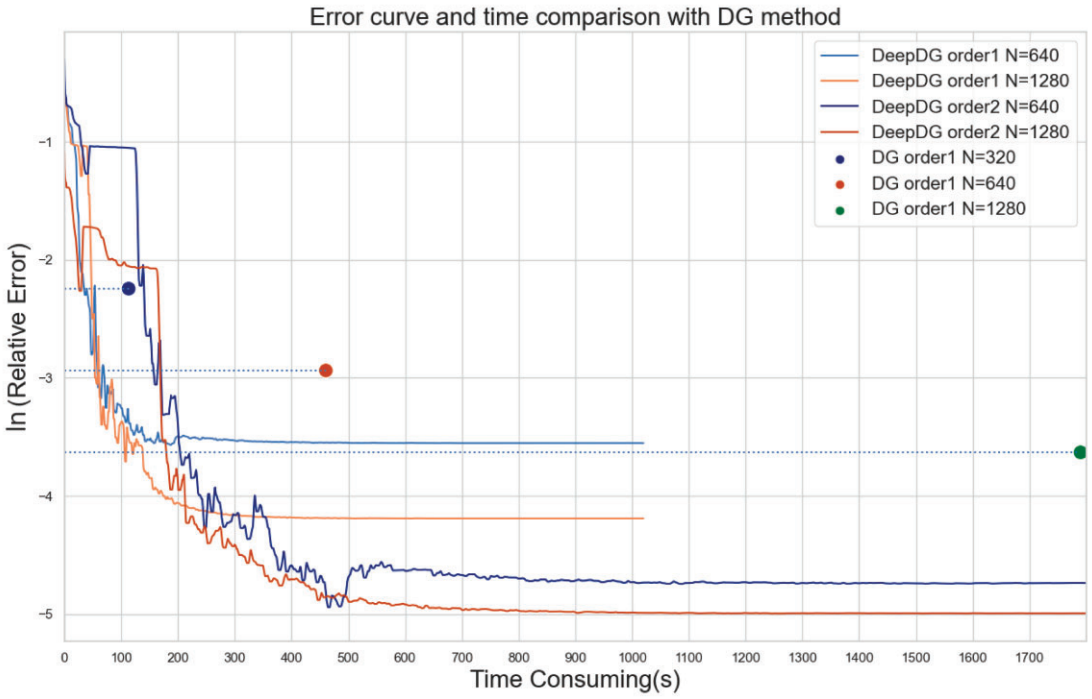


Figure 9. Numerical errors for solving the 1D explosive source acoustic equation: Relative error during the training process. Each of the solid lines is the trend of the relative error of the DeepDG with different grid numbers N over time. Each of the solid points is the relative error and the time cost of DG with different grid numbers N over time.

To provide a more precise comparison of the DeepDG and DG methods in terms of relative error and running time, we present the error curve during the training process of the DeepDG in Fig. 9. This figure shows that our DeepNM achieves a significant speed improvement (3–15 times) compared to the traditional method. The DeepDG method exhibits a rapid decline in relative error at the beginning of the training, allowing for quick attainment of the desired accuracy in practical applications. Furthermore, our experiments revealed that the second-order method exhibits a later stage of rapid relative error decline than the first-order method. However, the relative error at convergence is smaller than that of the first-order method, indicating that we can improve the accuracy of our DeepNM by increasing the order of the numerical format. Nonetheless, this will increase the calculation time to some extent. The specific time cost, relative error, and wavefield storage space comparisons are shown in Tables 5 and 6.

3.3.2 DeepLWC. We uniformly use the second-order LWC format for numerical calculations. The network we used to solve this problem has four blocks, a fully connected layer, and the size $m = 40$. The model has a total of 9961 parameters. The results from the DeepLWC and LWC are shown in Fig. 10.

In order to compare the relative error and running time of the DeepLWC and LWC methods more intuitively, we give the error curve during the training process of the DeepLWC in Fig. 11. Each of the solid lines is the trend of the relative error of the DeepLWC with different grid numbers N over time. Since the traditional LWC method only iterates to the last time layer to give the result at $t = 0.5$ s, the results of the LWC method are displayed in the form of solid points in Fig. 11 (because the traditional method only has the final running time and the corresponding relative error). We can see that the DeepLWC method declines rapidly at the beginning of the training, which allows the method to achieve the desired accuracy in the application quickly. It can be seen from Fig. 11 that the relative error at convergence is significantly lower than the traditional method. In addition, the time taken by the DeepLWC method to achieve the relative error of the traditional LWC method is much shorter (4–30 times) than that of the traditional method. The specific time cost, relative error, and wavefield storage space comparisons are shown in Tables 7 and 8.

3.4. Two-layer model

In this section, we consider the two-layer model shown in Fig. 12. The wave velocities in the upper and lower media are 2 km s^{-1} and 5 km s^{-1}, respectively. The size of the entire domain is $0 \leq x \leq 4$ km and $0 \leq z \leq 4$ km, and the same source as the previous example is used, with a frequency of $f_0 = 30$ Hz located at $z = 1.6$ km. The receiver is located at $z = 1.7$ km.

Table 5. 1D explosive source acoustic equation. Comparison of the time cost, relative error, and wavefield storage space between the DG($k = 1$) with $N = 640$, the DeepDG($k = 1$) with $N = 640,\ 1280$ and the DeepDG($k = 2$) with $N = 640,\ 1280$. k is the order of DG format. The final relative L_2 error is the averaged L_2 relative error in the last 1000 steps.

DG(k=1)			
Element No. (N)	Time cost (s)	Relative L_2 error (%)	Storage space (kb)
640	460.5	5.3	10240
DeepDG(k=1)			
Element No. (N)	Time cost when the relative error reaches 5.3% (s)	Final relative L_2 error (%)	Storage space (kb)
640	64.9	2.8	70
1280	64.9	1.5	70
DeepDG(k=2)			
Element No. (N)	Time cost when the relative error reaches 5.3% (s)	Final relative L_2 error (%)	Storage space (kb)
640	175.8	0.71	105
1280	190.0	0.67	105

Table 6. 1D explosive source acoustic equation. Comparison of the time cost, relative error, and wavefield storage space between the DG($k = 1$) with $N = 1280$, the DeepDG($k = 1$) with $N = 1280$ and the DeepDG($k = 2$) with $N = 640,\ 1280$. k is the order of DG format. The final relative L_2 error is the averaged L_2 relative error in the last 1000 steps.

DG(k=1)			
Element No. (N)	Time cost (s)	Relative L_2 error (%)	Storage space (kb)
1280	1790.2	2.65	39680
DeepDG(k=1)			
Element No. (N)	Time cost when the relative error reaches 2.65% (s)	Final relative L_2 error (%)	Storage space (kb)
1280	120.2	1.5	70
DeepDG(k=2)			
Element No. (N)	Time cost when the relative error reaches 2.65% (s)	Final relative L_2 error (%)	Storage space (kb)
640	225.8	0.71	105
1280	199.2	0.67	105

For this problem, we employ a network architecture comprising five blocks, consisting of 11 fully connected layers. The width of the network was set to $m = 60$. The total number of parameters in the model amounted to 36901. In order to compute the wave field at time $t = 0.6$ s, we divide the interval $[0, 4\ \text{km}]$ into $N_x = N_z = 400$ elements of the same size and choose time step $\Delta t = 0.0004$ s. We use the 4-order LWC method in this experiment. Figures 13 and 14 show the numerical solutions computed by DeepLWC and the LWC at $t = 1.2$ s. The left-hand column is the result of DeepLWC and the middle column is the result of traditional LWC. The misfits show the absolute pointwise error. Figures 13 and 14 show that DeepLWC provides a good solution to the models with strong velocity between the adjacent layers. Figure 14 shows that the

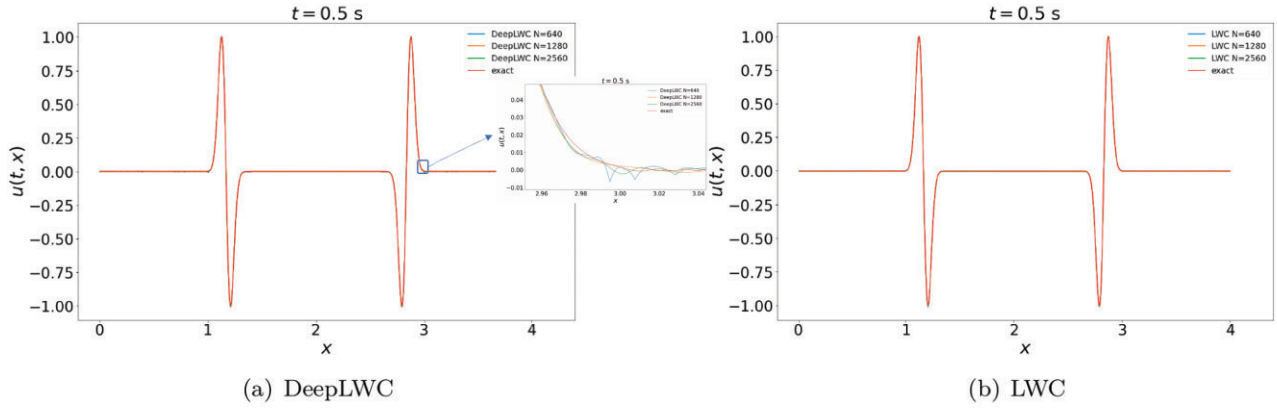


Figure 10. Numerical solutions for the 1D explosive source acoustic equation: The waveforms of the acoustic wavefields at time $t = 0.5$ s generated by (a) DeepLWC and (b) LWC.

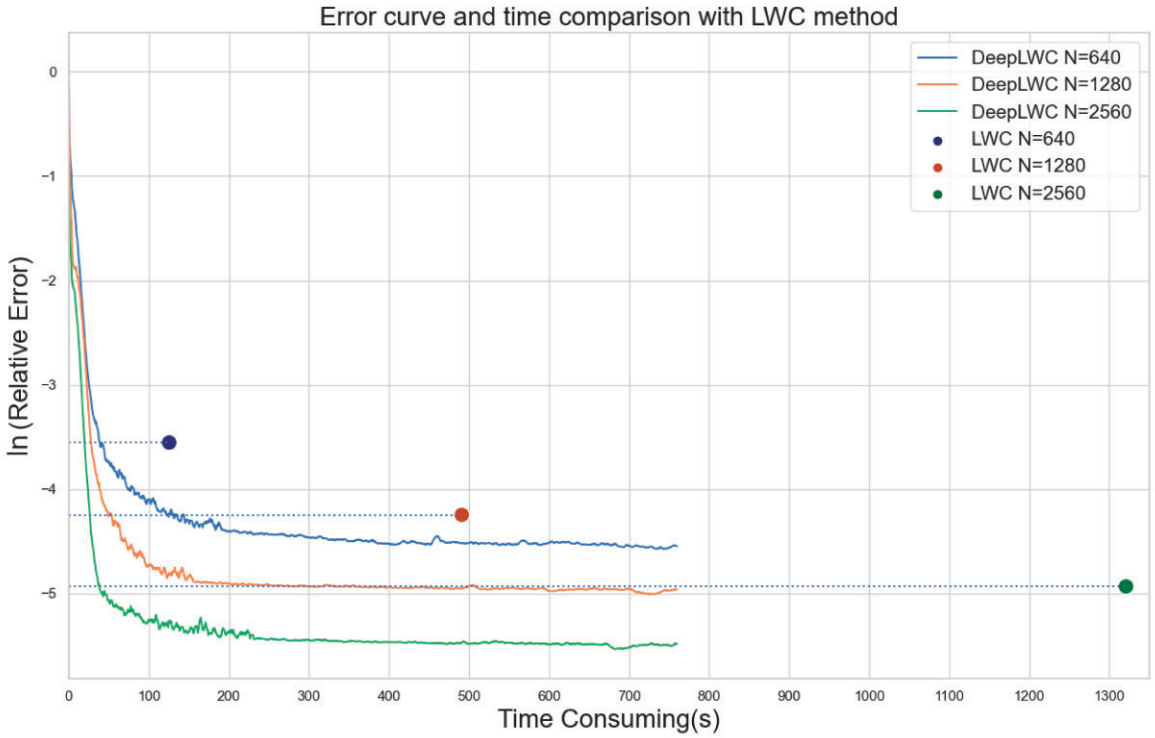


Figure 11. Numerical error for solving the 1D explosive source acoustic equation: Relative error during the training process. Each of the solid lines is the trend of the relative error of the DeepLWC with different grid numbers N over time. Each of the solid points is the relative error and the time cost of LWC with different grid numbers N over time.

DeepLWC gives a more accurate solution than the traditional LWC. As a naturally nonlinear method, DeepLWC is inherently adaptive, and we believe that this characteristic contributes to its improved accuracy.

The time cost of LWC is 3510 s and that of DeepLWC is 492 s. The computational speed of DeepLWC is almost seven times faster than that of LWC. LWC's storage space is 5892 mb and DeepLWC's is 0.12 mb. This means that the storage of the DeepLWC is $0.12/5892 = 2 \times 10^{-5}$ of that of the LWC method for this case. Figure 15 shows the evolution of the history of convergence of loss terms in equation (22). The experiment shows that our method remains effective for high-dimensional complex medium problems and offers significant improvements over traditional methods.

Table 7. 1D explosive source acoustic equation. Comparison of the time cost, relative error, and wavefield storage space between the LWC with $N = 1280$ and the DeepLWC with $N = 640,\ 1280,\ 2560$. The final relative L_2 error is the averaged L_2 relative error in the last 1000 steps.

LWC			
Element No. (N)	Time cost (s)	Relative L_2 error (%)	Storage space (kb)
1280	491	1.43	14080
DeepLWC			
Element No. (N)	Time cost when the relative error reaches 1.43% (s)	Final relative L_2 error (%)	Storage space (kb)
640	120.4	1.0	83
1280	52.1	0.66	83
2560	31.9	0.39	83

Table 8. 1D explosive source acoustic equation. Comparison of the time cost, relative error, and wavefield storage space between the LWC with $N = 2560$ and the DeepLWC with $N = 1280,\ 2560$. The final relative L_2 error is the averaged L_2 relative error in the last 1000 steps.

LWC			
Element No. (N)	Time cost (s)	Relative L_2 error (%)	Storage space (kb)
2560	1321	0.72	53760
DeepLWC			
Element No. (N)	Time cost when the relative error reaches 0.72% (s)	Final relative L_2 error (%)	Storage space (kb)
1280	270.9	0.66	83
2560	42.6	0.39	83

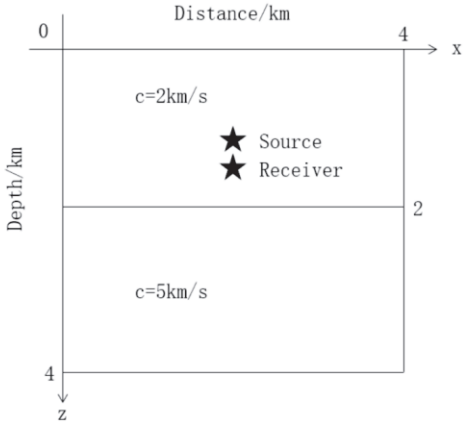


Figure 12. Two-layer model with a strong discontinuity at the depth of $z = 2$ km. The explosive source is located at $z = 1.6$ km, while the receiver is located at $z = 1.7$ km.

4. Discussions and conclusions

In this paper, we introduced a deep learning framework that combines traditional numerical methods to tackle wave equations, particularly hyperbolic conservation law equations. Our findings reveal several distinct advantages of the novel DeepNM approach. It significantly enhances computational speed, surpassing traditional methods by more than tenfold while maintaining equivalent accuracy. Moreover, it drastically reduces storage requirements by only necessitating the storage of the deep learning neural network for reconstructing the entire wave field within the computational domain. DeepNM ex-

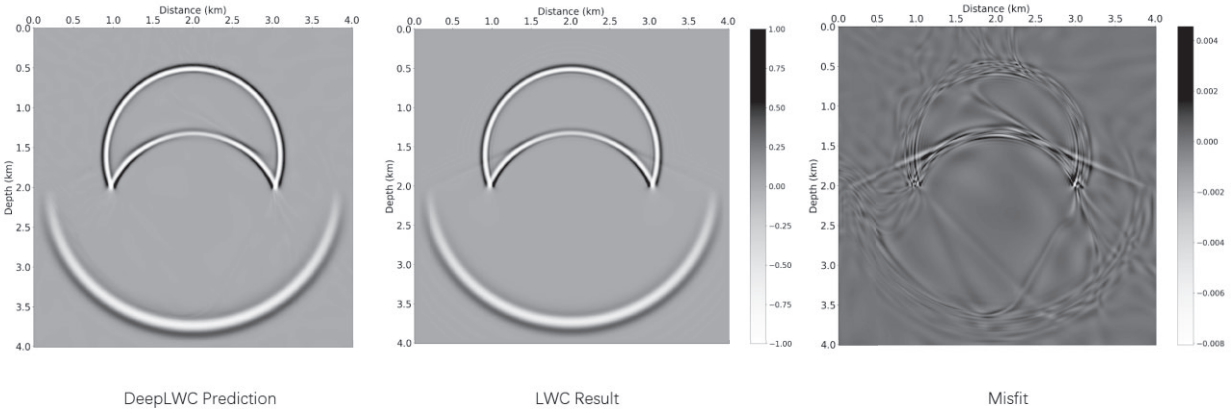


Figure 13. Numerical solutions for the two-layer model: The acoustic wavefields at time $t = 1.2$ s generated by DeepLWC and LWC. The misfits show the absolute pointwise error.

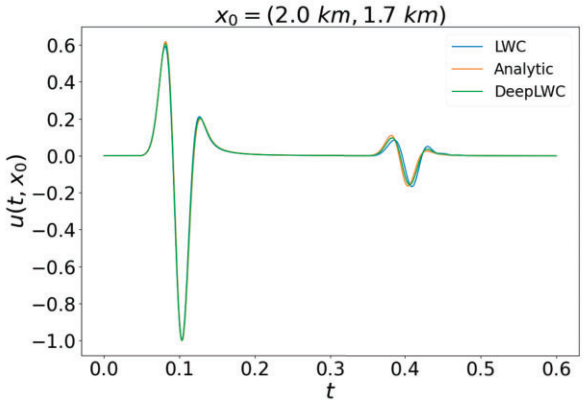


Figure 14. Numerical solutions for the two-layer model: Waveforms for the displacement generated by the DeepLWC, LWC, and the analytic method for the acoustic wave equation in the two-layer model.

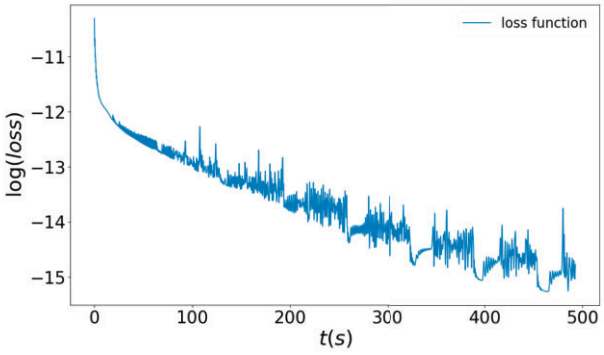


Figure 15. Loss function for solving the two-layer model during the training process.

hibits inherent adaptability, enhancing accuracy even with lower-order schemes compared to traditional numerical methods. Furthermore, it achieves higher precision through the utilization of higher-order numerical formats or the increase of grid points for more demanding problems.

In contrast to PINN, DeepNM maximizes the benefits of conventional mathematical techniques in solving partial differential equations, resulting in marked improvements in accuracy and computational efficiency. Notably, DeepNM introduces a novel research paradigm for numerical equation-solving that can be seamlessly integrated with various traditional numerical methods. This integration brings about accelerated computations and a reduction in storage requirements when compared to conventional approaches.

An important advantage of our approach is that there is no need to pre-prepare a training set for training. Our training process is the process of neural networks approximating the solution to the equations. Our approach is an improvement upon traditional methods rather than the mainstream end-to-end techniques commonly employed in current deep learning-based PDE solvers.

Our method has numerous potential applications. First, it can easily be applied to many other numerical formats, offering new perspectives for traditional numerical methods and improving their computational efficiency, effectively breaking through CFL conditions in a sense. Furthermore, due to our advantages in computational speed and storage, our approach can be employed in large-scale seismic wavefield simulations, wave equation-based seismic migration, and full waveform inversion. As the dimensionality of the problem increases, our method is expected to exhibit even greater improvements in computational efficiency and reduced storage compared to traditional methods.

However, there are notable challenges to address in future work. Currently, the convergence rate remains inconclusive, necessitating further investigation. Additionally, the treatment of initial and boundary conditions is somewhat intricate, and not as straightforward as with traditional methods. Furthermore, for high-dimensional problems, the efficiency of random sampling is too low, indicating a need for developing new sampling methods.

In summary, DeepNM offers significant potential in the realm of wave propagation modeling, paving the way for enhanced accuracy and computational efficiency.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (grant nos. 42330801, and 42374143).

Conflict of interest statement. None declared.

Appendix: Comparison with physics-informed neural networks

PINN is currently a widely used method for solving partial differential equations based on deep learning (Raissi *et al.* 2019, Mao *et al.* 2020, Cai *et al.* 2021, Chen *et al.* 2021, Zhang *et al.* 2022). One of the reasons for its wide application is that its mathematical principles are relatively straightforward, and it is also easy to implement technically. Before comparing our method with PINN, we briefly introduce how to use PINN to solve the 1D acoustic wave equation.

PINN for Burgers' equation

We define $\bar{f}$ to be given by the equation (29):

$$\bar{f} := u_t + \left(\frac{u^2}{2}\right)_x, \qquad x \in \Omega, \quad 0 \le t \le T, \tag{A1}$$

and proceed by approximating $u(t, x)$ by a DNN. This assumption, together with equation (A1), leads to a PINN $\bar{f}(t, x)$. This network can be derived by applying the chain rule for differentiating compositions of functions using an automatic differential (Baydin *et al.* 2018). The shared parameters between the neural networks $u(t, x)$ and $\bar{f}(t, x)$ can be learned by minimizing the mean squared error loss

$$MSE = MSE_u + MSE_{\bar{f}}, \tag{A2}$$

where

$$MSE_u = \frac{1}{N_u}\sum_{i=1}^{N_u}\left|u(t_u^i, x_u^i) - u^i\right|^2, \tag{A3}$$

and

$$MSE_{\bar{f}} = \frac{1}{N_{\bar{f}}}\sum_{i=1}^{N_{\bar{f}}}\left|\bar{f}(t_{\bar{f}}^i, x_{\bar{f}}^i)\right|^2. \tag{A4}$$

Here, $\{t_u^i, x_u^i, u^i\}_{i=1}^{N_u}$ denotes the initial and boundary training data on $u(t, x)$, and $\{t_{\bar{f}}^i, x_{\bar{f}}^i\}_{i=1}^{N_{\bar{f}}}$ specifies the collocations points for $\bar{f}(t, x)$.

PINN compared with DeepNM

For the sake of convenience, we have conducted a comparative analysis between PINN and DeepDG. Upon scrutinizing the optimization problems inherent in both methods, a fundamental contrast emerges in the manner by which they calculate partial derivatives of $\bar{f}$, such as $\frac{\partial u}{\partial t}$ and u_x. Our approach leverages conventional numerical discretization techniques to derive these derivatives, a strategy that maintains computational efficiency regardless of network complexity. In contrast, PINN harnesses the well-established automatic differentiation technology in deep learning for partial derivative calculations. While this method is more straightforward to implement in programming, it substantially escalates both memory and computational demands.

This increased burden is attributable to two primary factors. First, the backpropagation algorithm necessitates the storage of the entire computation graph structure, a memory-intensive task, especially when dealing with intricate network architectures and numerous layers. Secondly, backpropagation relies on numerical calculations rather than symbolic differentiation formulas, demanding the computation of derivatives for all internal points, denoted as $N_{\bar{f}}$, which can be computationally intensive. In contrast, our approach efficiently computes derivative terms.

In summary, DeepNM synergistically combines the advantages of traditional numerical methodologies and deep learning techniques, setting it apart from PINN. Specifically, our method utilizes numerical approaches with low computational overhead for partial derivative computations, in stark contrast to PINN's automatic differentiation approach. As a result, DeepNM significantly reduces computational burdens, exceeding tenfold compared to PINN. Our numerical experiments further confirm that DeepNM outperforms PINN in terms of relative error reduction and convergence speed, particularly in the context of the Burgers' equation.

References

Asem M. DiffusionNet: Accelerating the solution of Time-Dependent partial differential equations using deep learning. arXiv, https://doi.org/10.48550/arXiv.2011.10015, 19 Nov 2020, preprint: not peer-reviewed.

Atkins H, Atkins H. Continued development of the discontinuous Galerkin method for computational aeroacoustic applications. *AIAA 1997-1581. 3rd AIAA/CEAS Aeroacoustics Conference*, May 1997. https://doi.org/10.2514/6.1997-1581

Baydin AG, Pearlmutter BA, Radul AA *et al.* Automatic differentiation in machine learning: a survey. *J Mach Learn Res* 2018;**18**:1–43.

Burstein SZ. Finite-difference calculations for hydrodynamic flows containing discontinuities. *J Comput Phys* 1966;**1**(2):198–222. https://doi.org/10.1016/0021-9991(66)90003-9

Cai S, Wang Z, Wang S *et al.* Physics-informed neural networks for heat transfer problems. *J Heat Transf* 2021;**143**:060801. https://doi.org/10.1115/1.4050542

Chen Z, Liu Y, Sun H. Physics-informed learning of governing equations from scarce data. *Nat Commun* 2021;**12**:6136. https://doi.org/10.1038/s41467-021-26434-1

Chen J, Jin S, Lyu L. A deep learning based discontinuous Galerkin method for hyperbolic equations with discontinuous solutions and random uncertainties. arXiv, https://doi.org/10.48550/arXiv.2107.01127, 2 Jul 2021, preprint: not peer-reviewed. https://doi.org/10.4208/jcm.2205-m2021-0277

Cockburn B, Shu CW. TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws. II. General framework. *Math C omput* 1989;**52**:411–35.

Cockburn B, Shu CW. The Runge-Kutta local projection-discontinuous-Galerkin finite element method for scalar conservation laws. *ESAIM: Math Model Numer Anal* 1991;**25**:337–61. https://doi.org/10.1051/m2an/1991250303371

Cockburn B, Shu CW. The Runge–Kutta discontinuous Galerkin method for conservation laws V: multidimensional systems. *J Comput Phys* 1998;**141**:199–224. https://doi.org/10.1006/jcph.1998.5892

Cockburn B, Lin SY, Shu CW. TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws III: one-dimensional systems. *J Comput Phys* 1989;**84**:90–113. https://doi.org/10.1016/0021-9991(89)90183-6

Cockburn B, Hou S, Shu CW. The Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws. IV. The multidimensional case. *Math Comput* 1990;**54**:545–81.

Courant R, Friedrichs K, Lewy H. Über die partiellen Differenzengleichungen der mathematischen Physik. *Math Ann* 1928;**100**:32–74.

Dablain M. The application of high-order differencing to the scalar wave equation. *Geophysics* 1986;**51**:54–66. https://doi.org/10.1190/1.1442040

Dumbser M, Munz CD. ADER discontinuous Galerkin schemes for aeroacoustics. *CR Mécanique* 2005;**333**:683–7. https://doi.org/10.1016/j.crme.2005.07.008

Engsig-Karup AP, Hesthaven JS, Bingham HB *et al.* DG-FEM solution for nonlinear wave-structure interaction using Boussinesq-type equations. *Coast Eng* 2008;**55**:197–208. https://doi.org/10.1016/j.coastaleng.2007.09.005

Goodfellow I, Bengio Y, Courville A. *Deep Learning*. Cambridge, MA: MIT Press, 2016.

Weinan E, Han J, Jentzen A *et al.* Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Commun Math Stat* 2017;**5**:349–80.

Han J, Jentzen A, Weinan E. Solving high-dimensional partial differential equations using deep learning. *Proc Natl Acad Sci* 2018;**115**:8505–10. https://doi.org/10.1073/pnas.1718942115

He X, Yang D, Wu H. A weighted Runge–Kutta discontinuous Galerkin method for wavefield modelling. *Geophys J Int* 2015;**200**:1389–410. https://doi.org/10.1093/gji/ggu487

He K, Zhang X, Ren S *et al.* Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016. p. 770–8. https://doi.org/10.1109/CVPR.2016.90

Hesthaven JS, Warburton T. Nodal high-order methods on unstructured grids: I. Time-domain solution of Maxwell's equations. *J Comput Phys* 2002;**181**:186–221. https://doi.org/10.1006/jcph.2002.7118

Jameson A. Numerical solution of the Euler equations for compressible inviscid fluids. *Numer Meth Euler Eq Fluid Dyn* 1985;**1**:1–46.

Kingma DP, Ba J. Adam: a method for stochastic optimization. arXiv, https://doi.org/10.48550/arXiv.1412.6980, 22 Dec 2014, preprint: not peer-reviewed.

Lax P. *Systems of conservation laws*. Technical report, Los Alamos National Laboratory, 1959.

LeCun Y, Bengio Y, Hinton G. Deep learning. *Nature* 2015;**521**:436–44. https://doi.org/10.1038/nature14539

LeVeque RJ *et al.* Finite volume methods for hyperbolic problems, Vol. **31**. Cambridge University Press, Cambridge, 2002.https://doi.org/10.1017/CBO9780511791253

Lu T, Zhang P, Cai W. Discontinuous Galerkin methods for dispersive and lossy Maxwell's equations and PML boundary conditions. *J Comput Phys* 2004;**200**:549–80. https://doi.org/10.1016/j.jcp.2004.02.022

Mao Z, Jagtap AD, Karniadakis GE. Physics-informed neural networks for high-speed flows. *Comput Meth Appl Mech Eng* 2020;**360**:112789. https://doi.org/10.1016/j.cma.2019.112789

Meng X, Li Z, Zhang D *et al.* PPINN: Parareal physics-informed neural network for time-dependent PDEs. *Comput Meth Appl Mech Eng* 2020;**370**:113250. https://doi.org/10.1016/j.cma.2020.113250

Montufar GF, Pascanu R, Cho K *et al.* On the number of linear regions of deep neural networks. *Adv Neur Inf Proc Syst* 2014;**27**:1–9.

Pinkus A. Approximation theory of the MLP model in neural networks. *Acta Numer* 1999;**8**:143–95. https://doi.org/10.1017/S0962492900002919

Raissi M, Perdikaris P, Karniadakis GE. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J Comput Phys* 2019;**378**:686–707. https://doi.org/10.1016/j.jcp.2018.10.045

Raissi M, Yazdani A, Karniadakis GE. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science* 2020;**367**:1026–30. https://doi.org/10.1126/science.aaw4741

Reed WH, Hill TR. *Triangular mesh methods for the neutron transport equation*. Technical report, Los Alamos Scientific Laboratory, 1973.

Shu CW. Total-variation-diminishing time discretizations. *SIAM J Sci Stat Comput* 1988;**9**:1073–84. https://doi.org/10.1137/0909073

Silver D, Huang A, Maddison CJ *et al.* Mastering the game of Go with deep neural networks and tree search. *Nature* 2016;**529**:484–9. https://doi.org/10.1038/nature16961

Sirignano J, Spiliopoulos K. DGM: a deep learning algorithm for solving partial differential equations. *J Comput Phys* 2018;**375**:1339–64. https://doi.org/10.1016/j.jcp.2018.08.029

Toro EF, Solvers R. *Numerical methods for fluid dynamics: a practical introduction*. Berlin: Springer, 1999.

Van der Vegt JJ, Van der Ven H. Space–time discontinuous Galerkin finite element method with dynamic grid motion for inviscid compressible flows: I. General formulation. *J Comput Phys* 2002;**182**:546–85. https://doi.org/10.1006/jcph.2002.7185

Zhang E, Dao M, Karniadakis GE, Suresh S. Analyses of internal structures and defects in materials using physics-informed neural networks. *Sci Adv* 2022;**8**:eabk0644. https://doi.org/10.1126/sciadv.abk0644