Journal of Computational Mathematics
Vol.41, No.6, 2023, 1281–1304.

http://www.global-sci.org/jcm
doi:10.4208/jcm.2205-m2021-0277

A DEEP LEARNING BASED DISCONTINUOUS GALERKIN METHOD FOR HYPERBOLIC EQUATIONS WITH DISCONTINUOUS SOLUTIONS AND RANDOM UNCERTAINTIES*

Jingrun Chen[1)]
School of Mathematical Sciences, University of Science and Technology of China, Hefei 230026, China
Suzhou Institute for Advanced Research, University of Science and Technology of China, Suzhou 215123, China
Email: jingrunchen@ustc.edu.cn

Shi Jin
School of Mathematical Sciences, Institute of Natural Sciences, and MOE-LSC, Shanghai Jiao Tong University, Shanghai, 200240, China
Email: shijin-m@sjtu.edu.cn

Liyao Lyu
Department of Computational Mathematics, Science, and Engineering, Michigan State University, East Lansing, MI, 48824, USA
Email: lyuliyao@msu.edu

Abstract

We propose a deep learning based discontinuous Galerkin method (D2GM) to solve hyperbolic equations with discontinuous solutions and random uncertainties. The main computational challenges for such problems include discontinuities of the solutions and the curse of dimensionality due to uncertainties. Deep learning techniques have been favored for high-dimensional problems but face difficulties when the solution is not smooth, thus have so far been mainly used for viscous hyperbolic system that admits only smooth solutions. We alleviate this difficulty by setting up the loss function using discrete shock capturing schemes–the discontinous Galerkin method as an example–since the solutions are smooth in the discrete space. The convergence of D2GM is established via the Lax equivalence theorem kind of argument. The high-dimensional random space is handled by the Monte-Carlo method. Such a setup makes the D2GM approximate high-dimensional functions over the random space with satisfactory accuracy at reasonable cost. The D2GM is found numerically to be first-order and second-order accurate for (stochastic) linear conservation law with smooth solutions using piecewise constant and piecewise linear basis functions, respectively. Numerous examples are given to verify the efficiency and the robustness of D2GM with the dimensionality of random variables up to 200 for (stochastic) linear conservation law and (stochastic) Burgers' equation.

Mathematics subject classification: 65C30, 65N99, 35L65.

Key words: Discontinuous Galerkin method, Loss function, Convergence analysis, Deep learning, Hyperbolic equation.

* Received October 1, 2021 / Revised version received January 26, 2022 / Accepted May 17, 2022 / Published online December 7, 2022 /

1) Corresponding author

1. Introduction

Hyperbolic equations with discontinuous solutions in the physical space arise in problems such as fluid mechanics, combustion, nonlinear acoustics, gas dynamics, and traffic flow [12,26]. One famous example is the compressible Euler equations in gas dynamics, which are the compressible Navier-Stokes equations without viscosity and heat conductivity. The inviscid equations develop discontinuous solutions, aka shocks, even if one starts from smooth initial data. Capturing shock waves has been an important subject in scientific computing and has been very successful [19,26]. Meanwhile, in reality, one may need to consider many sources of uncertainties that can arise in these models. They may be due to the incomplete knowledge of the model, such as the empirical equations of state or constitutive relations, imprecise measurement of physical parameters, and inaccurate measurement of boundary and initial data. Therefore, it is highly desirable to develop computational methods that not only capture the singular profile of solutions in the physical space but also take random uncertainties into account in the random space for high-fidelity simulations, along the line of uncertainty quantification (UQ) [23].

Due to the high dimensionality of the problems under study, it is natural to use deep-learning based approaches, which have been recently proposed for high-dimensional partial differential equations; see [13–15, 27–29, 31, 32, 35, 36] for examples and references therein. In these methods, the basic idea is to use a deep neural network (DNN) as the trail function to approximate the solution based on global optimization of a suitably chosen loss function. Specifically, the parameters in the DNN are optimized to make the DNN approximation satisfy the PDE and boundary/initial conditions as accurately as possible. Quite good approximate solutions are obtained for problems with dimensionality about 100. In all these methods, the loss function involves the (possibly higher-order) derivatives of the PDE solution, which prevents their ability to solve problems with discontinuous solutions, such as the (inviscid) Burgers' equation and the compressible Euler equations, and hence one usually solves viscous problems in which the solutions are smooth [31].

For hyperbolic equations with discontinuous solutions in the physical space, the discontinuous Galerkin (DG) method has been very popular [7–10, 25]. The flexibility of using discontinuous basis functions makes the DG methods capable of solving equations with discontinuous solutions, such as shock waves. There are many other shock capturing schemes [26] that can also be used. Here DG is chosen just as one example. For problems with uncertainties, the stochastic Galerkin (SG) method has been developed for PDEs with random coefficients [2,34], such as stochastic conservation laws [1, 24, 30], stochastic Hamilton–Jacobi equation [20] and stochastic wave equation [18, 33]. Compared with the Monte-Carlo (MC) method, the SG method achieves the spectral accuracy given the sufficient regularity of the PDE solution in the random space. Even though the SG methods are widely used for stochastic problems, their computational complexity grows exponentially with respect to the dimensionality of the random space. Therefore, when the dimensionality of the random space is large, the MC method is preferred.

In this work, we propose a deep learning based discontinuous Galerkin method (D2GM) to solve hyperbolic equations with discontinuous solutions and random uncertainties by combining the advantages of the DG method and DNNs. A key idea here is that at the discrete level, the DG method as an example here, the solution is smooth although its continuous counterpart is not. Thus one can expect that DNN will train better than the ones using AutoGrad in PyTorch or TensorFlow for time and/or spatial derivatives. We will give a convergence analysis for this

DNN solution for the case of 1d upwind flux. The idea of taking advantage of the smoothing effect of the discrete derivatives has been used previously for solving linear wave equations with discontinuous uncertain coefficients [22]. In the high-dimensional random space we use the MC method. The proposed method has the following properties:

- By using the DNN representation in both physical and random spaces, the D2GM can approximate the PDE solution well in high dimensions.
- By using the weak formulation and discontinuous element basis, the D2GM is able to approximate discontinuous PDE solutions with high accuracy.
- By using the mini-batch sampling with controllable number of samples, the D2GM overcomes the curse of dimensionality.

The rest of paper is organized as follows. In Section 2, the D2GM is proposed with details about the DNN, discontinuous element basis, loss function, boundary and initial conditions, and stochastic gradient descent method. A convergence analysis of D2GM (in 1D and using the upwind flux) is provided in Section 3. Numerical results with the dimensionality of random variables up to 200 for (stochastic) linear conservation law and (stochastic) Burgers' equation are shown in Section 4. Conclusions are drawn in Section 5.

2. Deep Learning Based Discontinuous Galerkin Method

In this section, we describe the D2GM in details. First, we introduce the construction of a DNN and build the discontinuous element space using the DNN. The associated loss function based on the DG method is then proposed with the enforcement of boundary/initial conditions. The stochastic gradient descent method is employed to find the optimal solution.

2.1. Deep neural network

A DNN contains a series of layers, and each layer has several neurons linked to pre- and post- layer neurons. Neurons are connected with an affine transformation and a nonlinear activation function. Such a DNN can be viewed as a nonlinear approximation of the target function. Precisely, suppose that the DNN has L layers, i.e., an input layer, $L-1$ hidden layers, and an output layer. The input layer takes $\boldsymbol{z}^0 = (t, \boldsymbol{x}, \boldsymbol{\omega})$ as the input and the output layer gives $z^L = \mathcal{N}(t, \boldsymbol{x}, \boldsymbol{\omega})$ as the output, where t is the temporal variable, $\boldsymbol{x}$ is the spatial variable, and $\boldsymbol{\omega}$ is the random variable. The relation between the l-th layer and the $(l+1)$-st layer $(l = 0, 1, \ldots, L-1)$ is given by

$$\begin{aligned} &\boldsymbol{z}^0 = (t, \boldsymbol{x}, \boldsymbol{\omega}) \quad \text{input}, \\ &\mathbf{z}_k^{l+1} = \sigma_l(\mathbf{w}_k^{l+1} \cdot \boldsymbol{z}^l + b_k^l), \quad l = 0, 1, \ldots, L-1, \quad 1 \le k \le m_{l+1}\,, \\ &\mathcal{N}(t, \boldsymbol{x}, \boldsymbol{\omega}) = \mathbf{w}^{L+1} z^L \quad \text{output}, \end{aligned} \tag{2.1}$$

where m_l is the number of neurons in the $l-$th layer ($m_L = 1$), σ is the activation function. Some popular σ includes the rectified linear unit (ReLU) function $\sigma(x) = \max(x, 0)$ and the sigmoid function $\sigma(x) = 1/(1 + e^{-x})$.

Let $\theta = (\theta_1, \cdots, \theta_J)$ include all $\mathbf{w}_k^l$ and b_k^l, with J the total number of coefficients in (2.1), which are to be obtained by minimizing the loss function, in order to match the DNN solution $\mathcal{N}(t, \boldsymbol{x}, \boldsymbol{\omega})$ with the target function $u(t, \boldsymbol{x}, \boldsymbol{\omega})$.

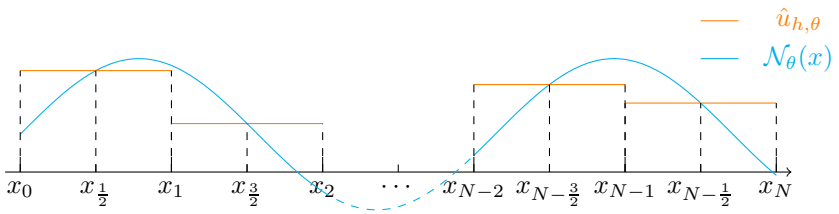


Fig. 2.1. Illustration of the discontinuous element space.

2.2. Discontinuous element basis

For brevity, we use the unit interval $[0,1]$ for demonstration. Denote

$$0 = x_0 < x_{\frac{1}{2}} < x_1 < \cdots < x_{N-\frac{1}{2}} < x_N = 1, \tag{2.2}$$

where $x_{i+\frac{1}{2}}$ is the middle point of the cell $I_i = [x_i, x_{i+1}]$. We also denote $\Delta x_i = x_{i+1} - x_i$ and $h = \max_i \Delta x_i$. For the uniform mesh, $h = \Delta x_i = \frac{1}{N}$.

The discontinuous element space is defined as

$$V_h^0 = \left\{ v : v|_{I_i} \in P^0(I_i),\ 0 \le i < N-1 \right\}, \tag{2.3}$$

where P^0 denotes the 0-th order polynomial (constant). We use a DNN to represent the element in V_h^0 as

$$u_{h,\theta}(x) = \mathcal{N}_\theta(x_{i+\frac{1}{2}}), \quad \text{if} \quad x_i \le x < x_{i+1}, \tag{2.4}$$

where θ is the parameter set in the DNN to be optimized. This can also be expressed in a way more like the Galerkin formulation

$$u_{h,\theta}(x) = \sum_{i=0}^{N-1} \mathcal{N}_\theta(x_{i+\frac{1}{2}})\varphi_i(x), \quad \varphi_i(x) = \begin{cases} 1, & x_i \le x < x_{i+1}, \\ 0, & \text{otherwise.} \end{cases} \tag{2.5}$$

This procedure is illustrated in Fig. 2.1. This definition can be generalized to the space of high-order piecewise polynomials

$$V_h^K = \left\{ v : v|_{I_i} \in P^K(I_i),\ 1 \le i < N-1 \right\}, \tag{2.6}$$

and any element in the space can be represented by $K+1$ DNNs $\mathcal{N}_\theta^j$, $j = 0, \ldots, K$, as

$$u_{h,\theta}(x) = \sum_{j=0}^{K} \sum_{i=0}^{N-1} \mathcal{N}_\theta^j(x_{i+\frac{1}{2}})\varphi_i^j(x), \tag{2.7}$$

where $\varphi_i^j(x)$ is the j-th order Legendre polynomial defined in I_i.

In high dimensions, this definition of V_h^0 can be easily generalized

$$u_{h,\theta}(\mathbf{x}) = \sum_{\boldsymbol{i}} \mathcal{N}_\theta(x_{\boldsymbol{i}+\frac{1}{2}})\varphi_{\boldsymbol{i}}(\mathbf{x}), \quad \varphi_{\boldsymbol{i}}(\mathbf{x}) = \begin{cases} 1, & \mathbf{x} \in I_{\boldsymbol{i}}, \\ 0, & \text{otherwise,} \end{cases} \tag{2.8}$$

where $\boldsymbol{x} = (x^1, x^2, \cdots, x^d) \in \mathbb{R}^d$, $\boldsymbol{i} = (i_1, i_2, \cdots, i_d)$ is a multi-dimensional index vector, $I_{\boldsymbol{i}} = [x_{i_1}^1, x_{i_1+1}^1) \times [x_{i_2}^2, x_{i_2+1}^2) \times \cdots \times [x_{i_d}^d, x_{i_d+1}^d)$, and $x_{\boldsymbol{i}+\frac{1}{2}}$ represents the center of $I_{\boldsymbol{i}}$. The generalization of V_h^k to the high-dimensional case can be done in a similar manner.

2.3. The DG method for hyperbolic conservation law

In this work, we consider the hyperbolic problem with random uncertainties of the following form:

$$u_t + \nabla_{\boldsymbol{x}} \cdot \boldsymbol{f}(u) = 0, \tag{2.9}$$

$$u(0, \boldsymbol{x}, \boldsymbol{\omega}) = u_0(\boldsymbol{x}, \boldsymbol{\omega}) \tag{2.10}$$

defined for $(t, \boldsymbol{x}, \boldsymbol{\omega}) \in [0, T] \times D \times \Omega$. Here $\boldsymbol{\omega}$ is a high-dimensional random variable representing uncertainties (or random inputs). The solution $u = u(t, \boldsymbol{x}, \boldsymbol{\omega})$ then depends on $\boldsymbol{\omega}$.

First, we will discrete the equation on spatial space. The semi-discrete DG method for solving (2.9) is defined as follows: Find the unique solution $u_h(t, \boldsymbol{x}, \boldsymbol{\omega}) \in V_h^k$ such that, for any test function $v_h \in V_h^k$ and all $0 \le i < N$, one has

$$\begin{aligned} &\frac{\mathrm{d}}{\mathrm{d}t} \left(u_h(t, \boldsymbol{x}, \boldsymbol{\omega}), v_h(\boldsymbol{x})\right)_{I_{\boldsymbol{i}}} - \left(\boldsymbol{f}(u_h(t, \boldsymbol{x}, \boldsymbol{\omega})), \nabla v_h(\boldsymbol{x})\right)_{I_{\boldsymbol{i}}} \\ &\quad + \hat{\boldsymbol{f}}(u_h(t, \boldsymbol{x}, \boldsymbol{\omega})) \cdot \mathbf{n} v_h(\boldsymbol{x}))|_{\partial I_{\boldsymbol{i}}} = 0, \end{aligned} \tag{2.11}$$

where $\boldsymbol{n}$ is the outward unit normal vector along $\partial I_{\boldsymbol{i}}$. In 1D, one has

$$\frac{\mathrm{d}}{\mathrm{d}t} \left(u_h(t, x, \boldsymbol{\omega}), v_h(x)\right)_{I_i} - \left(f(u_h(t, x, \boldsymbol{\omega})), v_h'(x)\right)_{I_i} + \hat{f}_{i+1} v_h(x_{i+1}^-) - \hat{f}_i v_h(x_i^+) = 0,$$

where the one-sided limit is defined as

$$v^{\pm}(x_j) = v(x_j^{\pm}) = \lim_{x \to x_j^{\pm}} v(x),$$

and the inner product is defined as

$$(a(t, x, \boldsymbol{\omega}), b(x))_D = \int_D a(t, x, \boldsymbol{\omega}) \cdot b(x) \mathrm{d}x.$$

Here $\hat{f}_i$ is a numerical flux, which is a single-valued function defined at the interface x_i and in general depends on the values of the numerical solution u_h from both sides of the interface. There are several choices to choose the flux [5] and we use the upwind flux

$$\hat{f}^{\mathrm{upwind}}(u^-, u^+) = \begin{cases} f(u^-), & \text{if} \quad a \ge 0, \\ f(u^+), & \text{if} \quad a < 0 \end{cases}$$

for the linear case $f(u) = au$ and Godunov flux

$$\hat{f}^{\mathrm{God}}(u^-, u^+) = \begin{cases} \min\limits_{u^- \le u \le u^+} f(u), & \text{if} \quad u^- < u^+, \\ \max\limits_{u^+ \le u \le u^-} f(u), & \text{if} \quad u^+ < u^+ \end{cases}$$

for the nonlinear case in this work. In high dimensions,

$$\hat{\boldsymbol{f}}(u_h(t, \boldsymbol{x}, \boldsymbol{\omega})) \quad \text{in} \ (\hat{\boldsymbol{f}}(u_h(t, \boldsymbol{x}, \boldsymbol{\omega})), \mathbf{n} v_h(\boldsymbol{x}))|_{\partial I_{\boldsymbol{i}}}$$

is replaced by the numerical flux on quadrature points.

The semi-discrete formulation (2.11) includes the temporal derivative, which needs to be discretized. A simple idea is to use the AutoGrad in PyTorch or TensorFlow, which provides

the temporal derivative automatically by back propagation. This kind of approach is widely used in solving PDEs with spatial derivatives evolved in the loss function [15,32]. We also introduce the temporal discretization with steps $0 = t_0 < t_1 < \cdots < t_N = T$ and $t_{n+1} - t_n = \Delta t$, and the semi-discrete formulation (2.11) becomes

$$\begin{aligned}&\left(\frac{u_h(t_{n+1},\boldsymbol{x},\boldsymbol{\omega}) - u_h(t_n,\boldsymbol{x},\boldsymbol{\omega})}{\Delta t}, v_h(\boldsymbol{x})\right)_{I_i}\\ &\quad - (\boldsymbol{f}(u_h(t_n,\boldsymbol{x},\boldsymbol{\omega})), \nabla v_h(\boldsymbol{x}))_{I_i}\\ &\quad + (\hat{\boldsymbol{f}}(u_h(t,\boldsymbol{x},\boldsymbol{\omega})), \mathbf{n} v_h(\boldsymbol{x}))|_{\partial I_i} = 0.\end{aligned} \tag{2.12}$$

In 1D, (2.12) reduces to

$$\begin{aligned}&\left(\frac{u_h(t_{n+1},x,\boldsymbol{\omega}) - u_h(t_n,x,\boldsymbol{\omega})}{\Delta t}, v_h(x)\right)_{I_i}\\ &\quad - (f(u_h(t_n,x,\boldsymbol{\omega})), v_h'(x))_{I_i}\\ &\quad + \hat{f}_{i+1} v_h(x_{i+1}^-) - \hat{f}_i v_h(x_i^+) = 0.\end{aligned} \tag{2.13}$$

Consider the DG approximation

$$u_{h,\theta}(t,x,\boldsymbol{\omega}) = \sum_{j'=0}^{K}\sum_{i'=1}^{N-1} \mathcal{N}_\theta^{j'}(t, x_{i'+\frac{1}{2}}, \boldsymbol{\omega})\varphi_{i'}^{j'}(x)\,.$$

Substituting this into (2.13), choosing $v_h(x) = \varphi_i^j(x)$, and using the orthogonality of the Legendre polynomials, we have

$$\begin{aligned}L_{i,j,n} \triangleq\ & \frac{\mathcal{N}_\theta^j(t_{n+1}, x_{i+\frac{1}{2}}, \boldsymbol{\omega}) - \mathcal{N}_\theta^j(t_n, x_{i+\frac{1}{2}}, \boldsymbol{\omega})}{\Delta t}\\ &- \left(f(u_{h,\theta}(t_n,x,\boldsymbol{\omega})), \frac{\mathrm{d}\varphi_i^j(x)}{\mathrm{d}x}\right)_{I_i}\\ &+ \hat{f}_{i+1}\varphi_i^j(x_{i+1}^-) - \hat{f}_i\varphi_i^j(x_i^+) = 0.\end{aligned} \tag{2.14}$$

The second term above can be further simplified when f is specified. For example, for linear conservation law when $f(u) = u$,

$$\left(f(u_{h,\theta}(t_n,x,\boldsymbol{\omega})), \frac{\mathrm{d}\varphi_i^j(x)}{\mathrm{d}x}\right)_{I_i} = \sum_{k=0}^{K} \mathcal{N}_\theta^k(t_n, x_{i+\frac{1}{2}}, \boldsymbol{\omega}) C_k^j, \quad j = 0,\ldots,K, \tag{2.15}$$

where $C_k^j = (\varphi_i^k(x), \frac{\mathrm{d}\varphi_i^j(x)}{\mathrm{d}x})_{I_i}$, and for Burgers' equation when $f(u) = \frac{1}{2}u^2$,

$$\left(f(u_{h,\theta}(t_n,x,\boldsymbol{\omega})), \frac{\mathrm{d}\varphi_i^j(x)}{\mathrm{d}x}\right)_{I_i} = \sum_{l=0}^{K}\sum_{l'=0}^{K} \mathcal{N}_\theta^l(t_n, x_{i+\frac{1}{2}}, \boldsymbol{\omega})\mathcal{N}_\theta^{l'}(t_n, x_{i+\frac{1}{2}}, \boldsymbol{\omega}) C_{l,l'}^j,$$

where $C_{l,l'}^j = (\varphi_i^l(x)\varphi_i^{l'}(x), \frac{\mathrm{d}\varphi_i^j(x)}{\mathrm{d}x})$. Therefore, the loss function for the DNN is defined as the residual error of (2.14) in the L^2 sense

$$\mathcal{L}(\theta) = \left(h\,\Delta t \sum_{i,j,n} L_{i,j,n}^2\right)^{\frac{1}{2}}, \tag{2.16}$$

and the DNN solution $\mathcal{N}_\theta$ is the solution that minimizes the loss function

$$\min_\theta \mathcal{L}(\theta). \tag{2.17}$$

Note that in this model the random variable $\boldsymbol{\omega}$ is still continuous and no discretization is applied in the random space. In numerical experiments, we apply the MC method for the random variable.

2.4. Boundary and initial conditions

Given $u(0, \boldsymbol{x}, \boldsymbol{\omega}) = u_0(\boldsymbol{x}, \boldsymbol{\omega})$, one can use

$$u_\theta(t, \boldsymbol{x}, \boldsymbol{\omega}) = t\mathcal{N}_\theta(t, \boldsymbol{x}, \boldsymbol{\omega}) + h(\boldsymbol{x}, \boldsymbol{\omega}) \tag{2.18}$$

to enforce the initial condition.

There are a couple of ways to enforce boundary conditions. The most straightforward way is to add a penalty term into the loss function. For example, the penalty term for Dirichlet boundary condition can be expressed as $\lambda\|u - g\|^2_{\partial D}$ with the penalty parameter λ. Another way is to build a DNN that satisfies the boundary condition exactly. For Dirichlet boundary condition, such a DNN can be constructed as

$$u_\theta(t, \boldsymbol{x}, \boldsymbol{\omega}) = L(\boldsymbol{x})\mathcal{N}_\theta(t, \boldsymbol{x}, \boldsymbol{\omega}) + G(\boldsymbol{x}, \boldsymbol{\omega}), \tag{2.19}$$

where $L(\boldsymbol{x})$ is a distance function that takes 0 on ∂D and is strictly positive inside D, $\mathcal{N}_\theta(t, \boldsymbol{x})$ is the neural network, and $G(\boldsymbol{x})$ is a smooth extension of $g(\boldsymbol{x})$ and equals $g(\boldsymbol{x})$ on ∂D.

In the current work, we can enforce the exact boundary condition on the numerical solution, i.e., $u_0 = u(x_0)$ and $u_N = u(x_N)$. For periodic boundary condition, we have

$$u_0 = u_{N-1}, \quad u_N = u_1. \tag{2.20}$$

This idea works for a grid-based method, and is applicable for all other kinds of boundary conditions.

2.5. Stochastic gradient descent method

In the DG method, for the high dimensional case, the number of degrees of freedom (dofs) scales like $(1/h)^d$ with d the dimensionality. Therefore, the classical method suffers from the curse of dimensionality. To overcome this difficulty, we apply the idea of stochastic gradient descend (SGD) method to evaluate the loss function (2.16) by selecting mesh points randomly over the index set i, j, k in each iteration with a fixed number of points. For the random variable, we also apply the MC method with a fixed number of points in the random space. Overall, the proposed method overcomes the curse of dimenisonality by design.

Here we give a summary of our algorithm. First, a neural network in the form of (2.5) or (2.7) is used to approximate the solution of the parameterized equation with boundary and initial conditions (2.9). The optimal set of parameters θ in the neural network is obtained by minimizing the loss function (2.16). Note that the loss function (2.16) still depends on the random variable $\boldsymbol{\omega}$ continuously. Therefore, in the SGD method, we generate N i.i.d. samples $\{\boldsymbol{\omega}_\ell\}_{\ell=1}^N$ and minimize

$$\left(\frac{h\,\Delta t}{N}\sum_\ell \sum_{i,j,n} L^2_{i,j,n}(\boldsymbol{\omega}_\ell)\right)^{\frac{1}{2}}.$$

The optimal θ is found after the minimization process converges. Afterwards, statistic properties of the solution can be computed by sampling the random variable in the neural network solution (2.5) or (2.7) with the Monte Carlo method.

3. Convergence

The convergence of the DNN solution can be established through standard Lax equivalence theorem kind of augument: consistency and stability imply convergence. We first state some preparation results which give consistency of the DNN approximation. The main reason that the DNN approximation (2.1) works is because of the *universal approximation theorem*, established in [11, 16].

To make the presentation simple and clear, we consider the deterministic (no $\boldsymbol{\omega}$ dependence) Eq. (2.9) over $[0,T]\times[0,1]$ with periodic boundary condition and assume that $f\in C^1$ and $f'>0$, thus the upwind scheme on uniform mesh writes

$$\begin{aligned}&\frac{U_i^{n+1}-U_i^n}{\Delta t}+\frac{f(U_i^n)-f(U_{i-1}^n)}{h}=0,\quad i=1,\dots,I,\quad n=0,\dots,N-1,\\&U_0^n=U_I^n,\\&U_i^0=u_0(x_i).\end{aligned}\tag{3.1}$$

In this section we will provide a proof of the convergence of the deep neural network approximation, along the line of [21]. Consider a continuous function $V(t,x)$ that satisfies Eq. (3.1) exactly at grid points

$$\begin{aligned}&\frac{V(t+\Delta t,x)-V(t,x)}{\Delta t}+\frac{f(V(t,x))-f(V(t,x-h))}{h}=0,\\&V(t,0)=V(t,1),\\&V(0,x)=u_0(x).\end{aligned}\tag{3.2}$$

Without loss of generality assume $u_0(x)\in C^1(D)$, with $D=[0,1]$ (if not the case one can interpolate through U_i^0 to get a C^1 function $V(0,x)$). For fixed Δt and h, clearly (3.2) implies that $V(t_n,x)\in C^1(D)$ for all $n\geq 0$, since $f\in C^1$.

From the definition of V clearly $V(t^n,x_i)=U_i^n$ for all $n\geq 0, 1\leq i\leq I$.

The loss function (2.16) is now

$$\mathcal{L}(\theta)=\left(h\Delta t\sum_{i=1}^{I}\sum_{n=0}^{N-1}\left|\frac{\mathcal{N}_\theta(t_n+\Delta t,x_i)-\mathcal{N}_\theta(t_n,x_i)}{\Delta t}+\frac{f(\mathcal{N}_\theta(t_n,x_i))-f(\mathcal{N}_\theta(t_n,x_i-h)}{h}\right|^2\right)^{\frac{1}{2}}.\tag{3.3}$$

It is obvious that the definition of V depends on the mesh size Δt and h. Therefore, for a fixed mesh size, we can apply the universal approximation theory and find a $\mathcal{N}_\theta$ to approximate V. Below we adopt the universal approximation theory to our setting.

Theorem 3.1. *Let σ be any non-polynomial function in $C^1(\mathbb{R})$ and $K=[0,T]\times D$ a compact set in $\mathbb{R}^2$. Then for any $\delta>0$, there is a network* (2.1) *such that*

$$\|V-\mathcal{N}_\theta\|_{W^{1,\infty}(K)}<\delta\,.$$

The above result implies that $\mathcal{N}_\theta$ depends on the mesh size, and we need to change the network as the mesh size reduces. The next theorem establishes the consistency of the DNN approximation.

Theorem 3.2. *Assume that the number of layers $L = 2$ and that the solution V to (3.2) belongs to $C^1([0,T] \times [0,1])$, and the activation function $\sigma(x) \in C^2$ is non-polynomial. Then for any $\delta > 0$, there exists θ and a sequence of the DNN solutions, denoted by $\mathcal{N}_\theta = \mathcal{N}(t,x;\theta)$, such that when the number of parameters is sufficiently large,*

$$|\mathcal{L}(\theta)| < C(T)\delta$$

for some positive constant $C(T)$ that may depend on T.

Proof. By (3.2),

$$\begin{aligned}\mathcal{I}_n(x,\theta) &= \frac{\mathcal{N}_\theta(t+\Delta t,x;\theta) - \mathcal{N}_\theta(t,x;\theta)}{\Delta t} + \frac{f(\mathcal{N}_\theta(t,x;\theta)) - f(\mathcal{N}_\theta(t,x-h;\theta)}{h} \\ &= \frac{[\mathcal{N}_\theta(t+\Delta t,x;\theta) - V(t+\Delta t,x)]}{\Delta t} - \frac{[\mathcal{N}_\theta(t,x;\theta) - V(t,x)]}{\Delta t} \\ &\quad + \frac{[f(\mathcal{N}_\theta(t,x;\theta)) - f(V(t,x))]}{h} - \frac{[f(\mathcal{N}_\theta(t,x-h;\theta)) - f(V(t,x-h))]}{h}. \end{aligned} \tag{3.4}$$

Given any δ, by Theorem 3.1, we have

$$\begin{aligned} &\|\mathcal{N}_\theta(t+\Delta t,x;\theta) - V(t+\Delta t,x)\| \le \delta, \\ &\|\mathcal{N}_\theta(t,x;\theta) - V(t,x)\| \le \delta, \\ &\|f(\mathcal{N}_\theta(t,x;\theta)) - f(V(t,x))\| \le \|f_u'\|\delta, \\ &\|f(\mathcal{N}_\theta(t,x-h;\theta)) - f(V(t,x-h))\| \le \|f_u'\|\delta. \end{aligned} \tag{3.5}$$

Therefore, $\mathcal{I}_n(x,\theta)$ can be bounded by δ multiplied by a constant C depending on $\Delta t, h$ and f_u', i.e.,

$$\|\mathcal{I}_n(\cdot,\cdot,\theta)\|_{l^\infty} \le C\delta\,. \tag{3.6}$$

Thus, by the Cauchy-Scharwtz inequality and the boundedness of D, the loss function in (3.3) can be bounded by δ multiplied by a constant that depends on $|D|$ and T as

$$\mathcal{L}(\theta) = \left(h\,\Delta t \sum_{i,n} |\mathcal{I}_n|^2\right)^{\frac{1}{2}} \le Ch\,IN\Delta t\,\delta \le CT\delta \tag{3.7}$$

since $Ih = 1$ and $n\,\Delta t \le T$. □

The above theorem shows that one can find the parameter θ such that the loss function convergences to zero. This shows the *consistency* of the DNN approximation. In fact the loss function $\mathcal{L}$ can be viewed as the truncation error of the DNN approximation, which will be made clear in the proof of Theorem 3.3. Note that Theorem 3.2 does not imply that $\mathcal{N}$ converges to the solution of the original problem (2.9). Next we prove the convergence of the DNN approximation, based on the *stability* argument.

Theorem 3.3. *Let θ_J be the sequence defined in Theorem 3.2, and let $\mathcal{N}_\theta$ be the solution to (2.17) and V be the classical numerical solution to (3.2), then*

$$\|\mathcal{N}_\theta(t_n,\cdot;\theta_J) - V(\cdot)\| \le \|\mathcal{N}_\theta(0,\cdot;\theta) - u(0,\cdot)\| + C(T)\delta\,.$$

Consequently,

$$\|\mathcal{N}_\theta(t_n, x_i; \theta_J) - U_j^n\| \le \|\mathcal{N}_\theta(0, \cdot; \theta) - u(0, \cdot)\| + C(T)\delta \quad \textit{for} \quad 1 \le i \le I, \quad n > 0.$$

Proof. Let

$$\mathcal{E}_i^n(\theta) = \mathcal{N}_\theta(t_n, x_i; \theta_J) - U_j^n = \mathcal{N}_\theta(t_n, x_i; \theta_J) - V(t^n, x_i) = \mathcal{N}_i^n - Vi^n.$$

Clearly, one has

$$\frac{\mathcal{E}_i^{n+1} - \mathcal{E}_i^n}{\Delta t} + \frac{f(\mathcal{N}_i^n) - f(\mathcal{N}_{i-1}^n)}{h} - \frac{f(u_i^n) - f(u_{i-1}^n)}{h} = \mathcal{I}_n(x_i, \theta).$$

Let $\lambda = \Delta t/h$. Thus

$$\begin{aligned}
\mathcal{E}_i^{n+1} &= \mathcal{E}_i^n + \lambda \left[(f(u_i^n) - f(\mathcal{N}_i^n)) - (f(u_{i-1}^n) - f(\mathcal{N}_{i-1}^n)) \right] + \Delta t \mathcal{I}_n(x_i, \theta) \\
&= \mathcal{E}_i^n + \lambda \left[-f'(\eta_i^n)\mathcal{E}_j^n + f'(\eta_{i-1}^n)\mathcal{E}_{i-1}^n \right] + \Delta t \mathcal{I}_n(x_i, \theta) \\
&= (1 - \lambda f'(\eta_i^n))\mathcal{E}_i^n + \lambda f'(\eta_{i-1}^n)\mathcal{E}_{i-1}^n + \Delta t \mathcal{I}_n(x_i, \theta), \qquad (3.8)
\end{aligned}$$

where η_j^n is a point between $\mathcal{N}_i^n$ and u_j^n. Now taking the l^1 norm on the above equality, assuming

$$\lambda \sup_\eta f'(\eta) \le 1,$$

and using the periodic boundary condition, one gets, for all $n \ge 0$,

$$\begin{aligned}
\|\mathcal{E}^{n+1}\|_{l^1} = \frac{1}{I}\sum_{i=1}^I |\mathcal{E}_i^{n+1}| &\le \frac{1}{I}\sum_{i=1}^I (1 - \lambda f'(\eta_i^n))|\mathcal{E}_i^n| + \frac{1}{I}\sum_{i=1}^I \lambda f'(\eta_{i-1}^n)|\mathcal{E}_{i-1}^n| + \Delta t \|\mathcal{I}_n(\cdot, \theta)\|_{l^1} \\
&\le \frac{1}{I}\sum_{i=1}^I (1 - \lambda f'(\eta_i^n))|\mathcal{E}_i^n| + \frac{1}{I}\sum_{i=1}^I \lambda f'(\eta_i^n)|\mathcal{E}_i^n| + \Delta t \|\mathcal{I}_n(\cdot, \theta)\|_{l^1} \\
&= \|\mathcal{E}^n\|_{l^1} + C\delta\Delta t. \qquad (3.9)
\end{aligned}$$

Consequently one has

$$\|\mathcal{E}^n\|_{l^1} \le \|\mathcal{E}^0\|_{l^1} + C\delta n\Delta t \le \|\mathcal{E}^0\|_{l^1} + CT\delta \qquad (3.10)$$

for all n such that $n\Delta t \le T$. Now the convergence $\mathcal{N} \to V$ as $J \to \infty$ is a consequence of Theorem 3.2, as long as one trains the initial data $\mathcal{E}^0$ well. □

Remark 3.4. It is straightforward to establish the convergence between the numerical solution V and the exact solution $u(t, x)$ by combining classical numerical analysis of the DG method [8]. This, together with Theorem 3.3, leads to the convergence of the DNN solution $\mathcal{N}_\theta$ to the exact solution $u(t, x)$. The proof also implies that both the DNN approximation error and the discretization error contribute to the approximation error in the D2GM, as demonstrated in Section 4.

Remark 3.5. The convergence analysis can be generalized to high-dimensional problems with random variables. In these cases, the MC method is employed to sample the random variables or a subset of indices for the spatial variables. The sampling error is inversely proportional to the square root of the number of samples, which attributes to the convergence of the loss function proven in Theorem 3.2 while the other parts of the convergence proof remains unchanged. In practice, the sampling error contributes to the total approximation error and is kept small by using a large number of samples.

4. Numerical Results

There are four sources of error in the D2GM: the DNN approximation error, the discretization error, the optimization error, and the sampling error. A large number of samples are used so that the sampling error will not affect the observation numerically. For the optimization error, Adam (Adaptive moment method) is used to find the optimal solution. Therefore, the first two sources of error dominates the numerical performance of the D2GM. For a DNN with the large number of parameters, the DNN approximation error is small and the discretization error dominates. Therefore, for moderate grid size, the convergence rate of D2GM is observed in the classical sense. When the grid size is small, the DNN approximation contributes more to the total error and the convergence rate of DG will be lost. The convergence rate will be recovered if a DNN with more parameters is employed.

4.1. Linear conservation law

First, we use a linear conservation law without random inputs to illustrate our method

$$\begin{cases} 2d\pi u_t - \sum_{k=1}^{d} u_{x^k} = 0, \quad x \in [0,1]^d, \\ u(0,x) = h(x) = \sin\left(2\pi \sum_{k=1}^{d} x^k\right) \end{cases} \tag{4.1}$$

with periodic boundary condition, and the exact solution $u(t,x) = \sin(t + 2\pi \sum_{k=1}^{d} x^k)$, $d = 1,2,3$. For the first-order method, following (2.18), we construct the numerical solution that satisfies the initial condition exactly

$$u_{h,\theta}(t,\boldsymbol{x}) = \sum_{\boldsymbol{i}} [t\mathcal{N}_\theta(t, x_{\boldsymbol{i}+\frac{1}{2}}) + g(x_{\boldsymbol{i}+\frac{1}{2}})]\varphi_{\boldsymbol{i}}(\boldsymbol{x}), \quad \varphi_{\boldsymbol{i}}(\boldsymbol{x}) = \begin{cases} 1, & \boldsymbol{x} \in I_{\boldsymbol{i}}, \\ 0, & \text{otherwise}, \end{cases} \tag{4.2}$$

and enforce the periodic boundary condition according to (2.20). Following (2.8), we define the DNN represented coefficients as

$$U_i(t) = t\mathcal{N}(t, x_{i+\frac{1}{2}}) + \sin(2\pi x_{i+\frac{1}{2}})$$

in 1D,

$$U_{i_1,i_2}(t) = t\mathcal{N}(t, x^1_{i_1+\frac{1}{2}}, x^2_{i_2+\frac{1}{2}}) + \sin(2\pi(x^1_{i_1+\frac{1}{2}} + x^2_{i_2+\frac{1}{2}}))$$

in 2D, and

$$U_{i_1,i_2,i_3}(t) = t\mathcal{N}(t, x^1_{i_1+\frac{1}{2}},, x^2_{i_2+\frac{1}{2}}, x^3_{i_3+\frac{1}{2}}) + \sin(2\pi(x^1_{i_1+\frac{1}{2}} + x^2_{i_2+\frac{1}{2}} + x^3_{i_3+\frac{1}{2}}))$$

in 3D, respectively.

For the upwind scheme, in 3D, the loss function for the semi-discrete scheme and the fully discrete scheme based on the forward Euler method as

$$\mathcal{L}_{\text{semi}}(\theta) = \Bigg(\Delta t h^3 \sum_{i_i,i_2,i_3,j} \Bigg(6\pi \partial_t U_{i_1,i_2,i_3}(t_j) - \frac{U_{i_1+1,i_2,i_3}(t_j) - U_{i_1,i_2,i_3}(t_j)}{h} - \frac{U_{i_1,i_2+1,i_3}(t_j) - U_{i_1,i_2,i_3}(t_j)}{h} - \frac{U_{i_1,i_2,i_3+1}(t_j) - U_{i_1,i_2,i_3}(t_j)}{h} \Bigg)^2 \Bigg)^{\frac{1}{2}}, \tag{4.3}$$

and

$$\mathcal{L}_{\mathrm{FE}}(\theta)=\left(\Delta t h^3\sum_{i_i,i_2,i_3,j}\left(6\pi\frac{U_{i_1,i_2,i_3}(t_{j+1})-U_{i_1,i_2,i_3}(t_j)}{\Delta t}-\frac{U_{i_1+1,i_2,i_3}(t_j)-U_{i_1,i_2,i_3}(t_j)}{h}\right.\right.$$
$$\left.\left.-\frac{U_{i_1,i_2+1,i_3}(t_j)-U_{i_1,i_2,i_3}(t_j)}{h}-\frac{U_{i_1,i_2,i_3+1}(t_j)-U_{i_1,i_2,i_3}(t_j)}{h}\right)^2\right)^{\frac{1}{2}},\qquad(4.4)$$

respectively.

Numerical results of both loss functions are recorded in Table 4.1. The fully discrete method based on the forward Euler scheme (4.4) shows a better approximation accuracy than the semi-discrete method using AutoGrad (4.3). This implies that use of discrete derivative may lead to better results for time-dependent problems. For moderate mesh size h, the first-order convergence is observed with respect to h. For smaller h, the first-order convergence is lost but is recovered when a wider network with the width 200 is employed; see Table 4.2 for details.

For the second-order method, the approximate solution in 1D is constructed as

$$u_{h,\theta}(t,x)=\sum_i[U_i^0\varphi_i^0(x)+U_i^1\varphi_i^1(x)],\qquad(4.5)$$

where

$$\varphi_i^0(x)=\begin{cases}1, & x\in I_i,\\ 0, & \text{otherwise},\end{cases}\qquad(4.6a)$$

Table 4.1: The averaged L^2 relative error in the last 1000 steps and the convergence rate for the linear conservation law (4.1) with two loss functions (4.3) and (4.4). A neural network with 4 hidden layers and two shortcut connections is used and the batchsize is chosen as 10000. In 1D, the network width is set to be 20 and the total number of parameters is 1341. In 2D, the network width is set to 40 and the total number of parameters is 5121. In 3D, the network width is set to be 60 and the total number of parameters is 11341.

d	$h=\Delta t$	Fully discrete		Semi-discrete	
		error	order	error	order
1	1/10	2.86 e-01		3.04 e-01	
	1/20	1.50 e-01	0.93	1.58 e-01	0.93
	1/40	7.73 e-02	0.94	8.05 e-02	0.98
	1/80	3.95 e-02	0.96	8.39 e-02	-0.05
	1/160	2.10 e-02	0.91	7.01 e-02	0.25
	1/320	1.72 e-02	0.28	1.44 e-01	-1.04
2	1/10	3.32 e-01		3.43 e-01	
	1/20	1.72 e-01	0.90	1.81 e-01	0.91
	1/40	8.90 e-02	0.95	8.89 e-02	1.03
	1/80	4.68 e-02	0.92	6.00 e-02	0.56
	1/160	2.57 e-02	0.86	5.40 e-02	0.15
	1/320	1.87 e-02	0.45	5.64 e-02	-0.06
3	1/10	3.59 e-01		3.75 e-01	
	1/20	1.92 e-01	0.90	2.02 e-01	0.89
	1/40	9.95 e-02	0.95	1.03 e-01	0.96
	1/80	5.20 e-02	0.93	6.94 e-02	0.57
	1/160	3.14 e-02	0.72	9.45 e-02	-0.44
	1/320	2.09 e-02	0.58	1.16 e-01	-0.30

Table 4.2: The averaged L^2 relative error in the last 1000 steps and the convergence rate for the linear conservation law (4.1) using loss function (4.4) and a wider neural network with width 200 in 3D.

$h = \Delta t$	error	order
1/10	3.64 e-01	
1/20	1.92 e-01	0.92
1/40	9.92 e-02	0.95
1/80	5.04 e-02	0.97
1/160	2.54 e-02	0.98
1/320	1.29 e-02	0.97
1/640	6.78 e-03	0.93
1/1280	4.24 e-03	0.67

$$\varphi_i^1(x) = \begin{cases} (x - x_{i+\frac{1}{2}}), & x \in I_i, \\ 0, & \text{otherwise}, \end{cases} \tag{4.6b}$$

$$U_i^0(t) = t\mathcal{N}_{\theta_0}^0(t, x_{i+\frac{1}{2}}) + \sin(2\pi x_{i+\frac{1}{2}}), \tag{4.6c}$$

$$U_i^1(t) = t\mathcal{N}_{\theta_1}^1(t, x_{i+\frac{1}{2}}) + 2\pi\cos(2\pi x_{i+\frac{1}{2}}). \tag{4.6d}$$

Based on (2.15), the corresponding loss function consists of two contributions

$$\begin{aligned} \mathcal{L}_0(\theta_0) &= \left(\Delta t h \sum_{i,j} \left(2\pi \frac{U_i^0(t_{j+1}) - U_i^0(t_j)}{\Delta t} h + \hat{f}_{i+\frac{3}{2}} - \hat{f}_{i+\frac{1}{2}} \right)^2 \right)^{\frac{1}{2}}, \\ \mathcal{L}_1(\theta_1) &= \left(\Delta t h \sum_{i,j} \left(2\pi \frac{U_i^1(t + \Delta t) - U_i^1(t)}{\Delta t} \frac{h^3}{12} + h u_0 + \frac{h}{2} \hat{f}_{i+\frac{3}{2}} + \frac{h}{2} \hat{f}_{i+\frac{1}{2}} \right)^2 \right)^{\frac{1}{2}}. \end{aligned} \tag{4.7}$$

Often $\mathcal{L}_0(\theta_0)$ and $\mathcal{L}_1(\theta_1)$ are not of the same order of magnitude, which adds additional difficulties to minimize both terms simultaneously

$$\arg \min_{\{\theta_0, \theta_1\}} \mathcal{L}_0(\theta_0) + \mathcal{L}_1(\theta_1), \tag{4.8}$$

where θ_0, θ_1 are the parameters of neural networks to approximate U^0 and U^1, respectively. We use ADMM [3] to optimize U^0 and U^1. Numerical results are shown in Table 4.3. Compared

Table 4.3: The averaged L^2 relative error in the last 1000 steps and the convergence rate for the 1D linear conservation law (4.1) solved by the second-order scheme. A neural network with 4 hidden layers and two shortcut connections is used and the batchsize is chosen as 10000. The network width is set to be 20 and the total number of parameters is 1341.

$h = \sqrt{\Delta t}$	error	order
1/10	1.01 e-01	
1/20	3.69 e-02	1.40
1/40	2.24 e-02	0.79
1/80	1.15 e-02	0.95
1/160	1.06 e-02	0.12
1/320	7.81 e-03	0.44

Table 4.4: The averaged L^2 relative error in the last 1000 steps and the convergence rate for the 1D linear conservation law (4.1) solved by the second-order scheme. A neural network with 6 hidden layers and two shortcut connections is used and the batchsize is chosen as 10000. The network width is set to be 60 and the total number of parameters is 12951.

$h = \sqrt{\Delta t}$	error	order
1/10	7.36 e-02	
1/20	1.45 e-02	2.34
1/40	2.33 e-03	2.63
1/80	6.31 e-04	1.88
1/160	2.11 e-04	1.57

with Table 4.1, we can find the second-order scheme has a better accuracy when two identical networks are applied. If a wider and deeper network is employed, then the second-order scheme is obtained with high accuracy; see Table 4.4.

Results of the first-order and second-order schemes are summarized in Fig. 4.1. The second-order scheme always has a better accuracy than the first-order scheme. A DNN with more parameters reduces the DNN approximation error and thus the convergence rate can be obtained over a larger range of grid size.

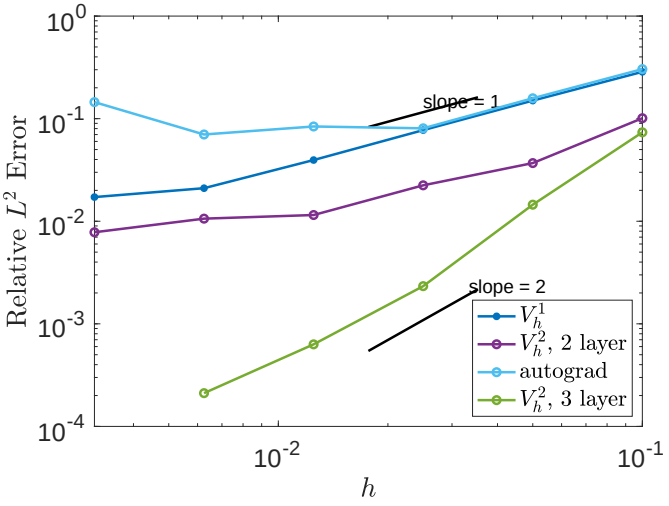


Fig. 4.1. The averaged error of two schemes for the 1D conservation law with respect to the mesh size h. V_h^1 represents the first-order scheme with the forward Euler method in time and V_h^2 represents the second-order scheme with two DNNs and the forward Euler method in time. Autograd represents the first-order scheme with AutoGrad in time.

4.2. Burgers' equation

Consider the Burgers' equation

$$u_t + \left(\frac{u^2}{2}\right)_x = 0 \tag{4.9}$$

with initial condition

$$u(0,x)=\begin{cases}1, & x<0,\\ 0, & x>0,\end{cases} \tag{4.10}$$

and reflecting boundary condition. The exact solution is discontinuous. The numerical solution is constructed as

$$u_\theta(t,x)=\begin{cases}\mathcal{N}_\theta(t,x_{i+\frac{1}{2}})\varphi_i(x), & t>0, \quad x\in[x_i,x_{i+1}),\\ u(0,x_{i+\frac{1}{2}}), & t=0,\end{cases} \tag{4.11}$$

where $\varphi_i(x)$ is defined in (2.5). This means that the numerical solution is approximated by a DNN at any point when $t>0$ and uses the exact solution when $t=0$. We divide the time interval $(0,T)$ into grids and assume that the temporal step size equals the spatial mesh size for simplicity.

The loss function for the semi-discrete scheme is

$$\begin{aligned}\mathcal{L}_{\text{semi}}(\theta)=&\left(\Delta t h\sum_{i,j}\left(\frac{\partial u_\theta(t_j,x_{i+\frac{1}{2}})}{\partial t}-\hat{f}^{\text{God}}(u_\theta(t_j,x_i^-),u_\theta(t_j,x_i^+))\right.\right.\\ &\left.\left.+\hat{f}^{\text{God}}(u_\theta(t_j,x_{i+1}^-),u_\theta(t_j,x_{i+1}^+))\right)^2\right)^{\frac{1}{2}},\end{aligned} \tag{4.12}$$

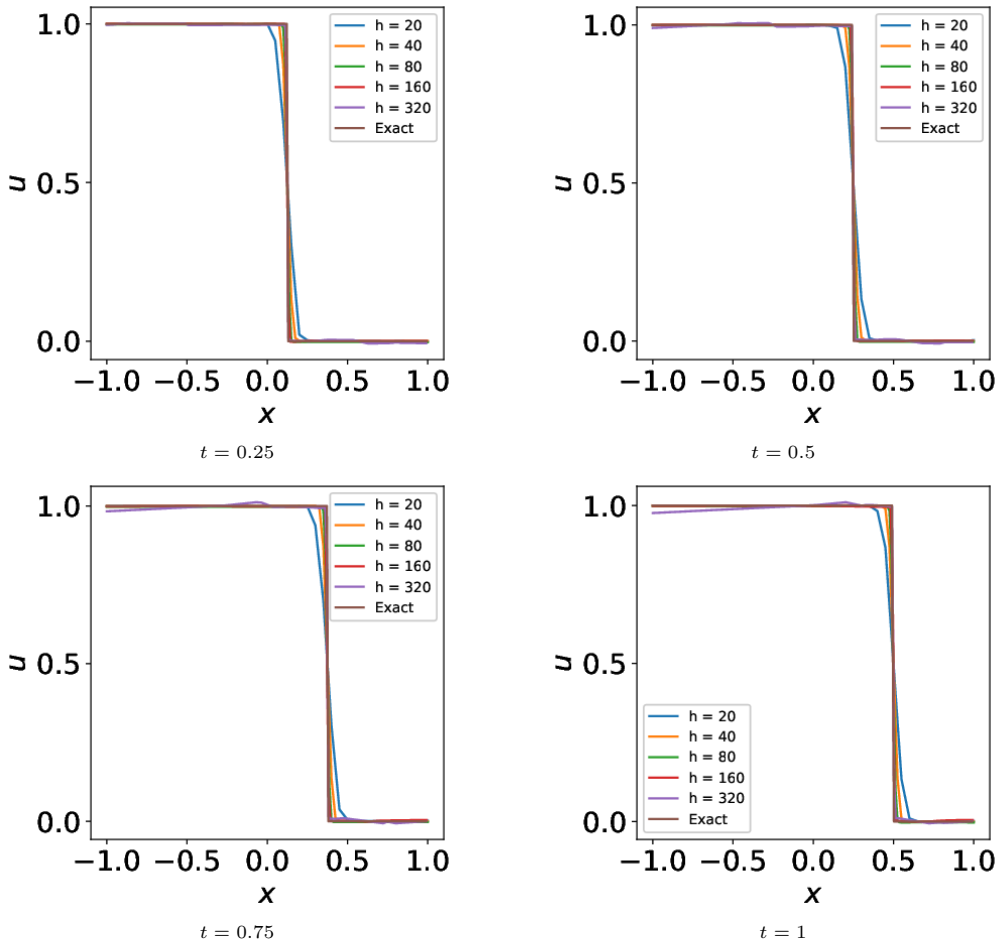
and the loss function for the fully-discrete scheme using the forward Euler method is

$$\begin{aligned}\mathcal{L}_{\text{FE}}(\theta)=&\left(\sum_{i,j}\left(\frac{u_\theta(t_{j+1},x_{i+\frac{1}{2}})-u_\theta(t_j,x_{i+\frac{1}{2}})}{\Delta t}-\hat{f}^{\text{God}}(u_\theta(t_j,x_i^-),u_\theta(t_j,x_i^+))\right.\right.\\ &\left.\left.+\hat{f}^{\text{God}}(u_\theta(t_j,x_{i+1}^-),u_\theta(t_j,x_{i+1}^+))\right)^2\right)^{\frac{1}{2}},\end{aligned} \tag{4.13}$$

respectively. The error of these two loss functions is shown in Table 4.5. It is observed that the forward Euler method produces much better results than the autograd method for the Burgers' equation (4.9) with a non-smooth solution (4.10). The detailed solution profiles are visualized in Fig. 4.2.

Table 4.5: The averaged L^2 relative error in the last 1000 steps and the convergence rate for the Burgers' equation (4.9)-(4.10) with two loss functions (4.13) and (4.12). A neural network with 4 hidden layers and two shortcut connections is used and the batchsize is chosen as 10000. In 1D, the network width is set to be 20 and the total number of parameters is 1341.

$h=\Delta t$	Fully discrete	Semi-discrete
1/10	9.87 e-02	3.82 e-01
1/20	4.88 e-02	3.37 e-01
1/40	3.48 e-02	3.03 e-01
1/80	2.58 e-02	3.12 e-01
1/160	1.84 e-02	1.91 e-01
1/320	1.73 e-02	3.86 e-01



$t = 0.25$ $t = 0.5$ $t = 0.75$ $t = 1$

Fig. 4.2. 1D solution profiles of the Burgers' equation (4.9) approximated by the neural network solution (4.11) with different mesh sizes h.

4.3. Stochastic linear conservation law

Consider the stochastic linear conservation law

$$2d\pi u_t - \left(1 + \exp\left(-\sum_{j=1}^{s} \omega^j\right)^2\right) \sum_{i=1}^{d} u_{x^i} = 0 \tag{4.14}$$

with periodic boundary condition and initial condition

$$u(0, \boldsymbol{x}, \boldsymbol{\omega}) = \sin\left(2\pi \sum_{i=1}^{d} x^i\right).$$

The exact solution of the problem is

$$u(t, \boldsymbol{x}, \boldsymbol{\omega}) = \sin\left(\left(1 + \exp\left(-\sum_{j=1}^{s} \omega^j\right)^2\right) t + 2\pi \sum_{i=1}^{d} x^i\right).$$

Table 4.6: The averaged L^2 relative error in the last 1000 steps and the convergence rate for the stochastic conservation law (4.14). The neural network used here has 6 hidden layers and 3 shortcut connections. When $s = 50$, the network width is 100 and the total number of parameters is 56101. When $s = 100$, the network width is 200 and the total number of parameters is 222201. The batchsize is 200000.

s	$h = \Delta t$	Expectation	Order	Variance	Order
50	1/40	1.54 e-01		2.13 e-01	
50	1/80	7.85 e-02	0.97	1.14 e-01	0.93
50	1/160	3.88 e-02	1.01	5.61 e-02	0.96
50	1/320	1.96 e-02	0.98	3.22 e-02	0.79
100	1/40	1.53 e-01		2.07 e-01	
100	1/80	7.83 e-02	0.97	1.12 e-01	0.88
100	1/160	3.93 e-02	0.99	5.82 e-02	0.95
100	1/320	2.01 e-02	0.96	2.93 e-02	0.98

The DNN solution is constructed as

$$u(t,\boldsymbol{x},\boldsymbol{\omega}) = \sum_{\boldsymbol{i}} [t\mathcal{N}_\theta(t,\boldsymbol{x}_{\boldsymbol{i}+\frac{1}{2}},\boldsymbol{\omega}) + g(\boldsymbol{x}_{\boldsymbol{i}+\frac{1}{2}})]\varphi_{\boldsymbol{i}}(\boldsymbol{x}), \tag{4.15}$$

where $\varphi_{\boldsymbol{i}}(\boldsymbol{x})$ is defined in (2.8). Since the fully discrete scheme works better than the semi-discrete scheme, we only use the fully discrete scheme with the forward Euler method in time. The corresponding loss function reads as

$$\mathcal{L}_{\mathrm{FE}}(\theta) = \left(\Delta t h^d \sum_{i,j} \left(2\pi \frac{u_\theta(t_{j+1},\boldsymbol{x}_{\boldsymbol{i}+\frac{1}{2}},\boldsymbol{\omega}) - u_\theta(t_j,\boldsymbol{x}_{\boldsymbol{i}+\frac{1}{2}},\boldsymbol{\omega})}{\Delta t} \right.\right.$$
$$\left.\left. - \left(1 + \exp\left(-\sum_{j=1}^{s} \omega_j \right)^2 \right) \frac{u_\theta(t_j,\boldsymbol{x}_{\boldsymbol{i}+1},\omega) - u_\theta(t_j,\boldsymbol{x}_{\boldsymbol{i}},\boldsymbol{\omega})}{h} \right)^2 \right)^{\frac{1}{2}}. \tag{4.16}$$

Fig. 4.3 plots the expectation and the variance of the solution along the line $x_1 = x_2 = x_3$ when $d = 3$, $s = 2$, and $s = 5$. Table 4.6 records the relative L^2 errors of the expectation and the variance for $s = 50$ and 100 respectively. The first-order accuracy is observed for the stochastic linear conservation law in both expectation and variance.

4.4. Stochastic Burgers' equation

Consider the stochastic Burgers' equation defined as

$$u_t + \left(\frac{u^2}{2} \right)_x = 0 \tag{4.17}$$

with initial condition

$$u(0,x,\boldsymbol{\omega}) = \begin{cases} 1 + \epsilon \sum_{i=1}^{s} \omega^i, & x < 0, \\ 0, & x > 0. \end{cases} \tag{4.18}$$

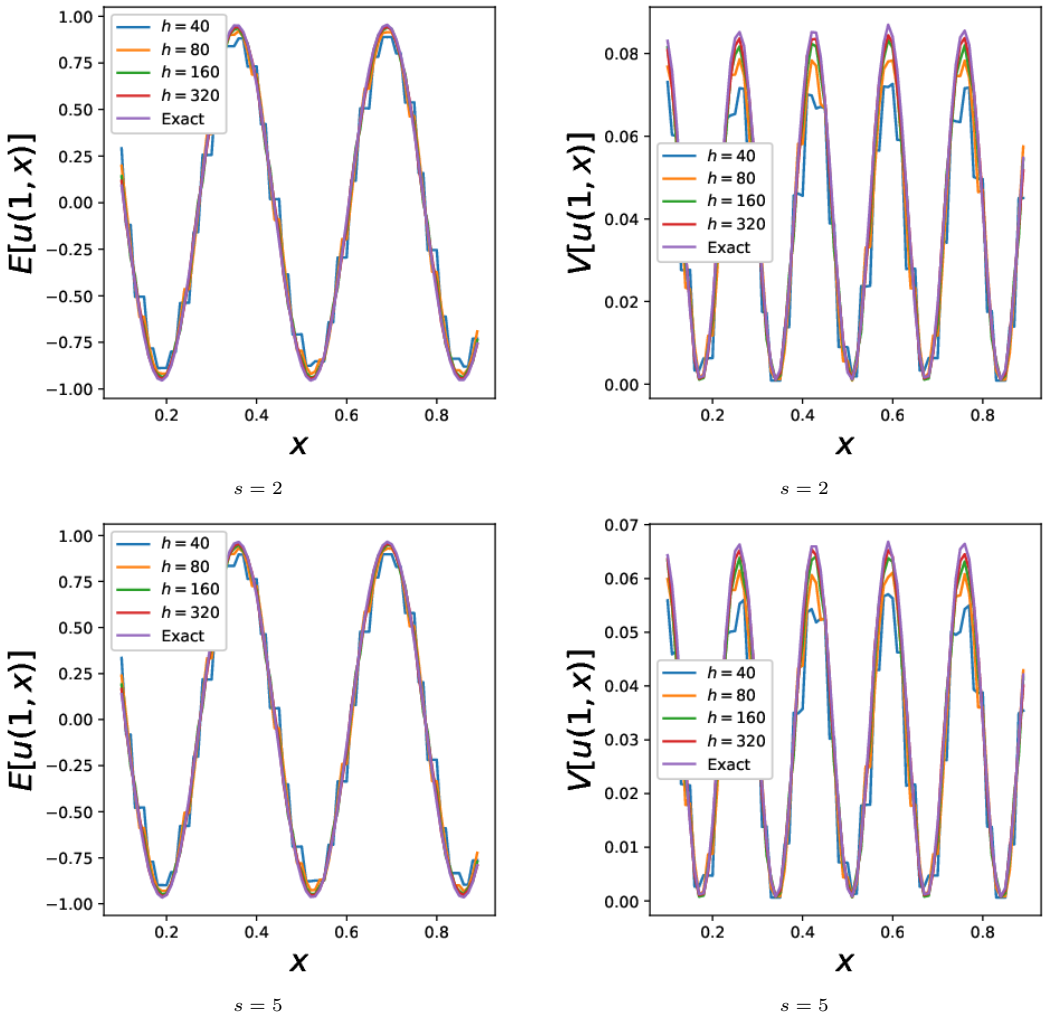


Fig. 4.3. Numerical and exact expectations and variances of the solution to the stochastic linear conservation law (4.14) along the line where $x_1 = x_2 = x_3$ when $d = 3$, $s = 2$ and $s = 5$ with different mesh sizes h. When $s = 2$, the network width is 40 and the total number of parameters is 8441. When $s = 5$, the network width is 50 and the total number of parameters is 13211. The batchsize is 200000.

The exact solution is

$$u(t, x, \boldsymbol{\omega}) = \begin{cases} z, & x < \dfrac{z}{2}, \\ 0, & x > \dfrac{z}{2}, \end{cases} \tag{4.19}$$

where $z = 1 + \epsilon \sum_{i=1}^{s} \omega^i$. The expectation of the solution is

$$\mathbb{E}_{\boldsymbol{\omega}}[u(t, x, \boldsymbol{\omega})] = \begin{cases} 1, & x < \dfrac{1-\epsilon}{2}, \\ \dfrac{1 - 4x^2 + 2\epsilon + \epsilon^2}{4\epsilon}, & \dfrac{1+\epsilon}{2} > x > \dfrac{1-\epsilon}{2}, \\ 0, & x > \dfrac{1+\epsilon}{2}. \end{cases} \tag{4.20}$$

Table 4.7: Network setups for stochastic equations with different number of random variables.

s	Number of hidden layers	Network width	Number of parameters
2	6	40	8441
5	6	50	13211
10	6	50	13451
50	6	100	55901
100	6	200	221801
200	6	400	883601

Table 4.8: Expectation and variance errors of the proposed method for the stochastic Burgers' equation when the MC method is used with the batchsize 10000.

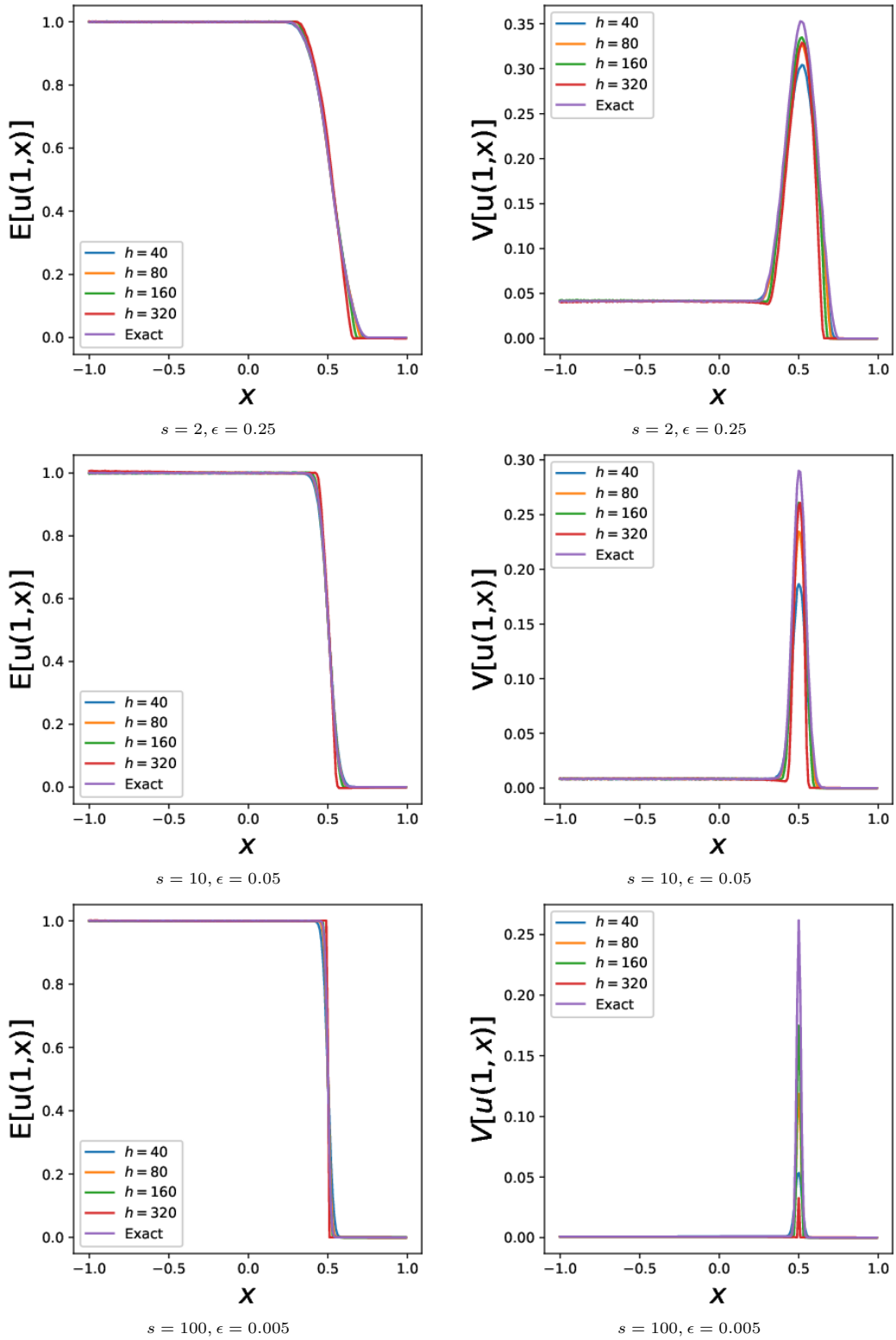
ϵ	s	h	Expectation error (L^2)	Variance error (L^2)
0.25	2	1/40	1.00 e-2	5.42 e-1
0.25	2	1/80	2.98 e-2	6.49 e-1
0.1	5	1/40	1.48 e-2	2.23 e-1
0.1	5	1/80	3.06 e-2	3.22 e-1
0.05	10	1/40	8.16 e-3	2.75 e-1
0.05	10	1/80	2.24 e-2	4.34 e-1
0.01	50	1/40	1.09 e-2	5.78 e-1
0.01	50	1/80	1.90 e-2	5.86 e-1
0.005	100	1/40	5.30 e-3	6.82 e-1
0.005	100	1/80	1.81 e-3	7.89 e-1
0.0025	200	1/40	1.02 e-2	8.96 e-1
0.0025	200	1/80	1.57 e-2	9.92 e-1

The reference variance of the solution is simulated by the MC method. The neural network setup for different s is listed in Table 4.7.

The approximate solution is constructed as

$$u_\theta(t,x,\boldsymbol{\omega}) = \begin{cases} \mathcal{N}_\theta(t, x_{i+\frac{1}{2}}, \boldsymbol{\omega})\varphi_i(x), & t > 0, \quad x \in (x_i, x_{i+1}), \\ u(0, x_{i+\frac{1}{2}}, \boldsymbol{\omega}), & t = 0, \end{cases} \tag{4.21}$$

and the loss function is the same as (4.13). Expectation and variance errors of the proposed method are recorded in Tables 4.8 and 4.9 when the batch size is 10000 and 50000, respectively. The relative L^2 error in expectation and variance reduces when the batch size is increased and the relative L^1 error is slightly better than the L^2 error. Furthermore, we apply the quasi-Monte Carlo method [4,6] to approximate the loss function; see Table 4.10. It is found that the error in this case is smaller than that of the MC method but cannot be further reduced with smaller mesh sizes. In addition, we apply the multilevel MC method [17] to approximate the loss function and the numerical result is recorded in Table 4.11. Again, slightly better results are obtained but the approximation of the variance is not good.



$s = 2, \epsilon = 0.25$ $s = 2, \epsilon = 0.25$

$s = 10, \epsilon = 0.05$ $s = 10, \epsilon = 0.05$

$s = 100, \epsilon = 0.005$ $s = 100, \epsilon = 0.005$

Fig. 4.4. 1D solution profiles of the stochastic Burgers' equation with different mesh sizes h.

Table 4.9: Expectation and variance errors of the proposed method for the stochastic Burgers' equation when the MC method is used with the batchsize 50000.

ϵ	s	h	L^2 error		L^1 error	
			Expectation	Variance	Expectation	Variance
0.25	2	1/40	2.44 e-3	1.27 e-1	1.63 e-03	8.66 e-02
0.25	2	1/80	3.57 e-3	8.82 e-2	2.06 e-03	5.77 e-02
0.25	2	1/160	9.70 e-3	1.14 e-1	4.42 e-03	7.40 e-02
0.25	2	1/320	2.28 e-2	2.21 e-1	1.10 e-02	1.46 e-01
0.1	5	1/40	4.16 e-3	2.30 e-1	2.03 e-03	1.60 e-01
0.1	5	1/80	2.44 e-3	1.24 e-1	1.34 e-03	9.78 e-02
0.1	5	1/160	4.34 e-3	9.16 e-2	2.17 e-03	8.04 e-02
0.1	5	1/320	1.61 e-2	2.21 e-1	8.10 e-03	1.75 e-01
0.05	10	1/40	6.79 e-3	3.37 e-1	2.99 e-03	2.40 e-01
0.05	10	1/80	2.25 e-3	1.86 e-1	1.13 e-03	1.45 e-01
0.05	10	1/160	4.68 e-3	1.27 e-1	2.28 e-03	1.17 e-01
0.05	10	1/320	2.01 e-2	3.36 e-1	8.94 e-03	2.74 e-01
0.01	50	1/40	1.80 e-2	6.42 e-1	5.36 e-03	5.01 e-01
0.01	50	1/80	5.74 e-3	4.04 e-1	1.67 e-03	3.32 e-01
0.01	50	1/160	3.09 e-3	2.69 e-1	1.18 e-03	2.68 e-01
0.01	50	1/320	4.40 e-2	9.12 e-1	1.70 e-02	8.06 e-01
0.005	100	1/40	2.58 e-2	7.53 e-1	6.54 e-03	5.01 e-01
0.005	100	1/80	8.64 e-3	5.25 e-1	2.09 e-03	3.32 e-01
0.005	100	1/160	1.95 e-3	3.76 e-1	7.22 e-04	2.68 e-01
0.005	100	1/320	3.07 e-2	9.47 e-1	5.65 e-03	8.06 e-01
0.0025	200	1/40	2.58 e-2	7.53 e-1	7.56 e-03	7.52 e-01
0.0025	200	1/80	8.64 e-3	5.25 e-1	2.51 e-03	5.68 e-01
0.0025	200	1/160	1.95 e-3	3.76 e-1	8.30 e-04	5.51 e-01
0.0025	200	1/320	3.07 e-2	9.47 e-1	3.52 e-03	9.09 e-01

Table 4.10: Expectation and variance errors of the proposed method for the stochastic Burgers' equation when the quasi-Monte Carlo method is used with the batchsize 50000.

ϵ	s	h	L^2 error		L^1 error	
			Expectation	Variance	Expectation	Variance
0.25	2	1/80	3.88 e-3	8.06 e-2	2.17 e-03	8.06 e-02
0.25	2	1/160	7.78 e-3	8.66 e-3	4.48 e-03	8.66 e-02
0.25	2	1/320	2.68 e-2	2.44 e-1	1.72 e-02	2.44 e-01
0.1	5	1/80	2.39 e-3	1.21 e-1	1.35 e-03	8.78 e-02
0.1	5	1/160	3.18 e-3	7.39 e-2	2.24 e-03	6.29 e-02
0.1	5	1/320	1.70 e-2	2.42 e-1	9.61 e-03	1.92 e-01
0.05	10	1/80	1.80 e-3	1.88 e-1	1.14 e-03	1.45 e-01
0.05	10	1/160	4.23 e-3	1.30 e-1	2.32 e-03	1.22 e-01
0.05	10	1/320	1.47 e-2	2.56 e-1	7.40 e-03	2.27 e-01
0.01	50	1/80	5.88 e-3	4.01 e-1	1.79 e-03	3,25 e-01
0.01	50	1/160	3.61 e-3	2.74 e-1	1.76 e-03	2.74 e-01
0.01	50	1/320	2.19 e-2	5.37 e-1	5.47 e-03	5.04 e-01
0.005	100	1/80	8.79 e-3	5.29 e-1	2.11 e-03	4.43 e-01
0.005	100	1/160	2.78 e-3	3.47 e-1	1.21 e-03	3.49 e-01
0.005	100	1/320	3.01 e-2	8.97 e-1	7.03 e-03	8.20 e-01

Table 4.11: Expectation and variance errors of the proposed method for the stochastic Burgers' equation when the multi-level MC method is used.

ϵ	s	h	L^2 error		L^1 error	
			Expectation	Variance	Expectation	Variance
0.01	50	1/80	5.82 e-3	4.03 e-1	2.11 e-03	3.27 e-01
0.01	50	1/160	2.37 e-3	2.36 e-1	9.99 e-04	2.18 e-01
0.01	50	1/320	8.37 e-3	2.76 e-1	3.42 e-03	2.80 e-01
0.005	100	1/80	8.82 e-3	5.28 e-1	2.20 e-03	4.41 e-01
0.005	100	1/160	2.07 e-3	3.53 e-1	7.66 e-04	3.51 e-01
0.005	100	1/320	3.03 e-2	8.72 e-1	5.92 e-03	7.96 e-01

5. Conclusions

In this work, based on the weak formulation of PDEs, we propose a deep learning based discontinuous Galerkin method (D2GM) to solve (stochastic) conservation laws. The main idea is that at the discrete level, the solution is smoother than that at the continuous level. By combining the advantages of discontinuous Galerkin method and deep neural networks, D2GM is able to solve problems with discontinuous solutions over the high-dimensional space. Convergence of the D2GM is proved under some assumptions. This method is tested for PDEs with non-smooth solutions over high-dimensional random space. Over some regime of mesh sizes, D2GM is found to be first-order and second-order accurate in practice. High-order schemes with discontinuous polynomial basis in space can be designed in the same manner. However, how to discretize the temporal derivative with high-order accuracy is unclear at the moment. For example, the leap-frog method is used together with the second-order scheme in space, but the overall second-order accuracy is not observed for the linear conservation law. Therefore, it will be of great interests to desgin high-order schemes for shock waves in the framework of deep neural networks. In summary, the proposed method shows a strong promise for solving high-dimensional uncertain PDEs with discontinuous solutions.

Acknowledgments. The work of J. Chen was supported by National Key R&D Program of China under grants 2018YFA0701700, 2018YFA0701701, and NSFC grant 11971021. The work of S. Jin was supported by the Natural Science Foundation of China under grant 12031013.

References

[1] R. Abgrall and S. Mishra, Uncertainty quantification for hyperbolic systems of conservation laws, in Handbook of numerical methods for hyperbolic problems, vol. 18 of Handb. Numer. Anal., Elsevier/North-Holland, Amsterdam, 2017, 507–544.

[2] H. Bijl, D. Lucor, S. Mishra, and C. Schwab, Uncertainty Quantification in Computational Fluid Dynamics, Springer, Cham, Switzerland, 2013.

[3] S. Boyd, N. Parikh, and E. Chu, Distributed optimization and statistical learning via the alternating direction method of multipliers, Now Publishers Inc, 2011.

[4] R.E. Caflisch, Monte Carlo and quasi-Monte Carlo methods, *Acta Numerica*, **1998** (1998), 1–49.

[5] G. Chavent and B. Cockburn, The local projection-discontinuous-Galerkin finite element method for scalar conservation laws, *ESAIM: Mathematical Modelling and Numerical Analysis*, **23** (1989), 565–592.

[6] J. Chen, R. Du, P. Li, and L. Lyu, Quasi-Monte Carlo sampling for solving partial differential equations by deep neural networks, *Numerical Mathematics: Theory Methods and Applications*, **14** (2021), 377–404.
[7] B. Cockburn, S. Hou, and C.W. Shu, The Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws. IV. the multidimensional case, *Mathematics of Computation*, **54** (1990), 545–581.
[8] B. Cockburn, G.E. Karniadakis, and C.W. Shu, Discontinuous Galerkin methods: theory, computation and applications, vol. 11, Springer Science & Business Media, 2012.
[9] B. Cockburn, S.Y. Lin, and C.W. Shu, TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws III: one-dimensional systems, *Journal of Computational Physics*, **84** (1989), 90–113.
[10] B. Cockburn and C.W. Shu, TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws. II. general framework, *Mathematics of Computation*, **52** (1989), 411–435.
[11] G. Cybenko, Approximation by superpositions of a sigmoidal function, Mathematics of Control, *Signals and Systems*, **2** (1989), 303–314.
[12] C. Dafermos, Hyperbolic Conservation Laws in Continuum Physics, Springer, 2016.
[13] W.E, Machine Learning and Computational Mathematics, *Communications in Computational Physics*, **28** (2020), 1639–1670.
[14] W.E, J. Han, and A. Jentzen. Algorithms for solving high dimensional PDEs: from nonlinear Monte Carlo to machine learning. Nonlinearity, **35**:1 (2021), 278.
[15] W.E and B. Yu, The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems, *Communications in Mathematics and Statistics*, **6** (2018), 1–12.
[16] K.I. Funahashi, On the approximate realization of continuous mappings by neural networks, **Neural networks**, **2** (1989), 183–192.
[17] J. Gopalakrishnan and G. Kanschat, A multilevel discontinuous Galerkin method, *Numerische Mathematik*, **95** (2003), 527–550.
[18] D. Gottlieb and D. Xiu, Galerkin method for wave equations with uncertain coefficients, *Communications in Computational Physics*, **3** (2008), 505–518.
[19] J.S. Hesthaven, Numerical methods for conservation laws, vol. 18 of Computational Science & Engineering, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2018. From analysis to algorithms.
[20] J. Hu, S. Jin, and D. Xiu, A stochastic Galerkin method for Hamilton–Jacobi equations with uncertainty, *SIAM Journal on Scientific Computing*, **37** (2015), A2246–A2269.
[21] H.J. Hwang, J.W. Jang, H. Jo, and J.Y. Lee, Trend to equilibrium for the kinetic Fokker-Planck equation via the neural network approach, *Journal of Computational Physics*, **419** (2020), 109665.
[22] S. Jin and Z. Ma, The discrete stochastic Galerkin method for hyperbolic equations with non-smooth and random coefficients, *Journal of Scientific Computing*, **74** (2018), 97–121.
[23] S. Jin and L. Pareschi, Uncertainty Quantification for Hyperbolic and Kinetic Equations, vol. 14, Springer, 2018.
[24] S. Jin, D. Xiu, and X. Zhu, A well-balanced stochastic Galerkin method for scalar hyperbolic balance laws with random inputs, *Journal of Scientific Computing*, **67** (2016), 1198–1218.
[25] A. Klöckner, T. Warburton, and J.S. Hesthaven, Viscous shock capturing in a time-explicit discontinuous Galerkin method, *Mathematical Modelling of Natural Phenomena*, **6** (2011), 57–83.
[26] R.J. LeVeque, Finite volume methods for hyperbolic problems, vol. 31, Cambridge University Press, 2002.
[27] S. Liang, L. Lyu, C. Wang, and H. Yang, Reproducing activation function for deep learning, arXiv Preprint arXiv:2101.04844, (2021).
[28] L. Lyu, Z, Zhang, M. Chen, and J. Chen. MIM: A deep mixed residual method for solving high-order partial differential equations. *Journal of Computational Physics*, **452** (2022): 110930.

[29] X. Meng, Z. Li, D. Zhang, and G.E. Karniadakis, PPINN: Parareal physics-informed neural network for time-dependent PDEs, *Computer Methods in Applied Mechanics and Engineering*, **370** (2020), 113250.

[30] G. Poëtte, B. Després, and D. Lucor, Uncertainty quantification for systems of conservation laws, *Journal of Computational Physics*, **228** (2009), 2443–2467.

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

[32] J. Sirignano and K. Spiliopoulos, DGM: A deep learning algorithm for solving partial differential equations, *Journal of Computational Physics*, **375** (2018), 1339–1364.

[33] T. Tang and T. Zhou, Convergence analysis for stochastic collocation methods to scalar hyperbolic equations with a random wave speed, *Communications in Computational Physics*, **8** (2010), 226–248.

[34] D. Xiu, Numerical methods for stochastic computations: a spectral method approach, Princeton University Press, 2010.

[35] Y. Zang, G. Bao, X. Ye, and H. Zhou, Weak adversarial networks for high-dimensional partial differential equations, *Journal of Computational Physics*, **411** (2020), 109409.

[36] D. Zhang, L. Lu, L. Guo, and G.E. Karniadakis, Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems, *Journal of Computational Physics*, **397** (2019), 108850.