Advances in Applied Mathematics and Mechanics
Adv. Appl. Math. Mech., Vol. **xx**, No. x, pp. 1-27

DOI: 10.4208/aamm.OA-2022-0278
xxx 2024

Cell-Average Based Neural Network Method for Hunter-Saxton Equations

Chunjie Zhang[1], Changxin Qiu[1,*], Xiaofang Zhou[1] and Xiaoming He[2]

[1] *School of Mathematics and Statistics, Ningbo University, Ningbo, Zhejiang 315211, China*
[2] *Department of Mathematics and Statistics, Missouri University of Science and Technology, Rolla, MO 65409, USA*

Received 1 November 2022; Accepted (in revised version) 3 July 2023

Abstract. In this paper, we develop a cell-average based neural network (CANN) method for solving the Hunter-Saxton equation with its zero-viscosity and zero-dispersion limits. Motivated from the finite volume schemes, the cell-average based neural network method is constructed based on the finite volume integrals of the original PDEs. Supervised training is designed to learn the solution average difference between two neighboring time steps. The training data set is generated by the cell average based on a single initial value of the given PDE. The training process employs multiple time levels of cell averages to maintain stability and control temporal accumulation errors. After being well trained based on appropriate meshes, this method can be utilized like a regular explicit finite volume method to evolve the solution under large time steps. Furthermore, it can be applied to solve different type of initial value problems without retraining the neural network. In order to validate the capability and robustness of the CANN method, we also utilize it to deal with the corrupted learning data which is generated from the Gaussian white noise. Several numerical examples of different types of Hunter-Saxton equations are presented to demonstrate the effectiveness, accuracy, capability, and robustness of the proposed method.

AMS subject classifications: 35E15, 65M15, 68T07
Key words: Finite volume scheme, cell-average based neural network, Hunter-Saxton equation, corruption data.

1 Introduction

The Hunter-Saxton (HS) equation [1,2] is a nonlinear wave equation which has been used to describe waves in a massive director field that propagate in nematic liquid crystals

*Corresponding author.
Emails: 2111071049@nbu.edu.cn (C. Zhang), qiuchangxin@nbu.edu.cn (C. Qiu), 2111071051@nbu.edu.cn (X. Zhou), hex@mst.edu (X. He)

when certain molecules move and cause interference. The HS equation under consideration here is given as

$$u_{xxt}+2u_xu_{xx}+uu_{xxx}=0. \tag{1.1}$$

If we consider the viscosity and dispersion [3,4], we have the corresponding regularization with viscosity:

$$u_{xxt}+2u_xu_{xx}+uu_{xxx}-\varepsilon_1u_{xxxx}=0, \tag{1.2}$$

and the corresponding regularization with dispersion:

$$u_{xxt}+2u_xu_{xx}+uu_{xxx}-\varepsilon_2u_{xxxxx}=0, \tag{1.3}$$

where $\varepsilon_1\geq 0$ and ε_2 are small constants. In the past two decades, this equation has attracted extensive attention because of its rich mathematical structure and properties. Therefore it is not surprising that many different numerical methods have been proposed and analyzed for the HS equation, including the finite difference method [5], local discontinuous Galerkin (LDG) method [6,7], collocation method [8–10], collocation finite element method [11], quasilinearization method [12,13] and others [14–16]. However, compared with the extensively studied classical numerical methods, the HS equation is still in great need of efforts for developing and analyzing the stable, accurate, and efficient numerical neural network methods, especially for peakon solutions.

Recently, there has been increasing interest in developing neural networks to solve partial differential equations (see, e.g., [17–30]). Neural networks generate a wide range of functions by combining linear transformations and activation functions. One of the most notable characteristics of neural networks is that they do not require a hand-crafted geometric mesh or point cloud, as do the traditional, well-studied finite difference, finite volume, and finite element methods. According to their basic technique and core goal, the neural network methods for solving PDEs can be roughly divided into two categories.

One group is to design neural network methods and apply the neural network methods to solve many types of partial differential equations. These neural network methods take x and t as the network input vectors, and have the advantages of automatic differentiation and mesh free, including the early work [31], the popular physics-informed neural network (PINN) methods [26,32–34], and many others [22,35–38]. Moreover, PIELM [39] is proposed to improve the speed of PINN in a larger domain for practical problems. In [40], PINNs are used to directly encode the control equations into the deep neural network through automatic differentiation to overcome the limitations of incompressible laminar flow and turbulence. In [41–43], weak formulations are applied in the loss function, instead of the PDEs explicitly enforced on collocation points. We further mention the works of [44] and [45], for which method of lines are explored with Fourier basis and Residual networks applied to evolve the dynamical system.

The second group is to apply neural network to improve the existing numerical methods. In [46], a multilayer perceptron (MLP) is constructed to identify troubled-cells. In [47], the authors use (deep) Reinforcement learning to learn the new solvers for conservation laws. In [48], an MLP network is designed to estimate the artificial viscosity in

the discontinuous Galerkin schemes. Researchers also developed WENO schemes augmented with convolution networks for shock detection in [49] and DG methods with convolution network for strong shock detection in [50]. Recently, the work [51] also developed an estimator based on TVB constants for DG methods by constructing a MLP model. Considering the vigorous development of neural network methods for PDEs, the Hunter-Saxton equation still needs significant efforts to develop and analyze the stable, accurate, and efficient numerical neural network methods.

In this paper, we propose a cell-average based neural network (CANN) method [52, 53], which is motivated from finite volume method, to solve the Hunter-Saxton equation. The CANN method follows the solution properties and characteristics of the PDEs to build up neural network solvers. The main idea is to handle all spatial variable related differentiation and integration approximation by the neural network. In adjacent time levels, the features of the current time level can be accurately captured by the CANN method based on the solution of the previous time level. In a word, through intensive training we force the neural network to learn the solution average evolution between two neighboring time steps. An interesting aspect of CANN method is the fact that it is mesh dependent and a local solver due to its base on the finite volume scheme. With the CANN method no effort is required for the differentiation terms related to the spatial variable x and we have no concerns about the choice of numerical flux scheme.

Once being well trained, the CANN method can be implemented like an explicit finite volume scheme and possess several advantages and nice properties. First, the CANN method can be relieved from the explicit scheme CFL restriction, and can adapt large time step size to evolve the solutions of the HS equation. Second, once being well trained for one initial condition, the CANN method can work well for a group of initial conditions for HS equation without retraining the neural network. Third, the CANN method works well for the corrupted data or low quality data generated by white noise for applications in real world. The predictions of the CANN model are fairly robust against data noise.

The rest of this article is organized as follows. In Section 2, we introduce motivation of CANN method and emphasize the learning data setup and the training process. Then, we provide numerical experiments to validate the proposed method and illustrate features and capability of the method in Section 3. Finally, we will summarize the conclusion in Section 4.

2 Cell-average based neural network (CANN) method

2.1 Problem setup, motivation and cell-averaged neural network method

We consider using the cell-averaged neural network (CANN) method to solve partial differential equations (PDEs):

$$u_t = \mathcal{L}(u), \quad (x,t) \in [a,b] \times \Re^+. \tag{2.1}$$

Here t and x denote the temporal and spatial variables and $[a,b]$ is the spatial domain. The differential operator $\mathcal{L}$ represents the commonly used second order, third and fourth order differential operators, such as $\mathcal{L}(u)=u_{xxx}$ or $\mathcal{L}(u)=-u_{xxxx}$ for the high order PDEs. For the Hunter-Saxton equation, by denoting $q=u_{xx}$, we have

$$\begin{cases} q_t=\mathcal{L}(u), \quad (x,t)\in\Omega\times(0,T], \\ \mathcal{L}(u)=\dfrac{1}{2}((u_x)^2)_x-\dfrac{1}{2}(u^2)_{xxx}+\epsilon_1 u_{xxxx}+\epsilon_2 u_{xxxxx}. \end{cases} \tag{2.2}$$

When $\epsilon_1=\epsilon_2=0$, (2.2) is the HS equation (1.1). When $\epsilon_1>0$, $\epsilon_2=0$, (2.2) is the regularization with viscosity of the HS equation (1.2). When $\epsilon_1=0$, $\epsilon_2\neq 0$, (2.2) is the regularization with dispersion of the HS equation (1.3).

The cell-average based neural network method is grid dependent and driven by the finite volume method. Traditional numerical methods will be served as the guideline to construct the deep learning method. Divide $[a,b]$ evenly into J cells, and $\Delta x=\frac{b-a}{J}$ is the cell size, where $x_{1/2}=a$, $x_{J+1/2}=b$. Then $[a,b]=\bigcup_{j=1}^{J} I_j$ with $I_j=[x_{j-1/2},x_{j+1/2}]$ as one computational cell. In addition, the time domain is evenly divided so that $t_n=n\times\Delta t$ and $t_0=0$. After enough training, the method can be used for the purpose of a regular meticulous volume scheme for solving PDEs (2.1).

In the traditional finite volume method, the partial differential equation (2.1) is integrated over the unit I_j in space and the sub interval $[t_n,t_{n+1}]$ in time. Then we have

$$\int_{t_n}^{t_{n+1}}\int_{I_j} u_t dxdt=\int_{t_n}^{t_{n+1}}\int_{I_j}\mathcal{L}(u)dxdt. \tag{2.3}$$

According to the definition of cell average $\bar{u}_j(t)=\frac{1}{\Delta x}\int_{I_j}u(x,t)dx$, (2.3) leads to

$$\bar{u}_j(t_{n+1})-\bar{u}_j(t_n)=\frac{1}{\Delta x}\int_{t_n}^{t_{n+1}}\int_{I_j}\mathcal{L}(u)dxdt. \tag{2.4}$$

The equation above is the integral format as our starting point to design our neural network method. The idea of cell-average based neural network method is to apply a neural network solver $\mathcal{N}(\cdot;\Theta)$ to approximate the right hand side of (2.4):

$$\mathcal{N}(\cdot;\Theta)\approx\frac{1}{\Delta x}\int_{t_n}^{t_{n+1}}\int_{I_j}\mathcal{L}(u)dxdt, \tag{2.5}$$

where Θ denotes the network parameter set of all weight matrices and biases. Then we have the neural network format for solving the PDEs at next time level t_{n+1}:

$$\bar{v}_j^{out}=\bar{v}_j^{in}+\mathcal{N}(\overrightarrow{V}_j^{in};\Theta). \tag{2.6}$$

Comparing (2.6) and (2.4) and denoting $\bar{v}_j^{in}=\bar{u}_j(t_n)$ and $\bar{v}_j^{out}\approx\bar{u}_j^{n+1}$, the format of the CANN method to approximate the solution average $\bar{u}_j^{n+1}$ at next time level t_{n+1} can be obtained:

$$\bar{u}_j^{n+1}=\bar{u}_j^{n}+\mathcal{N}(\overrightarrow{V}_j^{in};\Theta). \tag{2.7}$$

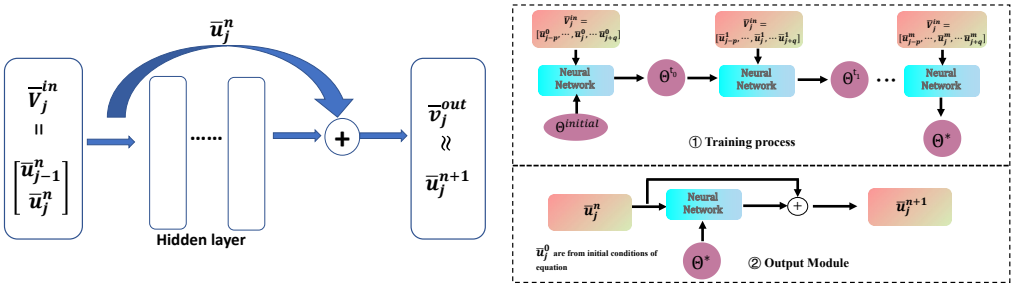

Figure 1: Illustration of cell-average based neural network method: left is one time level solution averages $m=0$ for training [52] and right is multiple time levels ($m>0$) of solution averages for training.

Here $\overrightarrow{V}_j^{in}$ is the training data set corresponding to given the solution averages $\{\bar{u}_j^n\}$ at time level t_n. It is an important component that needs to be carefully selected, as shown in Fig. 1. Its general format is defined as follows:

$$\overrightarrow{V}_j^{in} = \left[\bar{u}_{j-p}^n, \cdots, \bar{u}_{j-1}^n, \bar{u}_j^n, \bar{u}_{j+1}^n, \cdots, \bar{u}_{j+q}^n\right]^T, \tag{2.8}$$

where we include p cell averages to the left of $\bar{u}_j^n$ and q cell averages to the right of $\bar{u}_j^n$ in the definition of the input vector. For example, for some linear PDEs, we can use the simple architecture with $p=1$ and $q=0$, see the left figure of Fig. 1. However, for nonlinear PDEs, we always need to choose more complicated architecture with $p\geq 1$ and $q\geq 1$. Note that for $j=1$ or $j=J$ or those close to boundary cells, in this paper we apply the exact solution averages for ghost cells outside the domain to implement boundary conditions. The suitable stencil or the p and q values in (2.8) determine the effectiveness of the neural network method approximating the solution average $\bar{u}_j^{n+1}$ at the next time level.

In this paper, we consider a standard fully connected neural network with M ($M\geq 3$) layers. The first layer is the input vector, the last layer is the output vector, and there are $(M-2)$ hidden layers in the middle. Therefore, the minimum structure of neural network has 3 layers, i.e., $M=3$. The number of neurons in each layer is expressed by n_i ($i=1,\ldots,M$). The first layer takes $n_1=p+q+1$ as the dimension, and the last layer takes $n_M=1$ as the dimension. The abstract goal of machine learning is to find a function $\mathcal{N}$: $R^{p+q+1}\to R^1$, so that $\mathcal{N}(\cdot;\Theta)$ accurately approximates $\frac{1}{\Delta x}\int_{t_n}^{t_{n+1}}\int_{I_j}\mathcal{L}(u)dxdt$. The optimal parameter set of network $\mathcal{N}(\cdot;\Theta)$ is obtained by the training on the given data set Θ.

Our major contribution is to explore such a network structure that exactly matches an explicit one-step finite volume scheme. Thus the network parameter set, after offline supervised learning from a given data set, behaves as the coefficients of the scheme. After being well trained, the CANN method is a local and mesh dependent solver that can be applied to solve the equation like a regular explicit method. Then, we have the following definition.

Definition 2.1 ([52]). *A cell-average based neural network method is uniquely determined by the following four components: (1) the choice of spatial mesh size Δx; (2) the choice of time step size Δt; (3) the choice of network learning data including the input vector $\overrightarrow{V}_j^{in}$ of (2.8) and the target data set; and (4) the number of hidden layers and neurons per layer in the corresponding structure of the neural network. With the optimal weights and biases Θ^*, the cell-average based neural network method can be implemented as a regular explicit finite volume scheme*

$$\bar{v}_j^{n+1} = \bar{v}_j^n + \mathcal{N}(\overrightarrow{V}_j^n; \Theta^*), \quad \forall j = 1, \cdots, J, \quad \forall n = 0,1,2,\cdots. \tag{2.9}$$

2.2 Training process

In this section, we discuss how to train the network to obtain the optimal parameter set Θ^* so that the neural network (2.6) can accurately approximate the mean evolution of solutions $\bar{u}_j^n \to \bar{u}_j^{n+1}$. Considering Definition 2.1, the learning data set plays a key role in the whole training process. Training data sets are usually generated from initial conditions or given data at t_n while target data sets usually come from time level solution averages corresponding to $t \geq t_n$. Even though we select one trajectory as the target data, the well trained CANN can still be applied to other different trajectories corresponding to different initial conditions.

The learning data is collected in pairs, with each pair representing the average of the solutions at two adjacent time levels. The notation of the training data set is

$$S = \left\{ \left(\bar{u}_j^n\right), \bar{u}_j^{n+1}, j = 1, \ldots, J \right\}_{n=0}^{m}, \tag{2.10}$$

where the training data $(\bar{u}_j^n)$ is generated from solution averages of given data (or initial conditions with $n = 0$). The target data $\bar{u}_j^{n+1}$ is solution average obtained from observed data collection of real application problems or other numerical methods for the PDEs. We emphasize that the training data pair is the mean of the solutions collected in the spatial domain and from the time level t_0 to t_{m+1}.

In order to simplify the discussion of the method, this paper focuses on the corresponding approximation for each fitting problem on a network based on cell average. When the fitting problem (2.1) is given different initial value $u^i(x,0) = u_0^i(x)$ (i means the different initial conditions and boundary conditions), learning data sets will be collected from one trajectory of different initial values and boundary conditions to train the network. After enough training, it will be implemented like an explicit one-step finite volume format for solving other different initial values, without retraining the network. It should be pointed out that $u^i(x,0) = u_0^i(x)$ denotes different initial conditions under similar types. For example, if the trajectory data of $u^1(x,0) = u_0^1(x)$ is collected to train the network, then the well trained CANN can still be used to solve different initial value problems with trajectory $u^i(x,0) = u_0^i(x)$, $(i = 2,3,\cdots)$.

In addition, in order to test the robustness of CANN, we use low-quality data as learning data to train neural networks. Now let's call the noisy learning data set

$$S_N=\left\{\left(\bar{u}_j^n+\omega_j\right),\ \bar{u}_j^{n+1}+\xi_j,\ j=1,\ldots,J\right\}_{n=0}^{m}, \tag{2.11}$$

where ω_j and ξ_j are Gaussian white noises obtained from standard normal distribution.

The network output value $\bar{v}_j^{out}$ is obtained by applying $\bar{v}_j^{in}=\bar{u}_j^n$ in (2.6) and compared with the target value $\bar{u}_j^{n+1}$. And then it loops between the data set S of (2.10) (or S_N of (2.11)) to minimize the error or square loss function

$$L_{j,t_n}(\Theta)=(\bar{v}_j^{out}-\bar{u}_j^{n+1})^2, \tag{2.12}$$

for all $j=1,\ldots,J$ and for all $n=0,\cdots,m$. This loss function defined on a single data pair is called stochastic or approximate gradient descent.

Specifically for the last time level pair (t_m,t_{m+1}), we record the squared L_2 error defined below corresponding to iteration

$$L_2^2(t_{m+1})=\sum_{j=1}^{J}\Delta x L_{j,t_m}(\Theta)=\sum_{j=1}^{J}(\bar{v}_j^{out}-\bar{u}_j^{m+1})^2\Delta x. \tag{2.13}$$

We output the squared L_2 error of (2.13) corresponding to iteration index $i=1,\cdots,K$ to demonstrate the effectiveness of cell-average based neural network method.

Next we conclude the section with comments on the minimum size of training data set S or S_N, which are purely lab results observed from numerical tests.

Remark 2.1. For linear partial differential equations, one time level solution averages in the learning data set S or S_N, which corresponds to (t_0,t_1) or $m=0$ in (2.10), are sufficient for obtaining an effective neural network, see Fig. 1. For nonlinear partial differential equations, it is necessary to include multiple time levels ($m>0$) of solution averages in the training set S to make sure the neural network learns the evolution mechanism successfully, see Fig. 1.

Remark 2.2. Once one cell-average based neural network is well trained and available, it can be applied to solve the Hunter-Saxton equation associated with different initials and over different domains.

2.3 Implementation and summary of cell-average based neural network method

In this section, we discuss the detail of our CANN learning algorithm. The general procedures including the data collection are generated in Algorithm 2.1.

Using the spatial step size Δx, the temporal step size Δt, the previously selected network input vector

$$\overrightarrow{V}_j^n=\left[\bar{v}_{j-p}^n,\cdots,\bar{v}_{j-1}^n,\bar{v}_j^n,\bar{v}_{j+1}^n,\cdots,\bar{v}_{j+q}^n\right]^T,$$

Algorithm 2.1 Training process of CANN.

Require: Collect data set S: (S_N can be collected similarly)

training data set: $\bar{u}_j^m$ from given data corresponding to $t_0, t_1, \ldots, t_m$,

target data set: $\bar{u}_j^{m+1}$ from given data or exact values at t_{m+1}.

Ensure: CANN: $\bar{v}_j^{out} = \bar{v}_j^{in} + \mathcal{N}(\overrightarrow{V}_j^{in}; \Theta)$

$\bar{v}_j^{in} \leftarrow \bar{u}_j^n$

$\overrightarrow{V}_j^{in} \leftarrow \left[\bar{u}_{j-p}^n, \cdots, \bar{u}_{j-1}^n, \bar{u}_j^n, \bar{u}_{j+1}^n, \cdots, \bar{u}_{j+q}^n\right]^T$

$\Theta \leftarrow$ initial random values

while loss function values $\leq$ tolerance **do**

  $\bar{v}_j^{out} \leftarrow \bar{v}_j^{in} + \mathcal{N}(\overrightarrow{V}_j^{in}; \Theta)$

  $L_{j,t_n}(\Theta) \leftarrow (\bar{v}_j^{out} - \bar{u}_j^{n+1})^2$

  update $\Theta^* \leftarrow$ Stochastic Gradient Descent

end while

Output: $\bar{v}_j^{n+1} = \bar{v}_j^n + \mathcal{N}(\overrightarrow{V}_j^n; \Theta^*)$, $j = 1,2,3,\ldots$

return $\bar{u}_j^{n+1} \leftarrow \bar{v}_j^{out}$

Ensure: Apply $\bar{v}_j^{n+1} = \bar{v}_j^n + \mathcal{N}(\overrightarrow{V}_j^n; \Theta^*)$ to solve different initial value problems with $u^s(x,0)$.

and the obtained optimal weight and deviation Θ^*, we have a complete definition of the neural network method. Then it is implemented as a regular explicit finite volume scheme as below

$$\bar{v}_j^{n+1} = \bar{v}_j^n + \mathcal{N}(\overrightarrow{V}_j^n; \Theta^*), \quad \forall j = 1, \cdots, J, \quad \forall n = 0,1,2,\cdots. \tag{2.14}$$

We now discuss about the advantages of the proposed CANN method.

First, a surprising result is that cell-average based neural network approach can get relief of the CFL constraint on the time step. Classical numerical methods require time steps as small as $\Delta t \approx (\Delta x)^2$. The results show that the CANN method can be adapted to a stable method by using large time step (e.g., $\Delta t = 2\Delta x$) for various Δx. Once being trained well, the CANN method can be used as an explicit method to advance resolution efficiently and accurately.

Second, for cell-average based neural network method, we use the known data for the data in the ghost cell such that we can apply it for different initial conditions and boundary conditions easily, without retraining the neural network. A roughly generic template allows evolution from one current unit to the next, with a previously known unit solving the next unknown unit.

Third, to understand the power and robustness of the CANN method, we apply it to noise learning that deals with low-quality learning data or Gaussian noise from related application problems. Numerical experiments show that the CANN method is fairly

robust against data noise. The CANN method works well for the corrupted data and can capture the main structure of the wave propagation well in the approximation without retraining the neural network, even for relatively large level corrupted data with a normal distribution of $0.05\times N(0,1)$.

Remark 2.3. How to properly select the number of hidden layers and the number of neurons in each layer has not been discovered yet. In this paper, after a lot of practice, we found that a small number of hidden layers (less than 5 hidden layers) is conducive for the implementation, and the numerical results are very good. Thus, this CANN is also a kind of light neural network.

3 Numerical examples

In this section, we will show the numerical experiments for different types of Hunter-Saxton equations. Our primary focus is to demonstrate the accuracy of cell-average based neural network methods for numerical approximations. Then, considering the generalization ability of neural network, we carry out the well trained CANN solver to deal with different initial conditions directly without retraining the neural network. We also conduct corrupted data or low quality data which are generated from the noise to validate the capability of the CANN method.

In this section, we denote T as the final time where we compute the L_2 and L_∞ errors. As mentioned previously, the time step size Δt is chosen as one part of the definition of a neural network method (2.14). Thus we always have $T=N_t\Delta t$, where N_t is the total numbers of time step. Below we list the L_2 and L_∞ errors formula, which are the same as the ones for finite volume method.

$$Error_{L_2}(T)=\sqrt{\sum_{j=1}^{J}(\bar{v}_j^{out}(T)-\bar{u}_j(T))^2\Delta x}, \tag{3.1a}$$

$$Error_{L_\infty}(T)=\max_{j=1}^{J}|\bar{v}_j^{out}(T)-\bar{u}_j(T)|, \tag{3.1b}$$

where $\bar{v}_j^{out}(T)$ denote the solution of the cell-average based neural network method (2.14) on cell j and at final time T. We have $\bar{u}_j(T)$ denote either the exact solution average or the reference solution average on cell j and at time T. In the numerical tests, it can be approximately obtained from a highly accurate numerical method.

As marked down in Definition 2.1, four components of Δx, Δt, network input vector $\overrightarrow{V}_j^{in}$ and the structure of neural network (number of hidden layers and neurons per layer) together identify one neural network solver (2.14). We highlight that for HS equations, all numerical examples in this section apply multiple time level data pairs spread over the spatial domain for training, with J in (2.10) as the total number of partition over spatial domain, see Remark 2.1. We also mention that the square of L_2 error is used as training stopping condition, with which well trained network errors are around 10^{-6} or smaller.

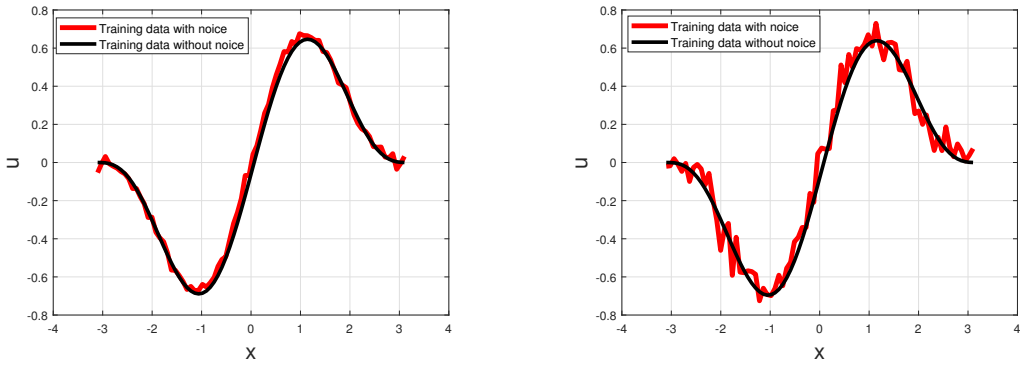

Figure 2: Noisy learning data with $\pm 2\%$ (left) and $\pm 5\%$ (right).

The noisy learning data set is denoted as (2.11) and ω_j and ξ_j are Gaussian white noise which are drawn from a standard normal distribution. Over the section we consider two cases with $\eta = 0.02$ and $\eta = 0.05$ (e.g., Fig. 2), which respectively correspond to $\pm 2\%$ and $\pm 5\%$ relative noises in all data.

3.1 Example 1

In this section, we consider the numerical simulation for the HS equation

$$u_{xxt} + 2u_x u_{xx} + u u_{xxx} = 0 \tag{3.2}$$

with the initial condition

$$u(x,0) = 0.01(x-\pi)^2(x+\pi)^2 \sin(x). \tag{3.3}$$

The exact solution of the HS equation (3.2) is

$$u(x,t) = 0.01(x-\pi)^2(x+\pi)^2 \sin(x-ct). \tag{3.4}$$

The computational domain is $D = [-\pi, \pi]$. Because different c will yield different initial values or exact solutions, we set the case of $c = 1$ as the foundational test. The other cases will be considered in later tests for different initial conditions.

Accuracy tests with $c = 1$. In this experiment, we test the accuracy of our CANN method. In the training process, the learning data set (2.10) is generated from the solution trajectory of (3.4) with $c = 1$. By setting $\Delta x = 0.0785$ and $\Delta t = \frac{1}{2}\Delta x$, we pick the neural network which has 1 hidden layer with 8 neurons for the training. The essential input vector for CANN method can be generated by

$$\overrightarrow{V}_j^{in} = \left[\bar{u}_{j-7}^n, \bar{u}_{j-6}^n, \bar{u}_{j-5}^n, \bar{u}_{j-4}^n, \bar{u}_{j-3}^n, \bar{u}_{j-2}^n, \bar{u}_{j-1}^n, \bar{u}_j^n, \bar{u}_{j+1}^n, \bar{u}_{j+2}^n, \bar{u}_{j+3}^n\right]^T. \tag{3.5}$$

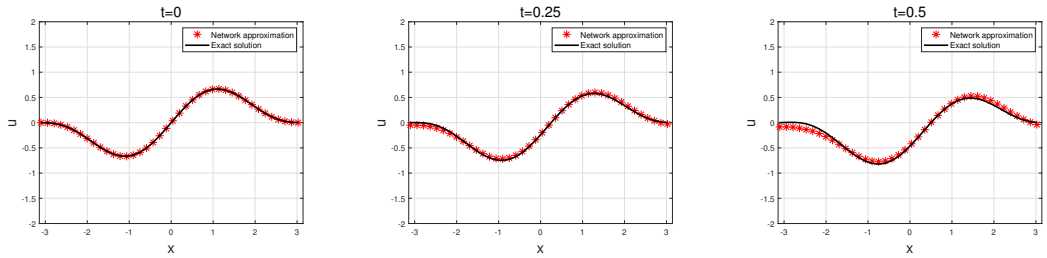

Figure 3: Trajectory simulation with $c=1$ and $\Delta t=\frac{1}{2}\Delta x$.

Network training is conducted for up to $k=10^4$ iterations. After enough training, the neural network is applied to solve the HS equation as a fixed numerical scheme up to $T=0.5$. Screen shots of $t=0$, $t=0.25$, and $t=0.5$ are shown in Fig. 3. We can found that the wave profile can be well approximated and the large time step size works well for the CANN method.

Additionally, we also test the accuracy of large time step size for our CANN method with the fixed spatial mesh size $\Delta x=0.0785$ and different temporal mesh size $\Delta t=\frac{1}{3}\Delta x,\frac{1}{2}\Delta x,\Delta x,\frac{8}{5}\Delta x$ correspondingly. L_2 and L_∞ errors are computed and listed in Table 1. We can find that even for large time step size, our method can obtain the similar errors around 10^{-3} for L_2 and 10^{-1} for L_∞ in the simulation. We further report that our CANN method can explicitly solve the HS equation with $\frac{8}{5}\Delta x$, which is bigger than the time step size of $\Delta t\leq\Delta x^2$ from the conventional CFL restriction.

Different trajectories approximation. Next, we apply CANN method to solve Eq. (3.2) under different initial conditions. Different initial conditions are obtained by applying different c in the exact solution (3.4). There is no need to retrain the CANN solver such that we can use it to solve the HS equation with different initial conditions directly.

In this paper, we choose the well trained neural network, which is obtained from the trajectory with $c=1$, to solve the different initial conditions with $c=0.1$ and $c=1.5$. As a fixed numerical scheme, the CANN solver does not need to change the architecture. However, in order to match the different initial conditions with parameter c, we still need to change the size of the time step $\Delta t=\frac{16}{5}\Delta x$ for $c=0.1$ and $\Delta t=\frac{1}{3}\Delta x$ for $c=1.5$ in the simulation process. From Fig. 4 and Fig. 5, we can clearly see that the CANN method

Table 1: Errors of neural network methods for Example 3.4 for $\Delta x=0.0785$ fixed with varying Δt.

Δt	L_2	L_∞
$1/3\Delta x$	5.7000e-03	1.110e-01
$1/2\Delta x$	5.6000e-03	1.104e-01
Δx	5.7000e-03	1.1160e-01
$8/5\Delta x$	5.7000e-03	1.1130e-01

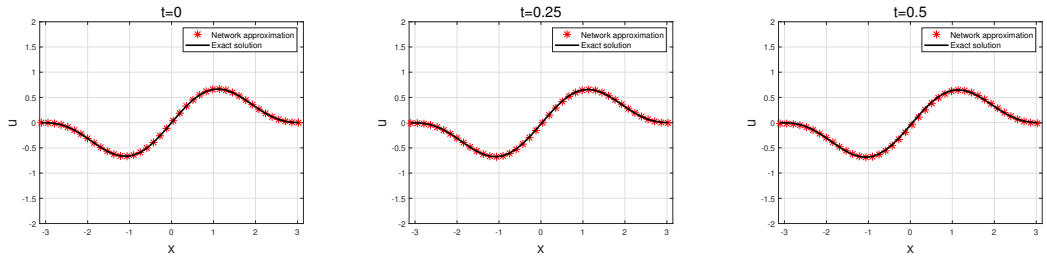

Figure 4: Apply optimal CANN to solve different initials with $c=0.1$ and $\Delta t=\frac{16}{5}\Delta x$.

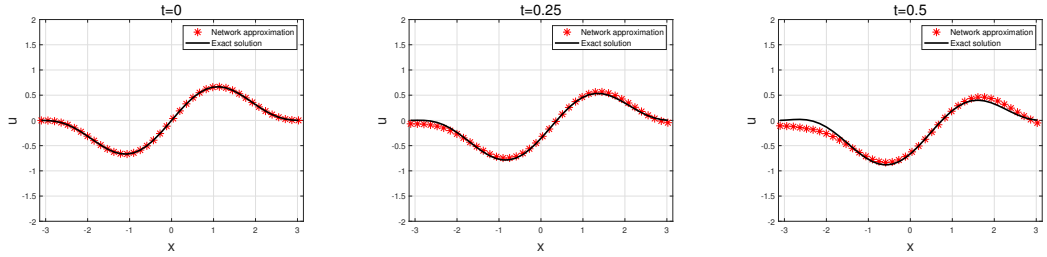

Figure 5: Apply optimal CANN to solve different initials with $c=1.5$ and $\Delta t=\frac{1}{3}\Delta x$.

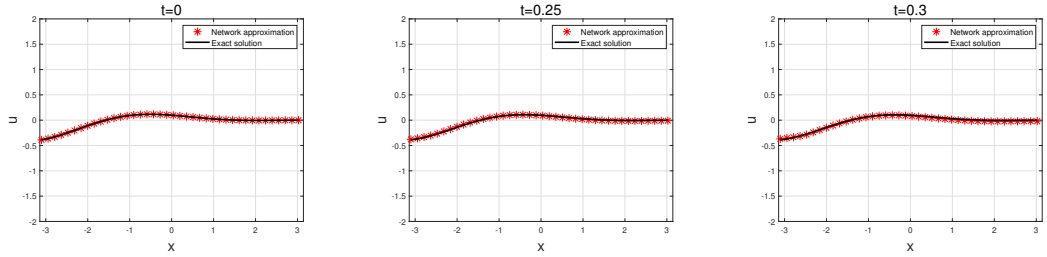

Figure 6: Apply optimal CANN to approximate different solution trajectory with $u(x,t)=0.01(x-\pi)^2\cos(x-\frac{1}{2}t)$.

is able to accurately capture a decent collection of waves propagation for HS equation. We also highlight that the smaller the parameter c is, the better approximation of the propagation of motion wave can be obtained by CANN solver without retraining.

Furthermore, we also test the cases whose solution trajectories are different. We utilize the well trained neural network, which is obtained from the trajectory with $c=1$ in Eq. (3.4), to approximate the different solution trajectories with

$$u(x,t)=0.01(x-\pi)^2\cos\left(x-\frac{1}{2}t\right) \quad \text{and} \quad u(x,t)=\tan\left(\frac{2}{5}(x-ct)\right).$$

In Fig. 6, we set $\Delta x=\pi/40$, $\Delta t=2\Delta x$ and obtain the L_2 error 1.70×10^{-3} and the L_∞ error 2.07×10^{-2}. In Fig. 7, we have the L_2 error 1.90×10^{-3} and the L_∞ error 5.62×10^{-2} by using the same setting of Δx and Δt. This indicates that our CANN method has a good

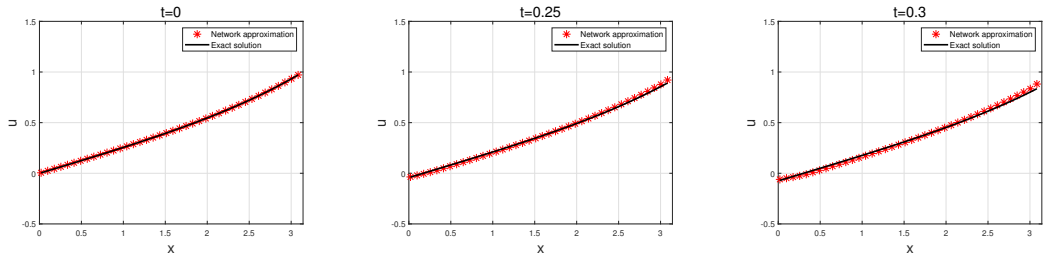

Figure 7: Apply optimal CANN to approximate different solution trajectory with $u(x,t)=\tan(\frac{2}{5}(x-t))$.

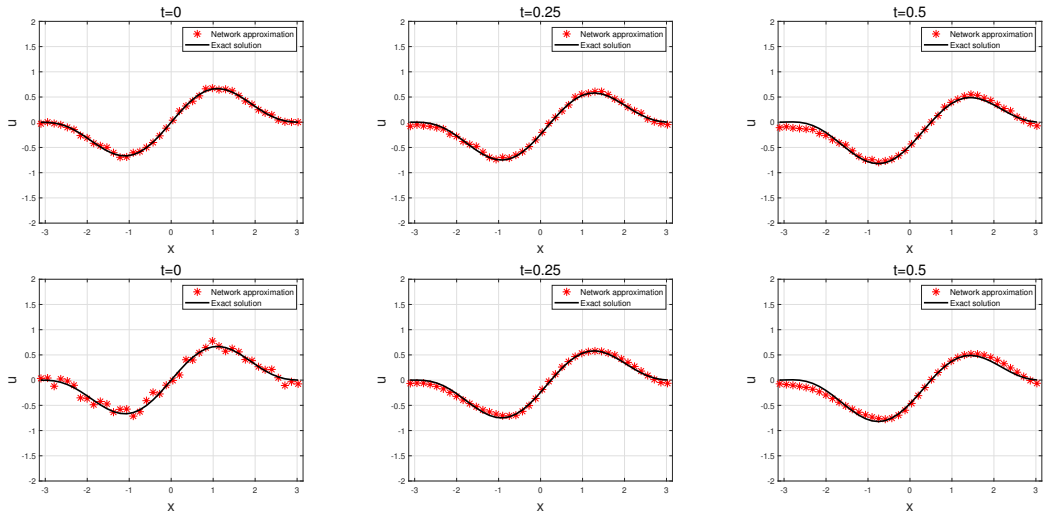

Figure 8: Trajectory simulation with corruption learning data by $\eta=0.02$ (top) and $\eta=0.05$ (bottom).

capability for generalization, allowing it to obtain a good numerical approximation of the different solution trajectories for the HS equation without retraining the network.

Low quality data. In order to test the capability of our CANN solver, we also utilize it to deal with the corrupted data generated from (2.11) with multiplicative factor $\eta=0.02$ and $\eta=0.05$, which respectively correspond to $\pm2\%$ and $\pm5\%$ relative noises in all data. An interesting fact of this part is that we still apply the CANN solver associated with $c=1$ from the accuracy test part and do not need to change the architecture any more.

By applying the CANN solver in the approximation for $\eta=0.02$ and $\eta=0.05$, we have the L_2 and the L_∞ errors around 7.3×10^{-3} and 1.375×10^{-1} with meshes of $\Delta x=0.0785$ and $\Delta t=\frac{4}{5}\Delta x$. Fig. 8 shows that the CANN method is fairly robust against data noise and can well capture the main structure of the wave propagation in the approximation without retraining the neural network.

In addition, we also try to simulate the solution with high frequency noise perturbation of $\eta=0.2$ and $\eta=0.5$. From Fig. 9 we can observe that the main structure of the wave

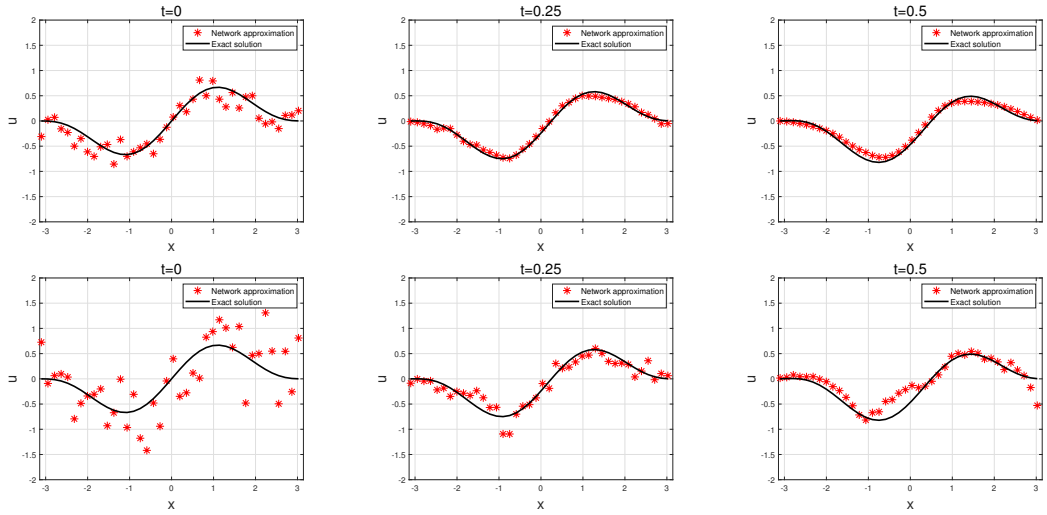

Figure 9: Trajectory simulation with corruption learning data by $\eta=0.2$ (top) and $\eta=0.5$ (bottom).

propagation captured by the CANN method is acceptable.

3.2 Example 2

In this example, we consider the regularization with viscosity of the HS equation

$$u_{xxt}+2u_xu_{xx}+uu_{xxx}-\varepsilon_1u_{xxxx}=0, \tag{3.6}$$

where $\varepsilon_1\geq 0$ is a constant. Here, we consider the zero-viscosity limit of (3.6) (i.e., zero-viscosity limit $\varepsilon_1=0$) with the initial data

$$u(x,0)=\begin{cases} 0, & x\leq 0,\\ x, & 0<x<1,\\ 1, & x\geq 1. \end{cases} \tag{3.7}$$

The corresponding exact solution is

$$u(x,t)=\begin{cases} 0, & x\leq 0,\\ \dfrac{x}{ct+1}, & 0<x<(ct+1)^2,\\ ct+1, & x\geq (ct+1)^2. \end{cases} \tag{3.8}$$

The computational domain is $D=[-6,6]$. In this example, we set the case of $c=0.5$ as the foundational test. Others will be considered in later tests for different initial conditions.

Accuracy tests with $c=0.5$. In this part, we choose the size of $\Delta x=0.15$ and $\Delta t=\frac{5}{24}\Delta x$ in the training process. Training data sets are generated from the solution trajectory of (3.8)

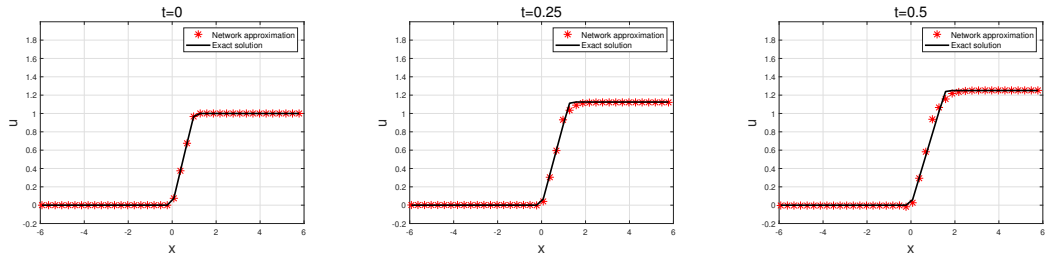

Figure 10: Trajectory simulation with $c=0.5$ and $\Delta t=\frac{5}{24}\Delta x$.

associated with $c=0.5$. The neural network picked contains 1 hidden layer of 8 neurons. The CANN solver applies input vector of

$$\overrightarrow{V}_j^{in}=\left[\bar{u}_{j-6}^n,\bar{u}_{j-5}^n,\bar{u}_{j-4}^n,\bar{u}_{j-3}^n,\bar{u}_{j-2}^n,\bar{u}_{j-1}^n,\bar{u}_j^n,\bar{u}_{j+1}^n,\bar{u}_{j+2}^n,\bar{u}_{j+3}^n\right]^T. \tag{3.9}$$

Network training is conducted for up to $k=10^4$ iterations. After well trained, we utilized the neural network to solve the HS equation up to $T=0.5$. Screen shots of $t=0$, $t=0.25$, and $t=0.5$ are shown in Fig. 10. We can see clearly that the moving profile can be obtained accurately by the CANN method.

Table 2: Errors of neural network methods for Example 3.8 with $c=0.5$, $\Delta x=0.15$ fixed with varying Δt.

Δt	L_2	L_∞
$5/24\Delta x$	2.6e-03	9.790e-02
$1/3\Delta x$	2.7e-03	1.016e-01
$5/9\Delta x$	2.6e-03	1.009e-01
$5/3\Delta x$	2.6e-03	1.088e-01

Furthermore, in order to test the accuracy with large time step, we consider the fixed spatial mesh size $\Delta x=0.15$ and different temporal mesh size $\Delta t=\frac{5}{24}\Delta x,\frac{1}{3}\Delta x,\frac{5}{9}\Delta x,\ \frac{5}{3}\Delta x$ correspondingly. The well trained CANN solver will be applied directly to solve the wave propagation up to $T=0.5$. L_2 and L_∞ errors are computed in Table 2. We have seen that even though the wave simulations have similar accuracy, our CANN method can use the large time step size $\Delta t=5/3\Delta x$ (compared to the CFL restriction $\Delta t=\Delta x^2$) to be implemented as a fast solver.

Different initial conditions. Next, in order to show the ability of generalization, we apply the well trained CANN which is generated from the learning data set associated with the trajectory $c=0.5$ to solve Eq. (3.8) under different initial conditions. In this paper, the CANN solver can be implemented as a fixed numerical scheme without changing the architecture. We just need to change the size of the time step with $\Delta t=\frac{5}{6}\Delta x$ to match the different parameter $c=0.1$ in the simulation process by using the optimal CANN method directly without retraining. Then the numerical approximations are obtained

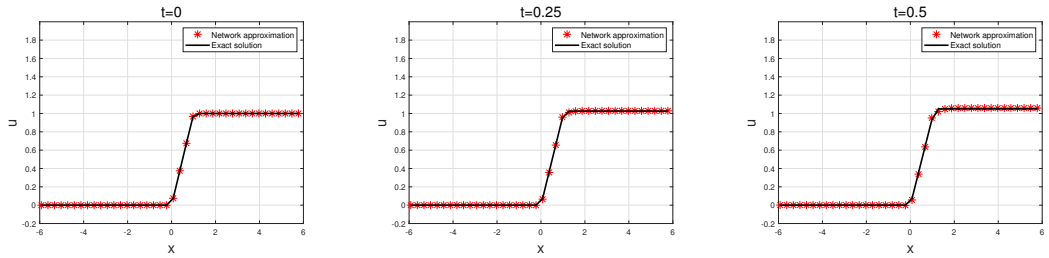

Figure 11: Apply optimal CANN to solve different initials with $c=0.1$ and $\Delta t=\frac{5}{6}\Delta x$.

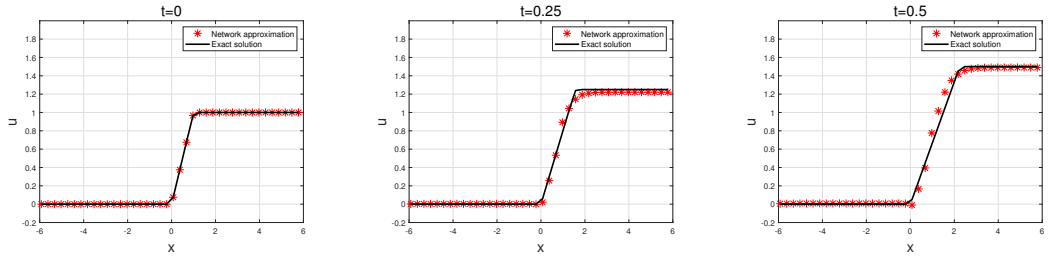

Figure 12: Apply optimal CANN to solve different initials with $c=1$ and $\Delta t=\frac{5}{42}\Delta x$.

accurately from CANN solver for different initial conditions of $c=0.1$ and $c=1$ without retraining the network, see Fig. 11 and Fig. 12. The CANN method has strong capability for generalization for different initial conditions in the wave propagation simulation.

Low quality data. In order to validate the robustness of our CANN method, we also use it to deal with the noisy learning data in (2.11) perturbed by a multiplicative factor. Without changing the architecture of the optimal neural network associated with $c=0.5$ in the accuracy tests, we apply it for the corrupted learning data with η=0.02 and η=0.05.

In the cases of low frequency $\eta=0.02$ and $\eta=0.05$, we have the L_2 and the L_∞ errors around 10^{-3} and 10^{-1} by choosing the cell size $\Delta x=0.15$ and $\Delta t=\frac{10}{9}\Delta x$. Screen shots of t=0, t=0.25, and t=0.5 are shown in Fig. 13. It's no surprise that the predictions for noisy data by using the CANN model agree well with the exact solution in the approximation without retraining the neural network.

Furthermore, we also test the cases of high frequency noise perturbation of η=0.2 and $\eta=0.5$. From Fig. 14, we can observe that the main structure of the wave propagation captured by CANN method is good.

3.3 Example 3

In this section, we consider the regularization with dispersion of the HS equation

$$u_{xxt}+2u_xu_{xx}+uu_{xxx}-\varepsilon_2u_{xxxxx}=0, \tag{3.10}$$

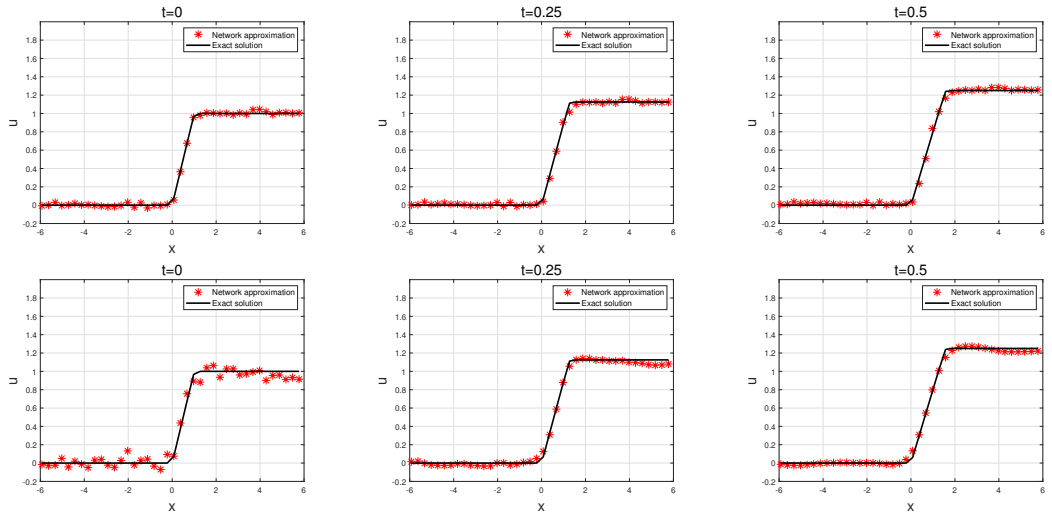

Figure 13: Trajectory simulation with corruption learning data by $\eta=0.02$ (top) and $\eta=0.05$ (bottom).

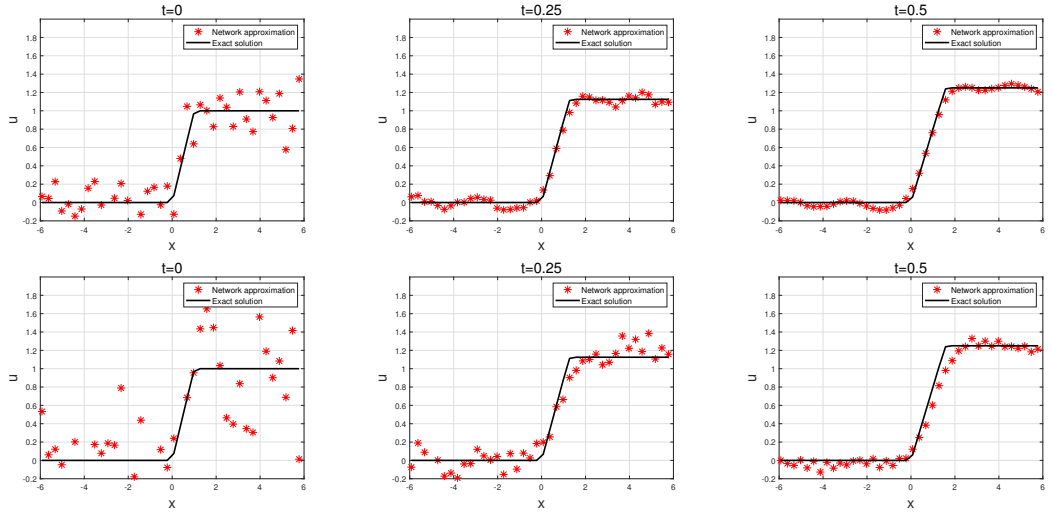

Figure 14: Trajectory simulation with corruption learning data by $\eta=0.2$ (top) and $\eta=0.5$ (bottom).

where $\varepsilon_2 \geq 0$ is a constant. In this example, we consider the zero-dispersion limit of (3.6) (i.e., zero-dispersion limit $\varepsilon_2 = 0$)with the initial data

$$u(x,0) = \begin{cases} 2, & x \leq 0.25, \\ 3-4x, & 0.25 < x < 0.75, \\ 0, & x \geq 0.75. \end{cases} \tag{3.11}$$

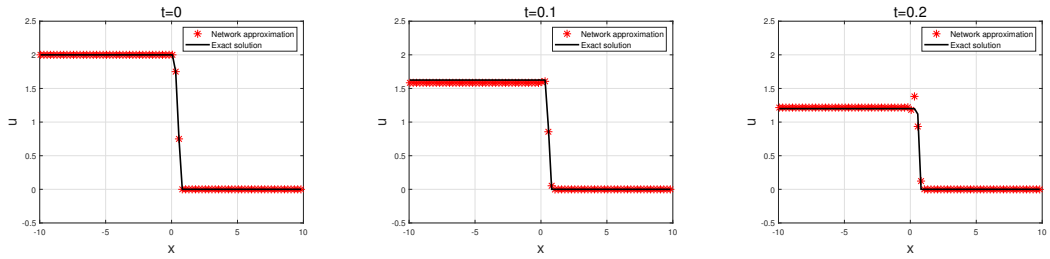

Figure 15: Trajectory simulation with $c=1$ and $\Delta t=\frac{1}{20}\Delta x$.

The exact solution of the HS equation (3.10) is

$$u(x,t)=\begin{cases} -4(ct-0.5), & x\leq 0.75-2(ct-0.5)^2, \\ \dfrac{2(x-0.75)}{ct-0.5}, & 0.75-2(ct-0.5)^2<x<0.75, \\ 0, & x\geq 0.75. \end{cases} \tag{3.12}$$

The computational domain is $D=[-10,10]$.

Accuracy tests: $c=1$. In this experiment, we choose $\Delta x=0.125$ and $\Delta t=\frac{1}{20}\Delta x$ in the training of CANN method. The learning data S can be generated from the single solution trajectory of (3.11) and (3.12) with $c=1$. The CANN solver contains 1 hidden layer of 8 neurons. And the essential input vector can be conducted by

$$\overrightarrow{V}_j^{in}=\left[\bar{u}_{j-4}^n,\bar{u}_{j-3}^n,\bar{u}_{j-2}^n,\bar{u}_{j-1}^n,\bar{u}_j^n,\bar{u}_{j+1}^n,\bar{u}_{j+2}^n,\bar{u}_{j+3}^n\right]^T. \tag{3.13}$$

After well trained, the neural network is applied for solving the HS equation (3.10) to $T=0.2$. Screen shots are shown in Fig. 15. We can see clearly that the moving shock profile is simulated very well by the CANN method.

In order to test the accuracy that explicitly evolve the solution forward in time with large time step size, we apply the CANN method to solve the HS equation with fixed spatial mesh size $\Delta x=0.125$ and different temporal mesh size $\Delta t=\frac{1}{20}\Delta x,\frac{1}{10}\Delta x,\frac{1}{5}\Delta x,\frac{4}{5}\Delta x$ correspondingly. L_2 and L_∞ errors are computed in Table 3. We can see clearly that our CANN method can solve the HS equation with larger time step size $\frac{4}{5}\Delta x$ which is bigger than the conventional method CFL restriction. In a word, the well trained CANN solver can be implemented as a explicit numerical scheme and relieved from CFL restriction on time step size.

Different initial conditions. Next, in order to test the generalization ability of our CANN method, we apply it to solve HS equation (3.10) under different initial conditions which are generated by applying different parameter c in the exact solution (3.12). Here, the well trained CANN from the accuracy test part with $c=1$ will be applied directly without retraining the neural network.

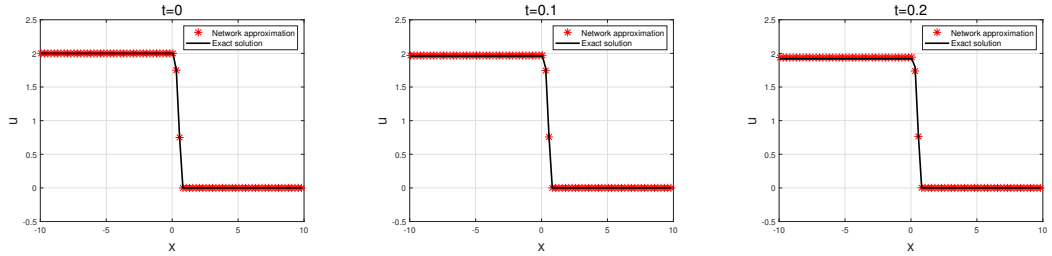

Figure 16: Apply optimal CANN to solve different initials with $c=0.1$ and $\Delta t=\frac{4}{5}\Delta x$.

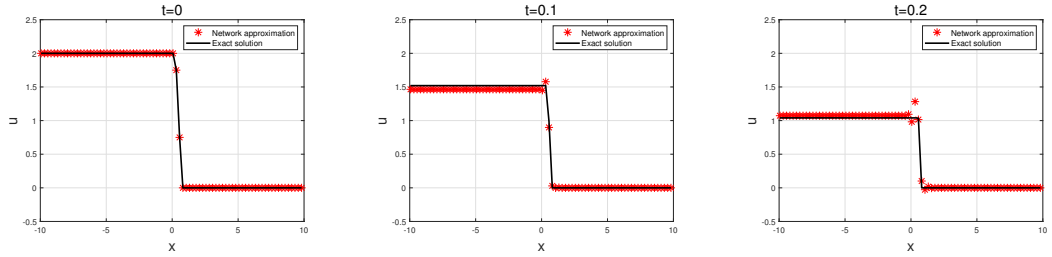

Figure 17: Apply optimal CANN to solve different initials with $c=1.2$ and $\Delta t=\frac{2}{25}\Delta x$.

In this paper, we test the different initial conditions with $c=0.1$ and $c=1.2$ as the examples. However, we still need to change the time step size to match the different parameter c in the process of generating the well trained CANN solver for different initial conditions. Here, we have $\Delta t=\frac{4}{5}\Delta x$ for $c=0.1$ and $\Delta t=\frac{2}{25}\Delta x$ for $c=1.2$.

From Fig. 16 and Fig. 17, we can clearly see that our CANN solver can obtain the good simulation for different initial conditions without retraining the neural network. Also, one interesting finding is that the smaller c is, the better simulation of the motion wave can be obtained.

Low quality data. Furthermore, we also carry out the CANN solver to deal with the corrupted learning data associated with Eq. (3.10). Here, we consider the learning data S_N with low frequency $\eta=0.02$ and $\eta=0.05$. The well trained CANN solver is from the accuracy test part and doesn't change the architecture any more, especially for the hidden layers and neurons.

Table 3: Errors of CANN method for Example 3.10 with varying Δt.

Δt	L_2	L_∞
$1/20\Delta x$	2.1000e-03	2.109e-01
$1/10\Delta x$	2.3000e-03	2.227e-01
$1/5\Delta x$	2.6000e-03	2.3010e-01
$4/5\Delta x$	4.5000e-03	2.6980e-01

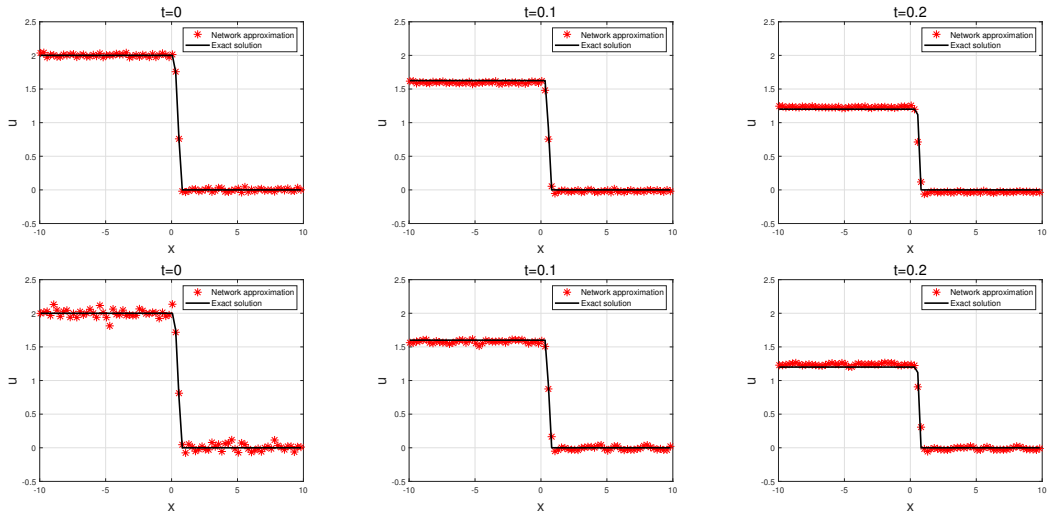

Figure 18: Trajectory simulation with corruption learning data by $\eta = 0.02$ (top) and $\eta = 0.05$ (bottom).

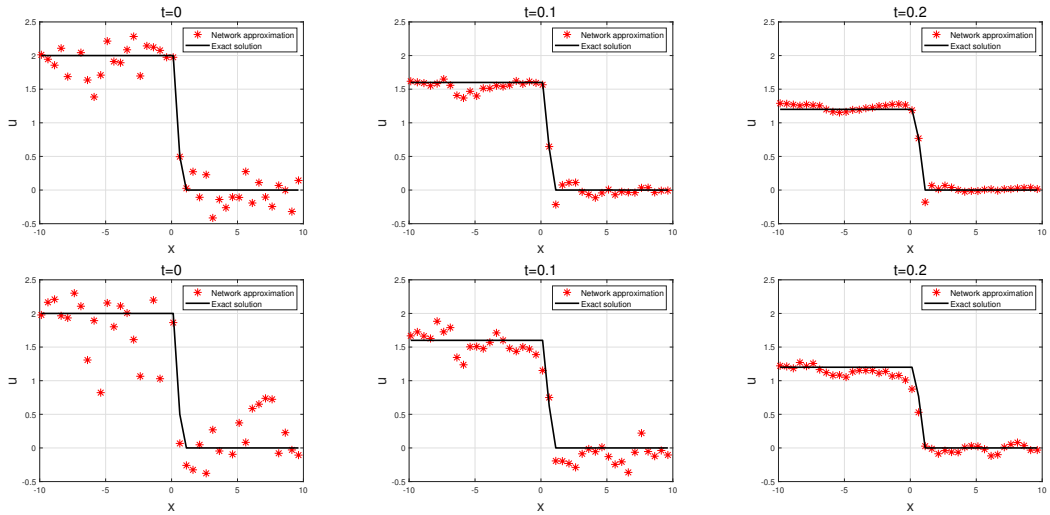

Figure 19: Trajectory simulation with corruption learning data by $\eta = 0.2$ (top) and $\eta = 0.5$ (bottom).

In this experiment, we denote meshes by $\Delta x = 0.125$ and $\Delta t = \frac{2}{5}\Delta x$. Fig. 18 show that for the corrupted learning data, our CANN method can still accurately and sharply capture the wave evolution that is comparable to exact solution. Furthermore, for the high frequency $\eta = 0.2$ and $\eta = 0.5$, we also have the good performance in Fig. 19. In a word, the predictions of the CANN method are robust for data turbulence and can accurately predict the main structure of wave propagation without retraining the neural network.

3.4 Example 4

In this section, we consider the HS equation in [14]

$$u_t+2u_xu_x-u_{xxt}+uu_x=2u_xu_xx+uu_xxx. \tag{3.14}$$

The initial condition is

$$u(x,0)=e^x. \tag{3.15}$$

The exact solution of the HS equation (3.14) is

$$u(x,t)=e^{x-ct}, \tag{3.16}$$

where different c will yield different initial values or exact solutions. The computational domain is $D=[-4,1]$.

Accuracy tests: $c=3$**.** In this experiment, we carry out the learning data set associated with trajectory solution of $c=3$ in (3.16) to test the accuracy of CANN method. By setting $\Delta x=0.05$ and $\Delta t=\frac{5}{4}\Delta x$, we choose the same neural network input vector of (3.9) as for HS equation (3.16). With the architecture of 1 hidden layer and 8 neurons, we train the CANN solver up to $k=10^4$ iterations. After well trained, the CANN solver can be applied for solving the HS equation to finial time $T=0.5$. Screen shots are shown in Fig. 20. We can find that the numerical approximation of CANN method are comparable to those exact solutions and no large oscillation is generated with the neural network method.

Additionally, in order to validate the accuracy of large time step for our CANN method, we consider the fixed spatial mesh size $\Delta x=0.05$ and different temporal mesh size $\Delta t=\frac{1}{2}\Delta x,\Delta x,\frac{5}{4}\Delta x$ and $\frac{5}{2}\Delta x$. In Table 4, we list the errors computed at $T=0.5$ corresponding to each coarse size Δt. Again, all simulations with different Δt are found stable. The larger time step size, for example $\Delta t=\frac{5}{2}\Delta x$, can be applied to implement the CANN solver such that it can be as an efficient neural network solver. We would like to point out that this Δt is bigger than the CFL restriction of the conventional numerical methods for explicit temporal discretization. But CANN method is still found stable and accurate for this Δt.

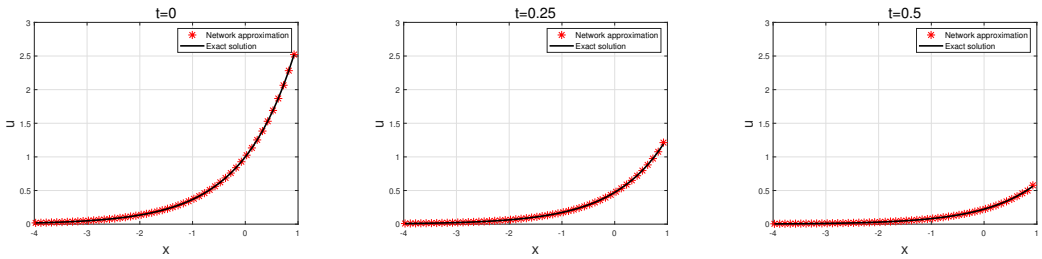

Figure 20: Trajectory simulation with $c=3$ and $\Delta t=\frac{5}{4}\Delta x$.

Table 4: Errors of CANN method for Example (3.16) with varying Δt.

Δt	L_2	L_∞
$1/2\Delta x$	1.8878e-04	1.44e-02
Δx	1.7465e-04	1.21e-02
$5/4\Delta x$	1.5877e-04	9.20e-03
$5/2\Delta x$	1.8424e-04	1.03e-02

Different trajectories approximation. Next, we test the generalization ability of CANN method by solving Eq. (3.14) under different initial conditions. Here, different initial conditions are obtained from the exact solution (3.16) with different parameter c. In this paper, the CANN solver, which is well trained in the accuracy test with parameter $c = 3$, will be utilized to solve the different initial conditions with $c = 1.5$ and $c = 4.5$. We highlight that the well trained CANN solvers can be used as fixed numerical scheme directly without retraining.

As a fixed numerical scheme, it does not need to change the architecture, especially for the numbers of hidden layers and neurons. However, in order to match the different parameter c in the simulation process, we need to change time step size. Now, we need to change $\Delta t = \frac{5}{2}\Delta x$ for $c = 1.5$ and $\Delta t = \frac{5}{6}\Delta x$ for $c = 4.5$. From Fig. 21 and Fig. 22, we can clearly see that our CANN method can capture the evolution accurately and sharply, which is comparable to exact solutions.

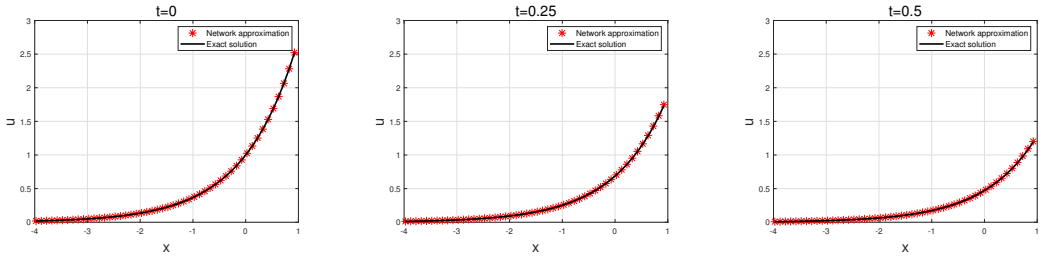

Figure 21: Apply optimal CANN to solve different initials with $c = 1.5$.

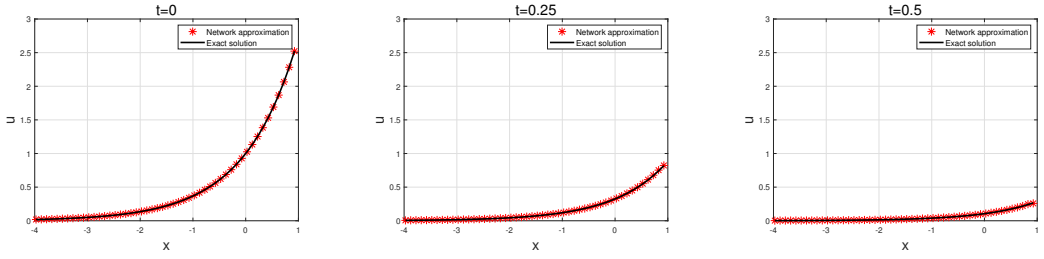

Figure 22: Apply optimal CANN to solve different initials with $c = 4.5$.

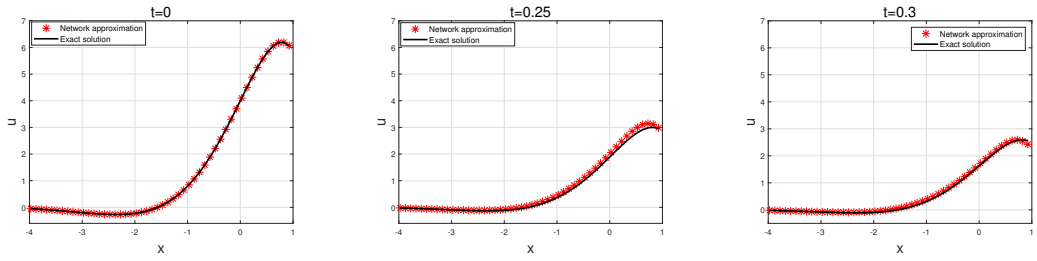

Figure 23: Apply optimal CANN to approximate different solution trajectory with $u(x,t)=4e^{(x-3t)}\cos(x-1/10t)$.

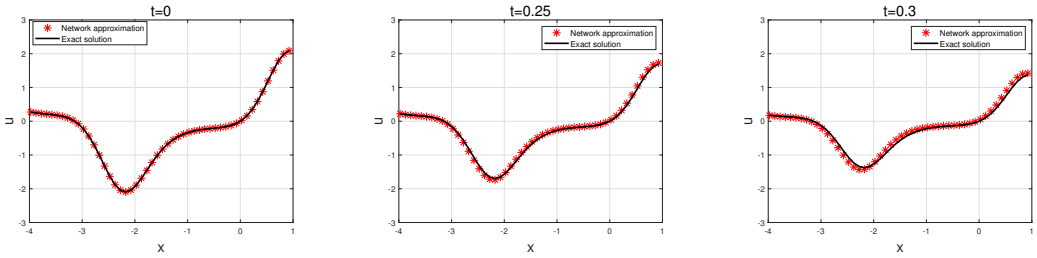

Figure 24: Apply optimal CANN to approximate different solution trajectory with $u(x,t)=\sin(x)e^{\sin(2x)-\frac{7}{5}t}$.

In addition, we also test the cases whose solution trajectories are completely different. We utilize the well trained neural network, which is obtained from the trajectory with $c=3$ in equation (3.16), to approximate the different solution trajectories with

$$u(x,t)=4e^{x-ct}\cos\left(x-\frac{1}{10}t\right) \quad \text{and} \quad u(x,t)=\sin(x)e^{\sin(2x)-\frac{7}{5}t}.$$

In Fig. 23 and Fig. 24, we can find that the performance of our CANN method in solving different solution trajectories is very good.

Low quality data. To validate the capability for corrupted data, we also utilize the CANN solver to deal with the learning data S_N in (2.11) with η.

In the case of $\eta=0.02$ and $\eta=0.05$, we find that the errors of L_2 and L_∞ are around 10^{-4} and 10^{-3} by setting $\Delta x=0.05$ and $\Delta t=\frac{5}{2}\Delta x$. From Fig. 25, we can observe that the main structure of the wave propagation can be captured well in the predictions. For the larger $\eta=0.2$ and $\eta=0.5$, Fig. 26 shows that the CANN method is fairly robust and works well for the high frequency noise.

4 Conclusions

We have developed a cell-average based neural network to solve the Hunter-Saxton equation. The cell-average based neural network combines the neural network with the fi-

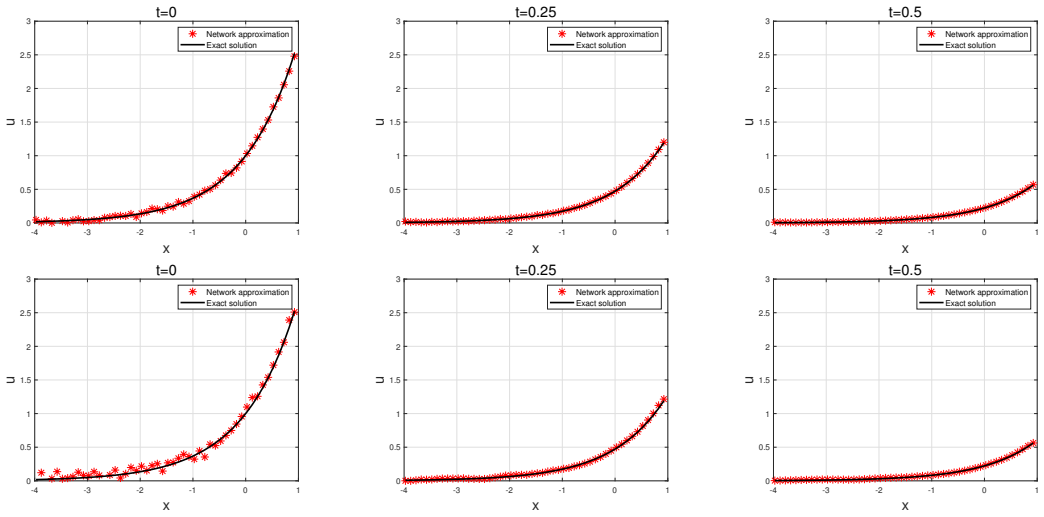

Figure 25: Trajectory simulation with corruption learning data by $\eta = 0.02$ (top) and $\eta = 0.05$ (bottom).

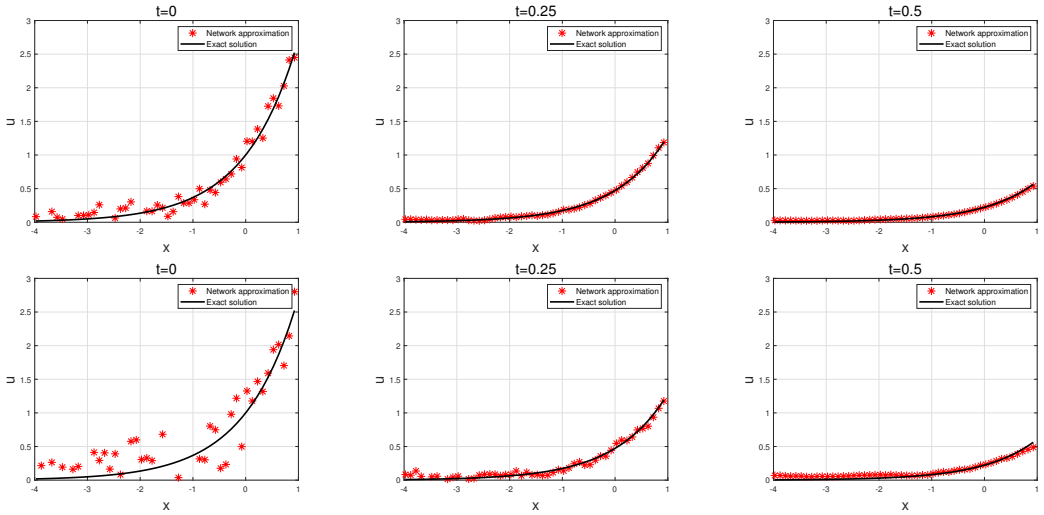

Figure 26: Trajectory simulation with corruption learning data by $\eta = 0.2$ (top) and $\eta = 0.5$ (bottom).

nite volume method, where a feedforward network is formed to learn the cell average between two consecutive time steps, based on the integral format of the finite volume method. The training data set is generated by cell average based on one single solution trajectory which is usually the initial value of the given PDE. The training process uses multiple time levels of cell averages to maintain the stability and control the accumulation errors in time. Once being well trained, this method can be implemented like a explicit finite volume scheme and suitable for solving the Hunter-Saxton equation with

different initial value conditions, while we don't need to retrain the neural network. We also utilize the CANN method to deal with the learning data with noise to demonstrate its capability and robustness. Numerical tests are carried out to demonstrate effectiveness, accuracy, capability and robustness of the CANN method. Applications for more complicated nonlinear PDEs will be conducted in the future work.

Acknowledgements

This work is supported by National Natural Science Foundation of China (No. 12201327) and Ningbo Natural Science Foundation (No. 2022J087).

References

[1] J. K. HUNTER, AND R. SAXTON, *Dynamics of director fields,* SIAM J. Appl. Math., 51(6) (1991), pp. 1498–1521.
[2] J. K. HUNTER, AND Y. ZHENG, *On a completely integrable nonlinear hyperbolic variational equation,* Phys. D, 79(2) (1994), pp. 361–386.
[3] J. K. HUNTER, AND Y. X. ZHENG, *On a nonlinear hyperbolic variational equation. I. Global existence of weak solutions,* Arch. Rational Mech. Anal., 129(4) (1995), pp. 305–353.
[4] J. K. HUNTER, AND Y. ZHENG, *On a nonlinear hyperbolic variational equation: II. The zero-viscosity and dispersion limits*, Arch. Ration Mech. An., 129(4) (1995), pp. 355–383.
[5] H. HOLDEN, AND K. RISEBRO, *Convergent difference schemes for the Hunter-Saxton equation,* Math. Comput., 76(258) (2007), pp. 699–744.
[6] Y. XU, AND C.-W. SHU, *Local discontinuous galerkin method for the Hunter–Saxton equation and its zero-viscosity and zero-dispersion limits,* SIAM J. Sci. Comput., 31(2) (2009), pp. 1249–1268.
[7] Y. XU, AND C.-W. SHU, *Dissipative numerical methods for the Hunter-Saxton equation,* J. Comput. Math., 28(5) (2010), pp. 606–620.
[8] M. AWAIS, M. S. HASHMI, A. WAHEED, AND Q. ALI, *Numerical treatment of Hunter-Saxton equation using cubic trigonometric b-spline collocation method,* AIP Adv., 7(9) (2017), pp. 095124-1–095124-12.
[9] K. S, H. REZAZADEH, AND W. ADEL, *Numerical investigation based on laguerre wavelet for solving the Hunter-Saxton equation,* Int. J. Appl. Comput. Math., 6 (2020), pp. 1–14.
[10] I. AHMAD, H. ILYAS, K. KUTLU, V. ANAM, S. I. HUSSAIN, AND J. L. G. GUIRAO, *Numerical computing approach for solving Hunter-Saxton equation arising in liquid crystal model through sinc collocation method,* Heliyon, 7(7) (2021), e07600.
[11] B. KARAAGAC, AND A. ESEN, *The Hunter-Saxton equation: A numerical approach using collocation method,* Numer. Meth. Partial Differential Equations, 34(5) (2018), pp. 1637–1644.
[12] S. ARBABI, A. NAZARI, AND M. T. DARVISHI, *A semi-analytical solution of Hunter-Saxton equation,* Optik, (2016), pp. 5255–5258.
[13] K. PARAND, AND M. DELKHOSH, *An efficient numerical solution of nonlinear Hunter–Saxton equation,* Commun. Theor. Phys., 67 (2017), 483.
[14] S. S. BEHZADI, *Numerical solution of Hunter-Saxton equation by using iterative methods,* J. Inform. Math. Sci., 3 (2011), pp. 127–143.
[15] A. ATANGANA, D. BALEANU, AND A. ALSAEDI, *Analysis of time-fractional Hunter-Saxton equation: a model of neumatic liquid crystal,* Open Phys., 14(1) (2016), pp. 145–149.

[16] M. IZADI, *Numerical approximation of Hunter-Saxton equation by an efficient accurate approach on long time domains,* Appl. Math. Phys., 83 (2021), pp. 291–300.
[17] W. E, *A proposal on machine learning via dynamical systems,* Commun. Math. Stat., 5(1) (2017), pp. 1–11.
[18] P. CHAUDHARI, A. OBERMAN, S. OSHER, S. SOATTO, AND G. CARLIER, *Deep Relaxation: partial differential equations for optimizing deep neural networks*, Res. Math. Sci., 5 (2018).
[19] S. H. RUDY, S. L. BRUNTON, J. L. PROCTOR, AND J. N. KUTZ, *Data-driven discovery of partial differential equations,* Sci. Adv., 3(4) (2017), e1602614.
[20] B. CHANG, L. MENG, E. HABER, L. RUTHOTTO, D. BEGERT, ANAD E. HOLTHAM, *Reversible architectures for arbitrarily deep residual neural networks,* AAAI Conference on AI, 2018.
[21] W. E, AND B. YU, *The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems,* Commun. Math. Stat. 6(1) (2018), pp. 1–12.
[22] J. SIRIGNANO, AND K. SPILIOPOULOS, *Dgm: A deep learning algorithm for solving partial differential equations,* J. Comput. Phys., 375 (2018), pp. 1339–1364.
[23] Z. LONG, Y. LU, AND B. DONG, *PDE-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network,* J. Comput. Phys., 399 (2019), 108925.
[24] J. HE, AND J. XU, *Mgnet: A unified framework of multigrid and convolutional neural network,* Sci. China Math., 62 (2019), pp. 1331–1354.
[25] Y. BAR-SINAI, S. HOYER, J. HICKEY, AND M. P. BRENNER, *Learning data-driven discretizations for partial differential equations,* Proc. Natl. Acad. Sci. USA, 116(31) (2019), pp. 15344–15349.
[26] 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,* J. Comput. Phys., 378 (2019), pp. 686–707.
[27] W. FENG, AND H. HUANG, *Fast prediction of immiscible two-phase displacements in heterogeneous porous media with convolutional neural network,* Adv. Appl. Math. Mech., 13(1) (2020), pp. 140–162.
[28] L. RUTHOTTO, AND E. HABER, *Deep neural networks motivated by partial differential equations,* J. Math. Imag. Vis., 62(04) (2020), pp. 352–364.
[29] H. LIU, J. SONG, H. LIU, J. XU, AND L. LI, *Legendre neural network for solving linear variable coefficients delay differential-algebraic equations with weak discontinuities,* Adv. Appl. Math. Mech., 13(1) (2021), pp. 101–118.
[30] X. ZHU, M. HE, AND P. SUN, *Comparative studies on mesh-free deep neural network approach versus finite element method for solving coupled nonlinear hyperbolic/ wave equations,* Int. J. Numer. Anal. Mod., 19 (2022), pp. 603–629.
[31] I. E. LAGARIS, A. LIKAS, ANAD D. I. FOTIADIS, *Artificial neural networks for solving ordinary and partial differential equations,* IEEE T. Neural Networks, 9(5) (1998), pp. 987–1000.
[32] M. RAISSI, P. PERDIKARIS, AND G. E. KARNIADAKIS, *Physics informed deep learning (Part I): data-driven solutions of nonlinear partial differential equations*, arXiv e-prints (2017), `http://arxiv.org/abs/1711.10561` arXiv:1711.10561.
[33] M. RAISSI, P. PERDIKARIS, AND G. E. KARNIADAKIS, *Physics informed deep learning (part II): Data-driven discovery of nonlinear partial differential equations,* `https://doi.org/10.48550/arXiv.1711.10566`.
[34] L. LU, X. MENG, Z. MAO, AND G. E. KARNIADAKIS, *Deepxde: A deep learning library for solving differential equations,* SIAM Rev., 63(1) (2021), pp. 208–228.
[35] J. LI, W. ZHANG, AND J. YUE, *A deep learning galerkin method for the second-order linear elliptic equations,* Int. J. Numer. Anal. Mod., 18 (2021), pp. 427–441.
[36] J. BERG, AND K. NYSTRÖM, *A unified deep artificial neural network approach to partial differential*

equations in complex geometries, Neurocomputing, 317 (2018), pp. 28–41.
[37] K. RUDD, AND S. FERRARI, *A constrained integration (cint) approach to solving partial differential equations using artificial neural networks,* Neurocomputing, 155 (2015), pp. 277–285.
[38] A. YANG, AND F. GU, *A mesh-less, ray-based deep neural network method for the helmholtz equation with high frequency,* Int. J. Numer. Anal. Mod., 19 (2022), pp. 587–601.
[39] V. DWIVEDI, AND B. SRINIVASAN, *Physics informed extreme learning machine (pielm)–a rapid method for the numerical solution of partial differential equations-sciencedirect,* Neurocomputing, 391 (2020), pp. 96–118.
[40] X. JIN, S. CAI, H. LI, AND G. E. KARNIADAKIS, *Nsfnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations,* J. Comput. Phys., 426 (2021), 109951.
[41] Y. ZANG, G. BAO, X. YE, AND H. ZHOU, *Weak adversarial networks for high-dimensional partial differential equations,* J. Comput. Phys., 411 (2020), 109409.
[42] Z. CAI, J. CHEN, AND M. LIU, *Least-squares ReLU neural network (LSNN) method for linear advection-reaction equation,* J. Comput. Phys., 443 (2021), 110514.
[43] Z. CAI, J. CHEN, AND M. LIU, *Least-squares ReLU Neural Network (LSNN) method for scalar nonlinear hyperbolic conservation law*, arXiv e-prints (2021), `http://arxiv.org/abs/2105.11627` arXiv:2105.11627.
[44] K. WU, AND D. XIU, *Data-driven deep learning of partial differential equations in modal space,* J. Comput. Phys., 408 (2020), 109307.
[45] T. QIN, Z. CHEN, J. D. JAKEMAN, AND D. XIU, *Data-driven learning of nonautonomous systems,* SIAM J. Sci. Comput., 43(3) (2021), pp. A1607–A1624.
[46] D. RAY, AND J. S. HESTHAVEN, *An artificial neural network as a troubled-cell indicator,* J. Comput. Phys., 367 (2018), pp. 166–191.
[47] Y. WANG, *Learning to discretize: Solving 1d scalar conservation laws via deep reinforcement learning,* Commun. Comput. Phys., 28(5) (2020), pp. 2158–2179.
[48] N. DISCACCIATI, J. S. HESTHAVEN, AND D. RAY, *Controlling oscillations in high-order discontinuous Galerkin schemes using artificial viscosity tuned by neural networks,* J. Comput. Phys., 409 (2020), 109304.
[49] Z. SUN, S. WANG, L.-B. CHANG, Y. XING, AND D. XIU, *Convolution neural network shock detector for numerical solution of conservation laws,* Commun. Comput. Phys., 28 (2020), pp. 2075–2108.
[50] A. D. BECK, J. ZEIFANG, A. SCHWARZ, AND D. FLAD, *A neural network based shock detection and localization approach for discontinuous Galerkin methods,* J. Comput. Phys., 423 (2020), 109824.
[51] X. YU, AND C.-W. SHU, *Multi-layer perceptron estimator for the total variation bounded constant in limiters for discontinuous Galerkin methods,* La Math., 1(1) (2021), 53–84.
[52] C. QIU, AND J. YAN, *Cell-average based neural network method for hyperbolic and parabolic partial differential equations,* `https://doi.org/10.48550/arXiv.2107.00813`.
[53] X. ZHOU, C. QIU, W. YAN, AND B. LI, *Mastering the Cahn–Hilliard equation and Camassa–Holm equation with cell-average-based neural network method,* Nonlinear Dyn., 111 (2023), pp. 4823–4846.