LEARNING THE DYNAMICS FOR UNKNOWN HYPERBOLIC CONSERVATION LAWS USING DEEP NEURAL NETWORKS

ZHEN CHEN*, ANNE GELB†, AND YOONSANG LEE†

Abstract. We propose a new data-driven method to learn the dynamics of an unknown hyperbolic system of conservation laws using deep neural networks. Inspired by classical methods in numerical conservation laws, we develop a new conservative form network (CFN) in which the network learns to approximate the numerical flux function of the unknown system. Our numerical examples demonstrate that the CFN yields significantly better prediction accuracy than what is obtained using a standard non-conservative form network, even when it is enhanced with constraints to promote conservation. In particular, solutions obtained using the CFN consistently capture the correct shock propagation speed without introducing non-physical oscillations into the solution. They are furthermore robust to noisy and sparse observation environments.

1. Introduction.

Hyperbolic systems of conservation laws arise in many applications, in particular where wave motion or advective transport is important, and include problems in gas dynamics, acoustics, optics and elastodynamics. These typically non-linear partial differential equations (PDEs) are well known to cause challenges for numerical simulation. Methods that work well for linear problems break down near discontinuities in the non-linear case unless special care is taken to mitigate the oscillatory behavior that will otherwise cause instabilities. Moreover it is well understood that numerical solvers must be written in *flux conserving form*, since otherwise the resulting numerical solution may yield the incorrect shock speed, [18, 24, 25].

Since enormous quantities of data can now be collected, and since there is increased capacity in computational storage along with better computational efficiency, data-driven algorithms that recover unknown dynamical systems from observation data, mainly for ordinary differential equations (ODEs) [4, 9, 27, 34, 40, 43, 47] but more recently for PDEs [26, 28, 29, 30, 35, 36, 44, 45] as well, are becoming more widespread. Many data-driven methods build neural network (NN) models that are trained to fit observed data. Such an approach is called "blind" since it does not take into account any information regarding the underlying system. As will be demonstrated in our numerical results, while a standard NN technique may yield accurate results within the training time domain, numerical difficulties arise once shocks are formed beyond that time period. Indeed, the NN may predict non-physical solutions or the wrong shock propagation speed.

To provide context, we briefly summarize some ODE and PDE dynamics-learning methods that are designed to impose known physical constraints regarding the underlying system. For instance, regularization terms can be introduced into the loss function to penalize the NN that would otherwise not satisfy physical constraints [11, 15]. Another typical approach seeks to integrate general physical principles into the design of neural networks. For example, the method in [46] embeds GENERIC formalism into neural networks to learn dynamical systems and involves two separate generators for reversible and irreversible dynamics, respectively. Systems of ODEs and PDEs are considered in [2], where general physical principles are integrated into

*ExxonMobil Technology and Engineering Company, Houston, TX 77002, USA.

†Department of Mathematics, Dartmouth College, Hanover, NH 03755, USA. Emails: zhen.chen@exxonmobil.com, annegelb@math.dartmouth.edu, Yoonsang.lee@dartmouth.edu Funding: All authors are supported by the DoD MURI grant ONR #N00014-20-1-2595. AG is also supported by the AFOSR grant #FA9550-22-1-0411 and the NSF grant DMS #1912685. YL is also supported by the NSF grant DMS #1912999.

what is called *dynamic mode decomposition* (DMD), which learns low-rank dynamics from high-dimensional measurements. This case-by-case study does not have a general framework that can be extended to conservation laws, however. There is a also class of data-driven methods that aim to identify the governing system of ODEs or PDEs, see e.g. [7, 38, 39]. The time derivative must be approximated from the observation data, thereby limiting observation environments to those where high quality data are obtainable in very short time intervals. Furthermore, the solution must remain smooth and differentiable throughout the considered time domain. In another category of data-driven approaches, [19, 20, 21, 28, 36], it is possible to obtain either the forward PDE solution (after some prescribed time), or some unknown parameters associated with the system of equations. Such methods rely on either knowing the system explicitly or terms within its equations. Notably, methods such as those proposed in [19, 20] are specifically designed for conservation laws and can be used to recover constant coefficients of inviscid and higher-order fluxes. The framework developed in [28] learns the Hamiltonian of a dynamical system, which is conserved. Mathematically this work is based on symplectic dynamics. To obtained desired accuracy, the method requires the calculation of derivatives of the state variable (or second order derivatives for KdV equations), which in turn requires dense observations of the state variables. By contrast, our new CFN approach bypasses such calculations by directly learning the flux function without relying on the Hamiltonian framework. Finally, we mention approaches that concentrate exclusively on solving the forward problem by combining machine learning (ML) techniques with traditional numerical PDE knowledge. Examples of such methods which have been used to solve conservation laws can be found in [3, 5, 37, 42].

In spite of their ubiquitousness in many applications, to the best of our knowledge there are no data-driven methods that are specifically designed to learn unknown systems of hyperbolic conservation laws. In this investigation we therefore seek to develop a method to do so, and in particular for the case in which the observation data are comprised of a set of perturbed numerical solutions of the true PDE at a finite number of different time instances within a short time (training) domain. Our new method employs tools from ML to construct a model from the training data to make long-term predictions for a system of hyperbolic conservation laws that extend beyond the short-time domain for which data are observed. Notably, our method is a departure from a traditional interpretation of learning where the training data is designed to contain all of the information eventually needed. In contrast, here we consider the realistic setting where the snapshot data comes from a given physical system up to a certain time, and we seek to *extrapolate* the solution to a future time. Such a scenario is important in long term predictions of non-linear dynamics that are modeled by hyperbolic conservation laws.

Based on similar behavior observed in traditional solvers for numerical conservation laws, here we propose a new *conservative form network* (CFN) for which the network learns the flux function of the unknown system. By incorporating the conservative flux form directly into the network architecture, we are able to mimic the structure of the conservative form scheme found in classical numerical hyperbolic conservation laws. Our new method is distinguishable from the aforementioned approaches as it designs the network to be specifically in conservative form. Our numerical experiments demonstrate that the data-driven method resulting from our new CFN is conservative. It furthermore correctly predicts the shock propagation speed in extended time domains without introducing non-physical oscillations into the solution. This is in contrast to data-driven methods that use either the standard "blind"

NN or those that incorporate a penaly term to promote conservation.

The rest of the paper is organized as follows. Section 2 gives a brief review of conservation laws along with some standard ideas related to dynamics-learning methods. We introduce our new conservative form network in Section 3. Section 4 discusses how our experiments are designed. This is followed by three classical examples of one-dimensional conservation laws in Section 5 used to validate our approach. Section 6 provides some concluding remarks.

2. Preliminaries.

2.1. Conservation laws. We are interested in learning the dynamics of an unknown hyperbolic system of conservation laws on a spatial domain $x \in (a, b)$ and temporal domain $t \in (0, T)$. The scalar form is given by

$$u(x,t)_t + f(u(x,t))_x = 0 \tag{2.1}$$

with appropriate initial and boundary conditions.[1] The main difficulty in solving (2.1) is due to the formation of shock discontinuities, which will occur even when the initial conditions are smooth. To retain the proper shock speed, numerical solvers for (2.1) must be written in flux conserving form [25, 24, 18]. Specifically, if the spatial domain is discretized as x_j, $j = 0, \ldots, N$, and time is incremented at $t = t_l$, numerical solvers for the interior of the interval should be of the form

$$\int_{x_j}^{x_{j+1}} u(x, t_{l+1})dx = \int_{x_j}^{x_{j+1}} u(x, t_l)dx + \int_{t_l}^{t_{l+1}} f(u(x_j, t))dt - \int_{t_l}^{t_{l+1}} f(u(x_{j+1}, t))dt. \tag{2.2}$$

In this investigation we seek to approximate the solution $\mathbf{u}(t) = \{\bar{u}_j(t)\}_{j=1}^N$ for $t \in (0, T)$, where $\bar{u}_j(t)$ is the cell average over a uniform grid $\{x_j\}_{j=1}^N$ given by

$$\bar{u}_j(t) = \int_{x_j - \frac{\Delta x}{2}}^{x_j + \frac{\Delta x}{2}} u(x,t)dx, \quad j = 1, \ldots, N, \quad \Delta x = \frac{b-a}{N}. \tag{2.3}$$

Studies regarding numerical conservation laws typically assume the flux term is known, with the goal to construct accurate, robust, and efficient solvers for $\mathbf{u}(t)$ by appropriately discretizing (2.2). Here, by contrast, we are interested in the case where we know apriori that the governing equation is a conservation law, but the flux function itself is unknown. The goal is then to determine how the solution will evolve given some early observations regarding the governing PDE. We will exploit our understanding of conservation laws by designing our numerical method to be in conservative form, as is given by (2.2). Once we are able to construct the numerical flux, it can be used to predict the evolution of the unknown PDE.

We now introduce some notation for the observable data. We will assume that the solution to the PDE is available at a set of discrete time instances $\{t_l\}_{l=1}^L$ for N_{traj} initial conditions resulting in so-called snapshots of the solution,

$$\mathbf{u}^{(k)}(t_l), \quad l = 1, \ldots, L, \quad k = 1, \ldots, N_{traj}. \tag{2.4}$$

The superscript k in (2.4) denotes the k-th "trajectory", which implies all L snapshots are evolved from the same initial state, with N_{traj} denoting the total number of trajectories. The N_{traj} initial conditions in our experiments are chosen by perturbing

[1] We will consider systems of conservation laws in our numerical experiments.

the true initial conditions of the PDE. The time step between two consecutive time instances is given by

$$\Delta t = t_{l+1} - t_l, \quad l = 1, \dots L-1,$$

and for simplicity we assume Δt is constant so that $t_l = l\Delta t$. Our (temporal) training domain is therefore given by

$$\mathcal{D}_{train} = [0, t_L] = [0, L\Delta t]. \tag{2.5}$$

We note that in this investigation we are interested in model prediction after time t_L. Our framework may also be used for learning some previous behavior, for instance at time $(n+\eta)\Delta t$, $n < L$ and $\eta \in (0,1)$. This would be especially useful for cases when the time difference Δt is very large.

2.2. Flow map-based dynamics learning. We now briefly review flow map-based deep learning of system dynamics for ordinary differential equations (ODEs) first proposed in [34], which will serve as a starting point for our flux learning technique. To this end, we consider the dynamical system given by

$$\frac{d\mathbf{u}}{dt} = g(\mathbf{u}), \qquad \mathbf{u} \in \mathbb{R}^N, \tag{2.6}$$

where $\mathbf{u} = u(x_j, t)$ and $g(\mathbf{u}) = -\frac{\partial f}{\partial x}(u(x,t))|_{x=x_j}$, $j = 0, \dots, N$. The fundamental distinction between the problem formulation in [34] and the problem in this investigation is that here we are considering a conservation law PDE model instead of a nonlinear system of ODEs. In either case, although we can observe snapshots of the solution $\mathbf{u}$, the flux function f in (2.1) and correspondingly g in (2.6) are unknown.

The flow map of (2.6), $\mathbf{\Phi} : \mathbb{R}^N \times \mathbb{R} \to \mathbb{R}^N$, characterizes its dynamics by mapping the state variable at current time $t = 0$ into a future state after some time Δt so that $\mathbf{u}(\Delta t) = \mathbf{\Phi}(\mathbf{u}(0), \Delta t)$. The main idea in [34] is to use a deep neural network to approximate the unknown flow map $\mathbf{\Phi}$ from the observation data.

The flow map-based dynamics deep learning approach begins by regrouping the observed data in (2.4) into pairs of adjacent time instances,

$$\{\mathbf{u}^{(m)}(0), \mathbf{u}^{(m)}(\Delta t)\}, \quad m = 1, \dots, M, \tag{2.7}$$

where M is the total number of such data pairs. Then a standard fully connected feed-forward deep neural network whose input and output layers both have N neurons is constructed. Specifically, we let $\mathcal{N} : \mathbb{R}^N \to \mathbb{R}^N$ be the associated mapping operator and define the residual network (ResNet) mapping as (see [17])

$$\mathbf{y}^{out} = [\mathcal{I} + \mathcal{N}]\left(\mathbf{y}^{in}\right), \tag{2.8}$$

where $\mathcal{I} : \mathbb{R}^N \to \mathbb{R}^N$ is the identity operator. Using the data set in (2.4) and setting $\mathbf{y}^{in} \leftarrow \mathbf{u}^{(m)}(0)$ and $\mathbf{y}^{out} \leftarrow \mathbf{u}^{(m)}(\Delta t)$, we obtain a network model

$$\mathbf{u}_{NN}^{(m)}(\Delta t; \Theta) = \mathbf{u}^{(m)}(0) + \mathcal{N}(\mathbf{u}^{(m)}(0); \Theta). \tag{2.9}$$

The network operator $\mathcal{N}$ can be trained by minimizing the mean square loss function,

$$\mathcal{L}(\Theta) = \frac{1}{M} \sum_{m=1}^{M} \|\mathbf{u}_{NN}^{(m)}(\Delta t; \Theta) - \mathbf{u}^{(m)}(\Delta t)\|_2^2, \tag{2.10}$$

where Θ denotes the network parameter set. Once the network is satisfactorily trained we can obtain a predictive model for any arbitrarily given initial condition $\mathbf{u}(t_0)$. Network structure variations and network theoretical properties can be found in [34]. In [33], the flow-map based learning was extended to include variable time stepping as well as other system parameters. Systems with missing variables were discussed in [12].

2.3. Learning PDE dynamics. To provide more general context for our new method, we briefly describe how the flow-map idea used for ODEs in subsection 2.2 can be extended to learn PDE dynamics, [8, 44], although we do not use this approach directly in our current investigation. In particular, our construction of the predictive model in (3.7) is equivalent to using the method of lines and then learning the flow-map of the discretized PDE dynamics. Importantly, this occurs *after* we learn the numerical flux, which is the central idea to our method.

The focus in [8] was on general deep neural networks (DNN), resulting in the development of a network structure for modeling non-specific types of unknown PDEs. The network structure is based on a user specified numerical PDE solver and consists of a set of multiple disassembly layers and one assembly layer[2] used to model the hidden spatial differential operators in the unknown PDE. The DNN model used in [8] defines the mapping

$$\mathbf{u}_{NN}(\Delta t) = \mathbf{u}(0) + \mathcal{A}(\mathcal{F}_1(\mathbf{u}(0)), \ldots, \mathcal{F}_J(\mathbf{u}(0)), \tag{2.11}$$

where $\mathcal{F}_1, \ldots, \mathcal{F}_J$ are the NN operators for the disassembly layers and $\mathcal{A}$ is the NN operator for the assembly layer. The mapping in (2.11) can be viewed as an application of ResNet (2.8) with $\mathcal{N} = \mathcal{A} \circ (\mathcal{F}_1, \ldots, \mathcal{F}_J)$. The same training process and loss function (2.10) for ResNet (2.8) can then also be applied to (2.11). After satisfactory training, given any new initial condition, predictions can be made by iteratively applying (2.11).

REMARK 2.1. *The method in [8], which directly applies (2.11), also seeks to learn PDE dynamics from data. A primary motivation for the method given there is to be able to consider environments where data are collected on structure-free grids. It is not guaranteed to capture the correct shock propagation speed for systems governed by conservation laws, however. Moreover, the assembly and disassmebly layers in (2.11) require a considerable amount of hand-tuning. This investigation, by contrast, is interested in learning the dynamics of hyperbolic conservation laws by incorporating the form (2.2) directly into the neural network. We also assume a structured grid of data, allowing the use of standard neural network structures (fully connected feed-forward networks).*

3. Constructing the network. Given trajectory data in (2.4), we now seek to construct a neural network $\mathcal{N}$ that learns the evolution of the underlying system. More precisely, we want $\mathcal{N}$ to learn to predict the state value $\mathbf{u}(t_{l+1})$ from the current state value $\mathbf{u}(t_l)$. In Section 3.2 we describe our approach for designing the network $\mathcal{N}$ for a system that is known to be conservative but for which the flux is unknown. The more traditional approach for constructing a network without considering its conservation properties is first reviewed in Section 3.1.

[2] We make use of the terms assembly and disassembly layers from [8] to help visualize the network architecture, and note that they are fully-connected.

3.1. Standard (non-conservative) form network (nCFN). A standard approach is to use a deep neural network $\mathcal{G}$ to approximate $f(u(x,t))_x$ in (2.1) directly for each cell average $\bar{u}_j(t^n)$ as

$$\mathcal{G}(u^n_{j-p},\dots,u^n_j,\dots,u^n_{j+q}) \approx f(\bar{u}_j(t^n))_x \tag{3.1}$$

where $u^n_j = \bar{u}_j(t_n)$, $p \geq 0, q \geq 0$, and then solve the ODE given by

$$\frac{d}{dt}u_j + \mathcal{G}(u^n_{j-p},\dots,u_j,\dots,u_{j+q}) = 0, \tag{3.2}$$

with some pre-determined time integration technique. Importantly, (3.1), which we will refer to as the non-conservative form network (nCFN), *does not* account for conservation in its design. As will be demonstrated in Section 5, the nCFN is not able to capture the dynamics of the solution $u(x,t)$ for t that extends beyond the training domain, given by $\mathcal{D}_{train}$ in (2.5). In particular using the nCFN may result in a numerical solution that yields the wrong shock speed.

A typical approach to embed the conservation property into the network model is to add a regularization term to the loss function [11, 15, 36]. Denoting the magnitude of the conserved quantity remainder in the system as $C(\mathbf{u})$, which we will derive in discrete form in Section 4.3, the regularization term can be constructed as

$$\mathcal{R}(\Theta) = \sum_{l=1}^{L} C(\mathbf{u}_{NN}(t_l;\Theta))^2. \tag{3.3}$$

Here Θ denotes the network parameter set. In this way, the regularization term $\mathcal{R}(\Theta)$ penalizes the remainder of each conserved quantity in the network prediction. We will refer to the approach of regularizing the nCFN with the loss function (3.3) as nCFN-reg. Our numerical examples will demonstrate that the solutions resulting from a non-conservative neural network are still unsatisfactory even when incorporating the regularization term, especially in long-term prediction.

3.2. Conservative form network (CFN). Motivated by classical results in numerical conservation laws, we propose a flux form network that seeks to preserve the conservation property. Specifically, we seek to update the cell average $\bar{u}_j(t_n)$ using the flux differences at the cell edges as

$$\frac{d}{dt}\bar{u}_j + \frac{1}{\Delta x}\Big(f_{j+1/2} - f_{j-1/2}\Big) = 0, \tag{3.4}$$

where $f_{j+1/2}$ denotes the flux at the cell edge $x = x_{j+1/2}$. To approximate $f_{j+1/2}$ we define the *neural flux* as

$$f^{NN}_{j+1/2} = \mathcal{F}(\bar{u}^n_{j-p},\dots,\bar{u}_j,\dots,\bar{u}_{j+q}), \tag{3.5}$$

where $\mathcal{F}$ is a fully connected feed-forward neural network operator and the inputs $\bar{u}^n_{j-p},\dots,\bar{u}_j,\dots,\bar{u}_{j+q}$ are neighboring cell averages centered at $x = x_{j-p},...,x_{j+q}$, respectively. For ease of presentation we denote the right hand side of (3.5) as $\mathcal{F}_{p,q}(\bar{u}_j)$. Our implementation is also simplified by using a symmetric stencil around $x_{j-1/2}$, so that $p = q-1$, although this is not required. To distinguish between the non-conservative flux forms, nCFN and nCFN-reg, we will refer to the conservative flux form network as CFN when discussing our numerical experiments.

3.3. Time integration. With the neural flux (3.5) in hand we now write the neural net form of (3.4) as

$$\frac{d}{dt}\bar{u}_j + \frac{1}{\Delta x}\Big(\mathcal{F}_{p,q}(\bar{u}_j) - \mathcal{F}_{p,q}(\bar{u}_{j-1})\Big) = 0. \tag{3.6}$$

We solve (3.6) as well as the non-conservative system in (3.2) using the method-of-lines approach. In our examples we will use the total variation diminishing Runge-Kutta (TVD-RK3) method, [14], which for the generic system of ODEs in (2.6) is given by

$$\begin{aligned}
\mathbf{u}^{(1)} &= \mathbf{u}^n + \Delta t \mathcal{F}(\mathbf{u}^n),\\
\mathbf{u}^{(2)} &= \frac{3}{4}\mathbf{u}^n + \frac{1}{4}\mathbf{u}^{(1)} + \frac{1}{4}\Delta t \mathcal{F}(\mathbf{u}^{(1)}),\\
\mathbf{u}^{n+1} &= \frac{1}{3}\mathbf{u}^n + \frac{2}{3}\mathbf{u}^{(2)} + \frac{2}{3}\Delta t \mathcal{F}(\mathbf{u}^{(2)}),
\end{aligned} \tag{3.7}$$

for integration between time steps n and $n+1$. Figure 1 provides a diagram showing the evolution of the state variable $\mathbf{u}$ for one time step through the conservative neural network model.

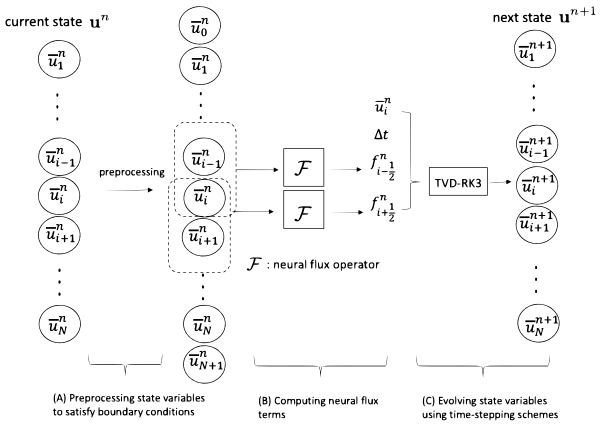


Fig. 1: Evolution of the state variable $\mathbf{u}$ for one time step through neural network model: State variables are fed into a preprocessing layer to incorporate boundary conditions. The model then computes neural flux terms for each cell edge. The flux terms and state variables are fed into the time-integration method.

REMARK 3.1. *To the best of our knowledge there are no theoretical results regarding the choice of time integration schemes that guarantee stability for numerical PDEs using neural networks. Thus, due to its theoretical stability guarantees for numerical conservation laws when using traditional solvers, we choose the TVD-RK3 method. Other time integration techniques may also be appropriate, and in some cases reduce computational cost or improve numerical accuracy.*

3.4. Boundary Conditions. For simplicity we will assume that the boundary conditions in (2.1) are known. In particular, to satisfy the periodic boundary conditions in Example 5.1 we simply apply

$$\bar{u}_0 = \bar{u}_N, \quad \bar{u}_{N+1} = \bar{u}_1. \tag{3.8}$$

No flux boundary conditions are assumed in Examples 5.3 and 5.4. Since the solution profiles near the boundaries remain constant over time, we simply impose the boundary conditions for each variable in both examples as

$$\bar{u}_0 = \bar{u}_1, \quad \bar{u}_{N+1} = \bar{u}_N. \tag{3.9}$$

Higher order numerical boundary conditions can similarly be employed.

3.5. The Recurrent Loss Function. Given trajectory data in (2.4), where each trajectory has multiple measurements, we define the *recurrent loss function* as

$$\mathcal{L}_{RNN}(\Theta) = \frac{1}{N_{traj}} \frac{1}{L} \sum_{k=1}^{N_{traj}} \sum_{l=1}^{L} \|\mathbf{u}_{NN}^{(k)}(t_l;\Theta) - \mathbf{u}^{(k)}(t_l)\|_2^2, \tag{3.10}$$

where

$$\mathbf{u}_{NN}^{(k)}(t_l;\Theta) = \underbrace{\mathcal{N} \circ \cdots \circ \mathcal{N}}_{l \text{ times}}(\mathbf{u}^{(k)}(t_0)). \tag{3.11}$$

As already discussed, the network evolution operator $\mathcal{N}$ is designed to predict the state value $\mathbf{u}(t_{l+1})$ from the current state value $\mathbf{u}(t_l)$, where Θ again denotes network parameter set. In contrast to (2.10), the recurrent loss function (3.10) calculates loss over multiple time steps. Using the recurrent loss approach has been found to improve numerical stability [8]. As mentioned in Section 3.1, one can embed the conservation property into the network by adding a regularization term in the loss function,

$$\mathcal{L}_{RNN}^{\lambda}(\Theta) = \mathcal{L}_{RNN}(\Theta) + \lambda^2 \mathcal{R}(\Theta), \tag{3.12}$$

where $\mathcal{R}$ is defined in (3.3). Following ML conventional notation we use λ^2 to denote the weighting parameter for the regularization term.

4. Experiment design. Our results presented in Section 5 will demonstrate that for the classical one-dimensional conservation laws, [24] studied in this investigation, using the CFN, for which the network learns to approximate the numerical flux function of the unknown conservation law via (3.5) from observation data, yields significantly better results than those obtained using either the nCFN in (3.1) or the nCFN regularized by the loss function, nCFN-reg. Below we provide the framework necessary to ensure the robustness and reliability of our experimental results. To this end we consider details related both to data collection and training.

4.1. Data Collection. To test our method we will consider examples of conservation laws for which the fluxes are known.[3] We will use this information both to generate synthetic training data with which to train the DNN for the evolution process as well as to compute reference solutions to evaluate our results. Importantly we note that knowledge of the true system does not in any way facilitate the DNN model approximation.

Our test problems range from idealistic, where we assume we have noise-free densely observed data for training and validation, to more difficult situations, where we consider two cases: (1) the observable data are accurate (noiseless) but sparsely observed and (2) the observable data are noisy but densely observed. To mimic observed

[3]Indeed in some of our examples the true solution is also known. However, since we randomly generate the initial conditions to obtain a set of N_{traj} snapshots, we will simply consider the "exact" (reference) solution to be the highly resolved numerical result.

data in (2.4) that would be available for training and validation, we numerically simulate the true underlying PDE model according to the observational settings provided in the particular case study for our examples.[4] We furthermore randomly sample the parameters in the initial conditions to obtain various trajectories of the observed data (see e.g. (5.3)). The number of trajectories N_{traj} and the trajectory length L vary depending on the underlying properties of the PDE (e.g. time to shock formation).

4.2. Network and Training Details. As shown in Figure 1, the CFN consists of the preprocessing layer, which ensures that the boundary conditions are satisfied, and the neural flux operator, which computes the flux at cell edges. The neural flux operator is constructed using a fully connected feedforward neural network and is obtained by training the network hyperparameters (weights and biases) as the minimum of the recurrent loss function (3.10). We employ the stochastic optimization method **Adam** [22] for this purpose. The nCFN and nCFN-reg utilize the same preprocessing layer for the boundary conditions and each employs one fully connected feedforward network to learn the increment of the state variables. For consistency all models are trained for 10,000 epochs with learning rate 10^{-4} in every example. The same set of network structures is also employed. Finally, we use the commonly chosen Rectified Linear Unit (ReLU) [23] as the activation function. These network and training details are summarized in Table 1.

model	(p,q)	hidden layers	hidden nodes	activations
CFN	(2,3)	5	64	ReLU
nCFN	(3,3)	5	64	ReLU
nCFN-reg	(3,3)	5	64	ReLU

Table 1: Neural network architecture details for all examples. Note that for each model p and q are chosen to provide symmetry ($p = q - 1$ for CFN and $p = q$ for nCFN), although this is not a requirement.

It is also possible to tune the regularization parameter λ^2 for the nCFN-reg loss function in (3.12). Indeed, one can choose $\lambda^2 = \lambda^2(t)$, so that the influence of the regularization can fluctuate as the PDE evolves. This would add considerable computational cost, however, and moreover, it is not readily apparent that employing standard approaches, such as the $L-$curve method or the discrepancy principle, [16], are appropriate here. For simplicity, here we let $\lambda_i^2 = 10^{2(1-i)}$, $i = 1, \ldots, 4$, and then choose λ^2 to be the λ_i^2 corresponding to the smallest loss value in (3.12) on a separate validation dataset after training is completed. In general we found that in Examples 5.1 and 5.3 that $\lambda^2 = 10^{-2}$ ($i = 2$) yielded the best results. Example 5.4 (the Euler equations for gas dynamics) was considerably more sensitive to the choice of λ^2, likely due to the oscillatory nature of the solution. In this case we refined our search to include $\lambda_5^2 = 5 \times 10^{-2}$. We therefore see that as an added advantage our new CFN approach does not require extensive regularization parameter tuning.

We emphasize that while our numerical experiments indicate that these parameter choices provide enough network complexity for each required learning task, we did not further try to optimize performance. Moreover, as we want to ensure the robustness of our method, in our experiments we typically follow the common practice for learning

[4] Unless otherwise noted we use the CLAWPACK conservation laws package, [10].

system dynamics [34, 32, 33, 44] and use the default values in Tensorflow [1] or other standard choices for all hyperparameters.

4.3. Constructing the Regularization Term. The regularization term (3.3) is designed to promote conservation in the nCFN-reg method. Below we show how this term is constructed for the scalar case. A straightforward extension can be made for systems.

We first expand (2.2) to the physical domain of the problem, (a, b), yielding

$$\int_a^b u(x,t_{l+1})dx - \int_a^b u(x,t_l)dx = \int_{t_l}^{t_{l+1}} f(u(a,t))dt - \int_{t_l}^{t_{l+1}} f(u(b,t))dt, l = 0,\dots,L, \tag{4.1}$$

where each t_l denotes the time at which a data trajectory in (2.4) is initially obtained.

Example 5.1 considers the inviscid Burgers equation with periodic boundary conditions. In this case (4.1) simplifies to

$$\int_a^b u(x,t_l)dx = \int_a^b u(x,t_0)dx, \quad l = 0,\dots,L. \tag{4.2}$$

For equations with non-periodic boundary conditions, (4.2) does not hold since in general $f(u(a,t)) \neq f(u(b,t))$. Hence to construct (3.3) we first define

$$F_a^{(l)} = \frac{1}{\Delta t}\int_{t_l}^{t_{l+1}} f(u(a,t))dt, \quad F_b^{(l)} = \frac{1}{\Delta t}\int_{t_l}^{t_{l+1}} f(u(b,t))dt, \quad l = 0,\dots,L-1, \tag{4.3}$$

and then use (4.3) to approximate (4.1) as

$$\sum_{j=1}^{N} \bar{u}_j(t_{l+1})\Delta x - \sum_{j=1}^{N} \bar{u}_j(t_l)\Delta x = F_a^{(l)}\Delta t - F_b^{(l)}\Delta t, \quad l = 0,\dots L-1, \tag{4.4}$$

which leads to

$$\sum_{j=1}^{N} \left(\bar{u}_j(t_l) - \bar{u}_j(t_0)\right)\Delta x = \sum_{k=1}^{l} \left(F_a^{(k-1)} - F_b^{(k-1)}\right)\Delta t, \quad l = 1,\dots,L. \tag{4.5}$$

From here we define the (discrete) conserved quantity remainder at each t_l as

$$C(\mathbf{u}(t_l)) := \left| \sum_{j=1}^{N} \left(\bar{u}_j(t_l) - \bar{u}_j(t_0)\right)\Delta x - \sum_{k=1}^{l} \left(F_a^{(k-1)} - F_b^{(k-1)}\right)\Delta t \right|, \tag{4.6}$$

where $\mathbf{u}(t) = (\bar{u}_1(t),\dots \bar{u}_N(t))^T$. It follows from (4.5) that if the conservation property holds then $C(\mathbf{u}(t_l)) = 0$. Regularization in (3.12) is therefore used to promote solutions that minimize (4.6). In practice the network prediction of $\mathbf{u}_{NN}(t_l;\Theta)$ is used to calculate (4.6), directly yielding $\mathcal{R}(\Theta)$ in (3.3). We note that we will also be able to analyze the conservation properties of each of our numerical methods in Section 5 by computing (4.6) over the time domain of the solution.

REMARK 4.1. *It is important to point out that* (4.6) *describes a best case scenario, where we have access to* (4.3). *In order to construct the regularization term for the nCFN-reg in our experiments, we compute* (4.3) *directly from the given flux terms in each example. This serves to demonstrate that even under ideal circumstances, regularizing the standard nCFN to promote conservation in the solution (nCFN-reg) is not as effective as constructing a conservative form network in the first place (CFN).*

5. Numerical Examples. We use three well-studied one-dimensional examples of hyperbolic conservation laws to analyze our new conservative form network (CFN) approach. We consider three different observational settings for each experiment: (i) an ideal case, where the observations are dense and noise-free; (ii) the situation where the observations are sparse but noise-free; and (iii) an environment for which the observations are dense but noisy. We compare the results of our new CFN approach to the more traditional non-conservative form network (nCFN) along with the regularized (nCFN-reg) version.

5.1. Inviscid Burgers Equation. Due to its simple formulation, the inviscid Burgers equation is often used to test the efficacy of numerical methods for non-linear conservation laws. Here we demonstrate our method for two cases; In subsection 5.1.1 a single shock is formed from smooth initial conditions while in subsection 5.1.2 the initial condition contains two discontinuities that subsequently collide and interact.

5.1.1. Single Shock Formation.

EXAMPLE 5.1. *The inviscid Burgers equation is given by*

$$u_t + (\frac{u^2}{2})_x = 0, \quad x \in (0, 2\pi), \quad t > 0, \tag{5.1}$$

with periodic boundary conditions $u(0,t) = u(2\pi,t)$. *The initial condition are given by*

$$\begin{aligned} u(x,0) &= \alpha + \beta \sin(x), \\ \alpha &\sim U[-\epsilon_s, \epsilon_s], \\ \beta &\sim U[1-\epsilon_s, 1+\epsilon_s], \end{aligned} \tag{5.2}$$

where $\epsilon_s = 0.25$.

The N_{traj} training data sets are generated by solving (5.1) using the Engquist–Osher flux along with TVD-RK3 time integration based on the initial conditions

$$\begin{aligned} u^{(k)}(x,0) &= \alpha^{(k)} + \beta^{(k)} \sin(x), \\ \alpha^{(k)} &\sim U[-\epsilon_s, \epsilon_s], \\ \beta^{(k)} &\sim U[1-\epsilon_s, 1+\epsilon_s] \end{aligned} \tag{5.3}$$

for $k = 1, \ldots, N_{traj}$ with $\epsilon_s = 0.25$.

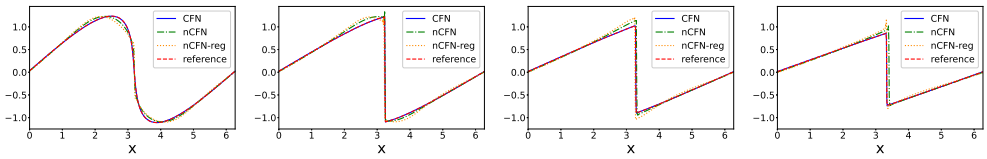
In all of our experiments we set $N_{traj} = 200$. Each training trajectory has length $L = 20$ for either choice of recurrent loss function, (3.10) or (3.12).

To check the *robustness* of our predictions for Example 5.1, we run our experiments for 50 choices of fixed α and β and compare the three methods, CFN, nCFN, and nCFN-reg. Our reference solution is calculated using the Engquist-Osher flux term on a fine grid, with $\Delta x = \frac{2\pi}{1024}$ in (3.6). All figures use the initial value with $\alpha = 0.06342$ and $\beta = 1.17322$ for illustration. Other choices for α and β yield comparable results.

Case I: Dense and noise-free observations. We first consider an idealized environment for which the observations are dense and noise-free. Specifically we choose $N = 512$, yielding $\Delta x = \frac{2\pi}{512}$, so that our solution is well-resolved. We also choose a constant time step Δt for all experiments so as not to complicate our analysis. In this regard we observe that the maximum wave speed for Burgers equation, $|u(x,t)|$, can be determined for all t using (5.3) as $\max\{|u|\} = 1 + 2\epsilon_s$. We therefore set $\Delta t = 0.005$ to satisfy the CFL condition with $\#CFL = 0.9$.

We note that the training time domain $[0, L\Delta t]$ with $L = 20$ contains only smooth solution snapshots. Since each DNN model requires the training data to include both smooth and discontinuous solution profiles to learn the long term dynamics of Example 5.1, a larger trajectory length L is needed. As choosing a larger L would significantly increase computational costs we employ a sub-sampling technique to generate training data from observed snapshots of the solution onto an extended domain. The same sub-sampling technique is used for Examples 5.2 and 5.4. The details are provided below.

We define a new parameter $M > L$ as the extended length of each trajectory. The snapshots of the solution, (2.4), are obtained for each of the N_{traj} trajectories at times $t = m\Delta t$, $m = 1, \ldots, M$. In our experiments we choose $M = 300$ which yields the total training time domain as $[0, 1.5]$. We then sub-sample each of the $k = 1, \ldots, N_{traj}$ by randomly selecting a start time value, $t_0^{(k)}$, from the set $\{\mu\Delta t\}_{\mu=0}^{M-L}$. Each sub-sampled trajectory of length L is then built consecutively from the snapshot solutions. That is, each trajectory is comprised of the solutions in (2.4) at sequential times $t_0^{(k)} + l\Delta t$, $l = 1, \ldots, L$. In this way we can train over a longer period of time without increasing the expense of network training. This approach, of course, requires that more initial observations are available.



(a) $t = 0.75$ (b) $t = 1.5$ (c) $t = 2.25$ (d) $t = 3$

Fig. 2: Comparison of the reference solution to Example 5.1 with the trained DNN model predictions at different times for dense ($N = 512$) and noise-free observations.

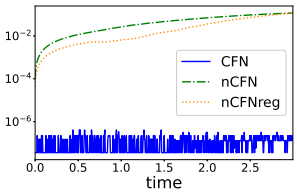
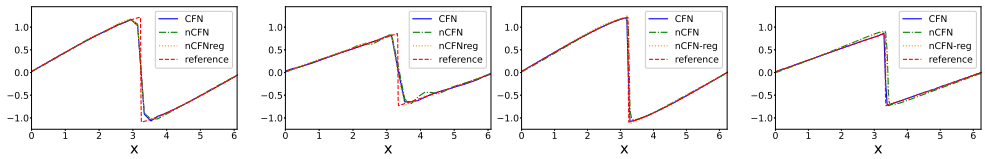


Fig. 3: Discrete conserved quantity remainder $C(\mathbf{u})$ in (4.6) of the network predictions for Example 5.1. Graph is in semi-log scale for visibility.

Figure 2 presents the solution to Example 5.1 for this ideal case at four different times, within and beyond training time domain $[0, 1.5]$. Observe that the three methods capture the solution profiles and predict the correct shock propagation speed within the training domain (shown for $t = 0.75$ and $t = 1.5$). The nCFN and nCFN-reg results are less accurate, and do not appear to be completely resolved. Beyond the training domain ($t > 1.5$), only the CFN and nCFN-reg methods yield the correct shock propagation speed (Figure 2c, Figure 2d). The nCFN-reg solution develops a

non-physical overshoot near the shock location. This behavior is further observed in Figure 3, where the conserved quantity remainder $C(\mathbf{u})$ obtained by (4.6) is displayed for each method. Clearly the CFN produces the only conservative method.

REMARK 5.1. *We also compared our results to those obtained using the method in [8] which has a global design and beyond the fully-connected layers also contains additional disassembly and assembly layers. This structure inherently means that the method has significantly more parameters to tune and also requires more training when compared to our CFN approach, which has a local flux structure. In particular the set of training data provided in all of our case studies, including the idealized environment, leads to overfitting in the training process and fails to yield conservation. Additional training data will lead to more comparable results, although there is no guarantee that they will ultimately yield the correct shock speed of propagation. The solution may furthermore exhibit non-physical oscillations near the shock.*[5]



(a) $N = 32$, $t = 1.5$ (b) $N = 32$, $t = 3$ (c) $N = 128$, $t = 1.5$ (d) $N = 128$, $t = 3$

Fig. 4: Comparison of the reference solution to Example 5.1 with the trained DNN model predictions at different sparsity levels using noise-free observations.

Case II: Sparse and noise-free observations. In this case the training data are obtained by solving Example 5.1 on a coarse grid. Specifically, for $\Delta x = \frac{2\pi}{N}$ we choose $N = 32, 128$. Once again we fix the time step as $\Delta t = 0.005$. Figure 4 compares the results using the CFN, nCFN and nCFN-reg for different sparsity levels at times in $(t = 1.5)$ and out of $(t = 3)$ the training time domain. Both time instances are after the shock forms. Observe that for each choice of N only the CFN captures the correct shock propagation speed. Figure 5 displays the pointwise error at different sparsity levels for each method when $t = 3$. It is apparent that the width of the interval containing the error resulting from shock shrinks (as expected) with increased resolution for all three methods. However, neither the nCFN nor the nCFN-reg demonstrate convergence.

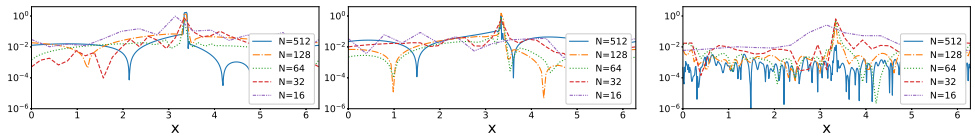
Case III: Dense and noisy observations. In this testing environment the observations in (2.4) now contain noise and are given by

$$\tilde{\mathbf{u}}^{(k)}(t_l) = \mathbf{u}^{(k)}(t_l) + \boldsymbol{\epsilon}_l^{(k)}, \quad l = 1, \ldots, L, \quad k = 1, \ldots, N_{traj}. \tag{5.4}$$

Here $\boldsymbol{\epsilon}_l^{(k)}$ is i.i.d. Gaussian with zero mean and variance σ^2. We test various σ values scaled from the absolute value mean of the solution, $\overline{|u|}$,

$$\sigma = a\overline{|u|}, \quad a \geq 0 \tag{5.5}$$

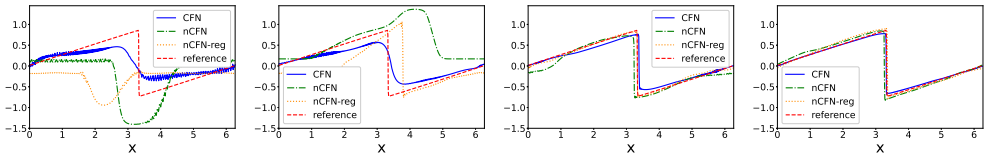
[5] We note that a primary motivation in [8] is to learn the dynamics of generic PDEs on unstructured grids, while the data in our examples are collected on structured grids.



(a) nCFN (b) nCFN-reg (c) CFN

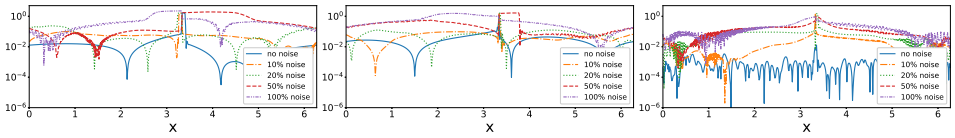
Fig. 5: Log-scale absolute error of the trained DNN model predictions to Example 5.1 at different sparsity levels when $t = 3$. No observation error.

where the mean $\overline{u}$ is taken over the spatiotemporal domain. We consider noise levels of 100% , 50%, 20% and 10%, that correspond to $\alpha = 1, 0.5, 0.2$, and 0.1 respectively.



(a) 100% noise (b) 50% noise (c) 20% noise (d) 10% noise

Fig. 6: Comparison of the reference solution to Example 5.1 with the trained DNN model predictions at different noise levels using dense ($N = 512$) observations when $t = 3$.



(a) nCFN (b) nCFN-reg (c) CFN

Fig. 7: Log-scale absolute error of the trained DNN model predictions to Example 5.1 using different noise levels using dense ($N = 512$) observations when $t = 3$.

The solution to Example 5.1 is presented in Figure 6 at time $t = 3$ after the shock forms. Learning the underlying dynamics is challenging when the observations are noisy since the non-physical oscillatory behavior caused by the noise can influence the training process (overfitting). Indeed Figure 6a demonstrates that none of the solutions corresponding to any of the three training networks can capture the shock in high noise environments. However, the CFN method is the only network that captures the rough profile of the underlying solution. In contrast, solutions resulting from both nCFN and nCFN-reg deviate significantly from the reference solution. While all methods improve as the amount of noise decreases, CFN and nCFN-reg

yield overall better results. In Figure 6c and Figure 6d, we observe that CFN and nCFN-reg capture the correct shock propagation speed (with some magnitude error). Figure 7 displays the pointwise errors for each of the methods at time $t = 3$, which are consistent to what is observed in Figure 2d. That is, even in the "ideal" case, neither the nCFN nor the nCFN-reg can be adequately resolved. Adding small amount of noise which is comparable to the error already incurred therefore does not affect the results. For the same reason, small amounts of noise can reduce the accuracy in the CFN case (since it is larger than the error produced for Case I). As noise is increased, the results for the nCFN and nCFN-reg method become meaningless – $\mathcal{O}(1)$ in much of the domain. The largest interval width of error surrounding the discontinuity is again seen in the nCFN case, which concurs with the results shown in Figure 6. Figures comparing the discrete conserved quantity remainder, (4.6), of each method are omitted for Cases II and III since the methods all generate the same general behavior pattern as what is shown for Case I in Figure 3.

5.1.2. Multiple Shock Interaction.

EXAMPLE 5.2. *We again consider the inviscid Burgers equation in* (5.1) *with periodic boundary conditions. Here our initial conditions are given by*

$$u(x,0) = \begin{cases} 0.8, & x \in [0, 2.5) \cup [4.5, 2\pi], \\ -0.1 & x \in [2.5, 3.5), \\ -0.7 & x \in [3.5, 4.5). \end{cases} \tag{5.6}$$

Observe that in contrast to the solution for Example 5.1 in which a smooth initial condition later forms a shock, here the initial shocks will eventually collide and interact, with the solution forming a rarefaction wave.

As was done in Example 5.1, we generate $N_{traj} = 200$ training data sets by numerically solving (5.1) with CLAWPACK using the Engquist-Osher flux with $N = 1024$ so that $\Delta x = \frac{2\pi}{1024}$, along with TVD-RK3 time integration and $\Delta t = .005$ chosen to satisfy the CFL condition.

For this example the initial conditions used for training are given by

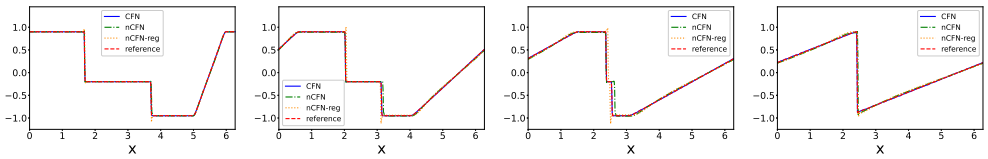
$$u^{(k)}(x,0) = \begin{cases} u_1^{(k)}, & x \in [\min\{y_1^{(k)}, y_2^{(k)}\}, \max\{y_1^{(k)}, y_2^{(k)}\}], \\ u_2^{(k)}, & \text{else}, \end{cases} \tag{5.7}$$

where $y_1^{(k)}, y_2^{(k)} \sim U[0, 2\pi]$, $u_1^{(k)}, u_2^{(k)} \sim U[-1, 1]$ for $k = k_1, \ldots, N_{traj}$. The form of (5.7) represents what might be included in the space of initial conditions, but importantly does not consider any future information regarding how the solution evolves.

As in Example 5.1, we again employ sub-sampling to generate training data from the observed snapshots of the solution on an extended domain, with the same length for extended trajectory $M = 300$. The sub-sampling is necessary since otherwise the short trajectories used for training do not contain enough information to capture the solution containing rarefaction waves occurring in the extended time domain (see Figure 8).

Case I: Dense and noise-free observations. We first consider an idealized environment for which the observations are dense and noise-free. Specifically we choose $N = 512$, yielding $\Delta x = \frac{2\pi}{512}$, so that our solution is well-resolved.

Figure 8 displays the solution for this ideal case at four different times, illustrating the collision of two shocks and the propagation of a rarefaction wave. While all



(a) $t = 0.5$ (b) $t = 1.5$ (c) $t = 2.5$ (d) $t = 3.5$

Fig. 8: Comparison of the reference solution to Example 5.2 with the trained DNN model predictions at different times for dense ($N = 512$) and noise-free observations.

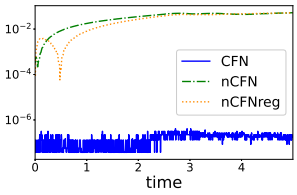
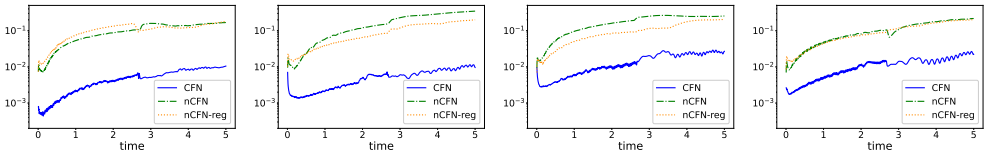


Fig. 9: Discrete conserved quantity remainder $C(\mathbf{u})$ in (4.6) of the network predictions for Example 5.2. Graph is in semi-log scale for visibility.

approaches are able to capture the general solution dynamics, only the CFN consistently predicts the correct shock speed both before and after the collision occurs. Moreover, non-physical overshoots are observed in the nCFN-reg solution near the shock location. Figure 9 shows the discreted conserved quantity remainder $C(\mathbf{u})$ in (4.6), demonstrating that the CFN is the only method maintaining conservation.



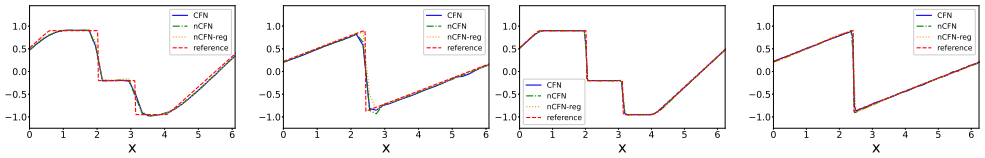
(a) (6,5,64) (b) (10,5,64) (c) (6,2,128) (d) (6,10,32)

Fig. 10: Relative l_2 error of the CFN, nCFN and nCFN-reg predictions using different hyperparameters. The hyperparameters are written in the sub-captions as (input stencil size, hidden layer number, hidden node number).

Figure 10 compares the robustness for the CFN with respect to various network parameters, such as the number of hidden layers, hidden nodes, and input stencil size $p + q$ (p, q as in equation (3.1)), to the nCFN and nCFN-reg. The relative l_2 error for each model prediction demonstrate that while the overall performance does not appear to be strongly influenced by the choice of hyperparameters, the CFN approach consistently achieves better accuracy.

Case II: Sparse and noise-free observations. As was done for Example 5.1, we choose $N = 32,128$ to simulate the sparse observation case with fixed time step

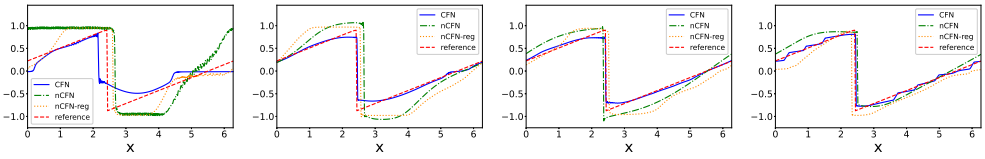
$\Delta t = 0.005$. Figure 11 compares the results using CFN, nCFN and nCFN-reg for different sparsity levels at time before ($t = 1.5$) and after ($t = 3.5$) shock collision. We observe that the lack of resolution ($N = 32$) similarly affects each method, with little difference in the solutions once the data are sufficiently resolved ($N = 128$). The results seen here seem to suggest that some overfitting occurs in Case I ($N = 512$) in both the nCFN and nCFN-reg solutions.



(a) $N = 32$, $t = 1.5$ (b) $N = 32$, $t = 3.5$ (c) $N = 128$, $t = 1.5$ (d) $N = 128$, $t = 3.5$

Fig. 11: Comparison of the reference solution to Example 5.2 with the trained DNN model predictions at different sparsity levels using noise-free observations.

Case III: Dense and noisy observations. We now consider the case where the observations are given by (5.4) with noise levels given by 100% , 50%, 20% and 10% respectively corresponding to $\alpha = 1, 0.5, 0.2$, and 0.1 in (5.5). As shown in Figure 12, none of the methods are able to predict the correct solution in the noisiest case. The situation dramatically improves as the noise level decreases to 50%. Although some error is apparent, the CFN consistently captures the correct shock speed and appears to have the best overall accuracy.



(a) 100% noise (b) 50% noise (c) 20% noise (d) 10% noise

Fig. 12: Comparison of the reference solution to Example 5.2 with the trained DNN model predictions at different noise levels for $N = 512$ observations at time $t = 3.5$.

Finally, we note that the discrete conserved quantity remainder calculated by (4.6) for both Case II and Case III exhibits the same qualitative behavior as seen in Figure 9.

5.2. Shallow water equations. When combined with initial conditions given in (5.9), Example 5.3 is known as the dam break problem which over time admits both shock and contact discontinuities.

EXAMPLE 5.3. *Consider the system of equations*

$$h_t + (vh)_x = 0,$$
$$(hv)_t + (hv^2 + \frac{1}{2}gh^2)_x = 0, \tag{5.8}$$

for $t > 0$ and $x \in (-5, 5)$. Here we use $g = 1$. We assume no flux boundary conditions and initial conditions given by

$$h(x,0) = \begin{cases} h_l, & \text{if } x \le x_0, \\ h_r, & \text{otherwise,} \end{cases} \qquad v(x,0) = \begin{cases} v_l, & \text{if } x \le x_0, \\ v_r, & \text{otherwise,} \end{cases} \tag{5.9}$$

where

$$h_l \sim U[2 - \epsilon_{h_l}, 2 + \epsilon_{h_l}], \quad \epsilon_{h_l} = 0.2,$$
$$h_r \sim U[1 - \epsilon_{h_r}, 1 + \epsilon_{h_r}], \quad \epsilon_{h_l} = 0.1,$$
$$v_l, v_r, x_0 \sim U[-\epsilon, \epsilon], \qquad \epsilon = 0.1.$$

Example 5.3 describes the one-dimensional dam break problem in which the initial heights of the water, h_l and h_r, are different on each side of the dam, located at x_0 in our numerical experiments. After the dam breaks, a rarefaction wave forms and travels to the left of the dam, while a shock wave starts to propagate on the right. The training data are observed at different time intervals up until time $t = 0.1$ and then used to train each of the three networks to predict the long term dynamics.

The $N_{traj} = 200$ training data sets of length $L = 20$ are generated by solving (5.8) using CLAWPACK (HLLE Riemann Solver) for initial conditions given by

$$h^{(k)}(x,0) = \begin{cases} h_l^{(k)}, & \text{if } x \le x_0^{(k)}, \\ h_r^{(k)}, & \text{otherwise,} \end{cases} \qquad v^{(k)}(x,0) = \begin{cases} v_l^{(k)}, & \text{if } x \le x_0^{(k)}, \\ v_r^{(k)}, & \text{otherwise,} \end{cases}$$

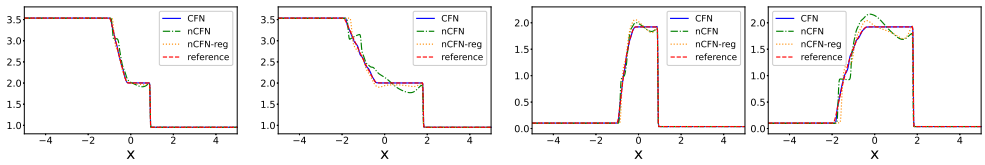
where

$$h_l^{(k)} \sim U[2 - \epsilon_{h_l}, 2 + \epsilon_{h_l}], \quad h_r^{(k)} \sim U[1 - \epsilon_{h_r}, 1 + \epsilon_{h_r}], \quad v_l^{(k)}, v_r^{(k)}, x_0^{(k)} \sim U[-\epsilon, \epsilon],$$

with $\epsilon_{h_l} = 0.2$, $\epsilon_{h_r} = .1$, $\epsilon = .1$ and $k = 1, \ldots, N_{traj}$. We obtain a reference solution using CLAWPACK using $N = 1024$ so that $\Delta x = \frac{10}{1024}$. All figures shown for Example 5.3 correspond to (5.9) with $h_l = 3.5691196, h_r = 1.17867352, v_l = -0.06466697, v_r = -0.04519738, x_0 = 0.00383271$. While some parameter choices yield comparable solutions for each method, the CFN consistently outperforms the other techniques.

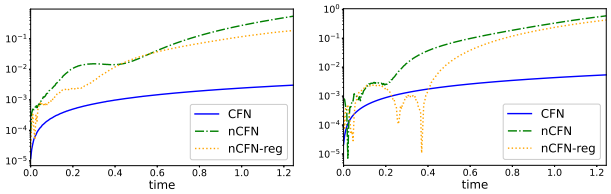
Case I: Dense and noise-free observations. In the ideal environment we set $N = 512$ so that $\Delta x = \frac{10}{512}$. We numerically impose the no flux boundary conditions using (3.9). CLAWPACK is employed to simulate solutions up to time $t = .1$ with data collections at time instances $t_l = l\Delta t$ for $l = 1, \ldots, 20$. The time step $\Delta t = 5 \times 10^{-3}$ is chosen to satisfy $\Delta t \le \min\{\Delta t_{CLAW}\}$, where CLAWPACK determines Δt_{CLAW} to guarantee stability for the solution in the given time domain.

Figure 13 compares the numerical solutions at time $t = 0.5$ and $t = 1$, both of which extend past the training time. While all methods capture the main features of the solution at $t = 0.5$, it is evident that the CFN yields the most accurate results. The errors in both the nCFN and nCFN-reg solutions are significantly larger when



(a) $h, t = 0.5$ (b) $h, t = 1$ (c) $hv, t = 0.5$ (d) $hv, t = 1$

Fig. 13: Comparison of the references solution to Example 5.3 and the trained DNN model predictions at different times for dense and noise-free observations.



(a) $C(\mathbf{h})$ (b) $C(\mathbf{h} \odot \mathbf{v})$

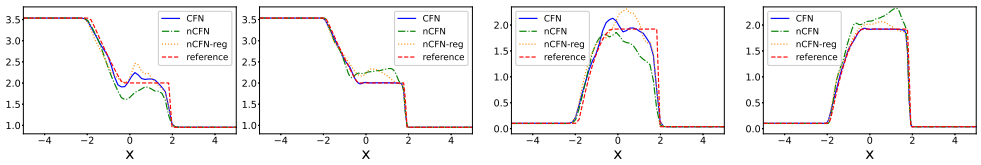
Fig. 14: Discrete conserved quantity remainder given by (4.6) of each method for Example 5.3. (a) $C(\mathbf{h})$ and (b) $C(\mathbf{h} \odot \mathbf{v})$. Graphs are in semi-log scale for visibility.

$t = 1$, and the rarefaction wave structure is not discernible in the nCFN case. We determine the conservation of each method by calculating (4.6) for $\mathbf{h}$ and $\mathbf{h} \odot \mathbf{v}$, where $\odot$ denotes elementwise multiplication, and show the results in Figure 14. As in the case for Burgers equation, only the CFN method is conservative.

As we did for Example 5.2 Case I, we again conducted experiments to test the robustness of the CFN method with respect to different network parameters. We obtained similar results as displayed in Figure 10, demonstrating both the robustness of our method as well as better performance when compared to both the nCFN and nCFN-reg.

Case II: Sparse and noise-free observations. To simulate this environment we use CLAWPACK to solve Example 5.3 on coarser grids, respectively $N = 64$ and $N = 128$, to obtain the training data collected at $t_l = l\Delta t$, $l = 1, \ldots, L$, where $L = 20$. For consistency we again choose $\Delta t = 0.005$ so that the training trajectory final time is $t = 0.1$.

Figure 15 compares the solutions using CFN, nCFN and nCFN-reg with the reference solution (again defined as the CLAWPACK solution with $N = 1024$) at time $t = 1$. For $N = 64$ it is apparent that none of the methods are able to accurately learn the system dynamics, and large fluctuations are particularly noticeable in the region between the rarefaction and shock wave. For $N = 128$ we observe improvement for all models. The CFN clearly yields the most accurate results, and is the only method able to capture the structure of the rarefaction wave. This is not surprising since we already observed in Figure 13 that $N = 512$ did not provide enough resolution, even for the nCFN-reg case. Thus we see the importance of training the network using the flux form.



(a) $h, N = 64$ (b) $h, N = 128$ (c) $hv, N = 64$ (d) $hv, N = 128$

Fig. 15: Comparison of the reference solutions to Example 5.3 and the trained DNN model predictions using noise free observations at time $t = 1$ for $N = 64$ and 128.

Case III: Dense and noisy observations. In this case the training data are given by

$$\begin{aligned} \tilde{\mathbf{h}}^{(k)}(t_l) &= \mathbf{h}^{(k)}(t_l) + \boldsymbol{\epsilon}_l^{(k)}, \\ \widetilde{(\mathbf{h}^{(k)} \odot \mathbf{v}^{(k)})}(t_l) &= (\mathbf{h}^{(k)} \odot \mathbf{v}^{(k)})(t_l) + \boldsymbol{\eta}_l^{(k)}, \end{aligned} \tag{5.10}$$

for $l = 1, \ldots, L$ and $k = 1, \ldots, N_{traj}$. The added noise $\boldsymbol{\epsilon}_l^{(k)}$ and $\boldsymbol{\eta}_l^{(k)}$ are i.i.d. Gaussian with zero mean and variance determined using various noise values based on the mean of $\mathbf{u}$ (5.5). We again consider noise levels corresponding to 100%, 50%, 20%, and 10%.

The solution profiles for height and momentum in Example 5.3 obtained using the different network constructions are shown in Figure 16. It is apparent that all three methods yield significant diffusion in high noise environments. It is noteworthy that when the amount of noise is at 20%, both the nCFN and nCFN-reg methods produce solutions that seem to increase (rather than diffuse) energy, suggesting that these methods are learning noise-related dynamics. In this regard, the CFN method appears to be the most robust, meaning that along with the overall improved quality of the solution with decreasing amounts of noise, the solution itself behaves consistently as a function of the noise level, with less diffusion apparent as the amount of noise decreases.

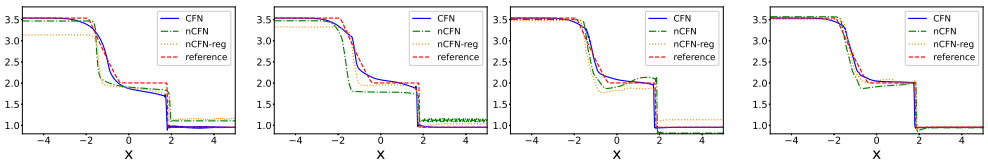
We again omit figures comparing the discrete conserved quantity remainder, (4.6), of each method for Cases II and III since the methods all generate the same general behavior pattern as what is shown for Case I in Figure 14.

5.3. Euler equation. As a final example we consider the Euler equations for gas dynamics, specifically the Shu-Osher problem [41]. The problem is challenging since the resulting shock wave impacts a sinusoidally-varying density field yielding more complex structures than apparent in Examples 5.1 and 5.3.

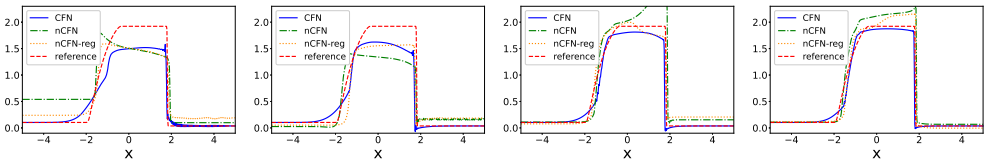
EXAMPLE 5.4. *Consider the system of equations for $t > 0$ given by*

$$\begin{aligned} \rho_t + (\rho u)_x &= 0, \\ (\rho u)_t + (\rho u^2 + p)_x &= 0, \\ E_t + (u(E + p))_x &= 0, \end{aligned} \tag{5.11}$$

in the domain $(-5, 5)$. We assume no flux boundary conditions and initial conditions



(a) 100% noise (b) 50% noise (c) 20% noise (d) 10% noise



(e) 100% noise (f) 50% noise (g) 20% noise (h) 10% noise

Fig. 16: Comparison of the reference solution for height h (top) and momentum hv (bottom) in Example 5.3 and the trained DNN model predictions at $t = 1$ for dense ($N = 512$) and noisy observations.

given by[6]

$$\rho(x,0) = \begin{cases} \rho_l, & \text{if } x \le x_0, \\ 1 + \varepsilon \sin(5x), & \text{if } x_0 < x \le x_1, \\ 1 + \varepsilon \sin(5x) e^{-(x-x_1)^4} & \text{otherwise}, \end{cases} \quad u(x,0) = \begin{cases} u_l, & \text{if } x \le x_0, \\ 0, & \text{otherwise}, \end{cases}$$

$$p(x,0) = \begin{cases} p_l, & \text{if } x \le x_0, \\ p_r, & \text{otherwise}, \end{cases} \qquad E(x,0) = \frac{p_0}{\gamma - 1} + \frac{1}{2}\rho(x,0)u(x,0)^2.$$

The parameters are given by

$$\begin{aligned} \rho_l &\sim U[\hat{\rho}_l(1-\epsilon), \quad \hat{\rho}_l(1+\epsilon)], \quad \hat{\rho}_l = 3.857135, \\ \varepsilon &\sim U[\hat{\varepsilon}(1-\epsilon), \quad \hat{\varepsilon}(1+\epsilon)], \quad \hat{\varepsilon} = 0.2, \\ p_l &\sim U[\hat{p}_l(1-\epsilon), \quad \hat{p}_l(1+\epsilon)], \quad \hat{p}_l = 10.33333, \\ p_r &\sim U[\hat{p}_r(1-\epsilon), \quad \hat{p}_r(1+\epsilon)], \quad \hat{p}_r = 1, \\ u_l &\sim U[\hat{u}_l(1-\epsilon), \quad \hat{u}_l(1+\epsilon)], \quad \hat{u}_l = 2.62936, \\ x_0 &\sim U[\hat{x}_0(1-\epsilon), \quad \hat{x}_0(1+\epsilon)], \quad \hat{x}_0 = -4, \end{aligned} \tag{5.12}$$

with $\epsilon = .1$, $x_1 = 3.29867$ *and* $\gamma = 1.4$. *We note that* $\hat{\rho}$, $\hat{p}_l$, $\hat{u}$ *are the same values as those used in the CLAWPACK Shu-Osher example.*

The $k = 1, \ldots, N_{traj}$ training sets are generated by solving (5.4) using CLAW-

[6]The usual Shu-Osher problem does not use $\rho(x,0) = 1 + \varepsilon \sin(5x) e^{-(x-x_1)^4}$ for x in the right part of the domain. We include this term to "flatten" the solution at the boundary so that we can apply (3.9) without introducing an artificial boundary layer.

PACK (HLLE Riemann Solver) for initial conditions given by

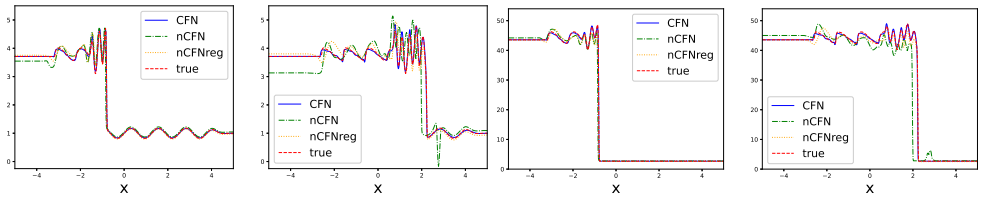
$$\rho^{(k)}(x,0) = \begin{cases} \rho_l^{(k)}, & \text{if } x \le x_0^{(k)}, \\ 1+\varepsilon^{(k)}\sin(5x), & \text{if } x_0^{(k)} < x \le x_1, \\ 1+\varepsilon^{(k)}\sin(5x)e^{-(x-x_1)^4}, & \text{otherwise}, \end{cases}$$

$$u^{(k)}(x,0) = \begin{cases} u_l^{(k)}, & \text{if } x \le x_0^{(k)}, \\ 0, & \text{otherwise}, \end{cases} \quad p^{(k)}(x,0) = \begin{cases} p_l^{(k)}, & \text{if } x \le x_0^{(k)}, \\ p_r^{(k)}, & \text{otherwise}, \end{cases}$$

$$E^{(k)}(x,0) = \frac{p_0^{(k)}}{\gamma-1} + \frac{1}{2}\rho^{(k)}(x,0)u^{(k)}(x,0)^2.$$

The corresponding parameters are given in (5.12) (written without the superscript k) and the boundary conditions are imposed using (3.9) in all experiments. The reference solution is obtained using CLAWPACK with $N = 1024$ so that $\Delta x = \frac{10}{1024}$.

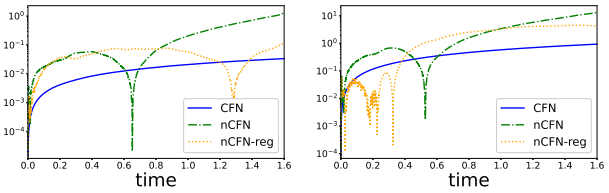
To train over a longer period of time without increasing the computational cost we once again employ the same sub-sampling technique used in Example 5.1 to generate training data from observed snapshots of the solution on an extended domain. As before we set $M = 300$ as the extended length of each trajectory. The snapshots of the solution, (2.4), are obtained via CLAWPACK for each of the $N_{traj} = 300$ trajectories at times $t = m\Delta t$, $m = 1, \ldots, M$, where $\Delta t = 0.002$ (chosen to satisfy the CFL condition). Each sub-sampled trajectory of length L is then built consecutively from the snapshot solutions. That is, each trajectory is comprised of the solutions in (2.4) at sequential times $t_0^{(k)} + l\Delta t$, $l = 1, \ldots, L$.



(a) $\rho, t = 0.8$ (b) $\rho, t = 1.6$ (c) $E, t = 0.8$ (d) $E, t = 1.6$

Fig. 17: Comparison of the reference density ρ and energy E solutions to Example 5.4 and the trained DNN model predictions at different times for dense ($N = 512$) and noise-free observations.

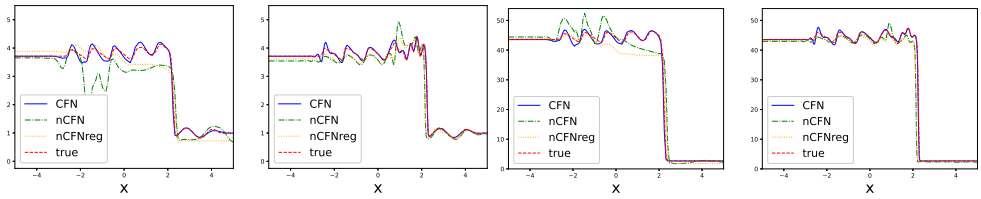
Case I: Dense and noise-free observations. We first consider an idealized environment for which the observations are dense and noise-free. Specifically we choose $N = 512$, yielding $\Delta x = \frac{10}{512}$, so that our solution is well-resolved. Figure 17 shows the solutions ρ and E at times $t = 0.8$ and $t = 1.6$, both of which are outside of training time domain $[0, 0.6]$. Observe that as the shock wave interacts with the density field, the solution exhibits oscillations to the left side of the shock front. It is evident that only the CFN network produces a solution that captures the oscillatory features of the solution. By contrast, the nCFN solution exhibits significant errors with non-physical oscillations to the right of the shock. It moreover produces the wrong shock front location at $t = 1.6$. The results for nCFN-reg are somewhat improved, but still do not accurately capture the dynamics of the system.



(a) $C(\boldsymbol{\rho})$ (b) $C(\mathbf{E})$

Fig. 18: Discrete conserved quantity remainder given in (4.6) of each method for Example 5.4. (a) $C(\boldsymbol{\rho})$ and (b) $C(\mathbf{E})$. Graphs are in semi-log scale for visibility.

Figure 18 confirms our observations in Figure 17. Specifically, we see that none of the methods are conservative, with the error increasing more rapidly in the nCFN and the nCFN-reg cases. The error corresponding to the CFN appears to grow linearly with time, suggesting long term numerical stability when considering classical numerical conservation laws analysis.



(a) $\rho, N = 128$ (b) $\rho, N = 256$ (c) $E, N = 128$ (d) $E, N = 256$

Fig. 19: Comparison of the reference density ρ and energy E solutions to Example 5.4 and the trained DNN model predictions at $t = 1.6$ for sparse and noise-free observations.

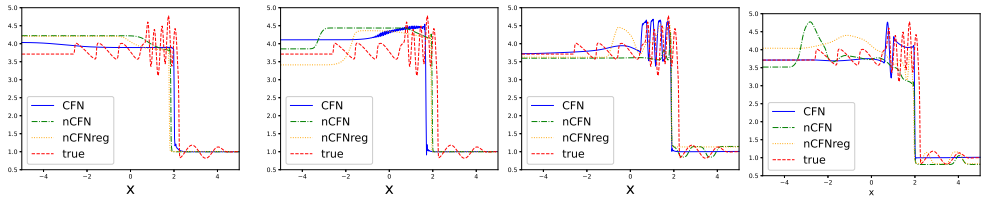
Case II: Sparse and noise-free observations. We now consider more sparsely observed data by choosing $N = 128$ and $N = 256$ shown in Figure 19. Given the results in the idealized environment, it is not surprising that neither the nCFN or nCFN-reg is able to capture the dynamics of Example 5.4 in the sparse observation case. While some solution details are lost, and there is noticeable error in the location of the shock front, it is evident that the CFN network still provides qualitative structure commensurate with the given resolution.

Case III: Dense and noisy observations. In this case the training data are

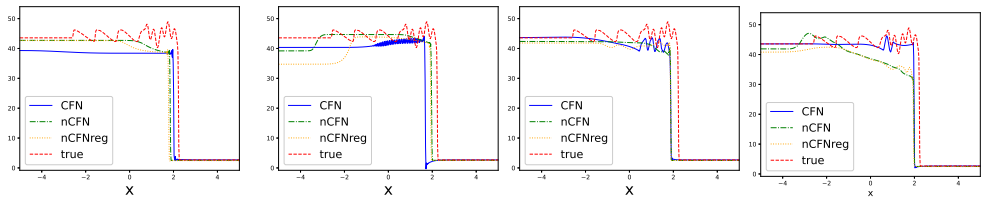
$$\begin{aligned} \tilde{\boldsymbol{\rho}}^{(k)}(t_l) &= \boldsymbol{\rho}^{(k)}(t_l) + \boldsymbol{\epsilon}_l^{(k)}, \\ \widetilde{(\boldsymbol{\rho}^{(k)} \odot \mathbf{u}^{(k)})}(t_l) &= (\boldsymbol{\rho}^{(k)} \odot \mathbf{u}^{(k)})(t_l) + \boldsymbol{\eta}_l^{(k)}, \\ \tilde{\mathbf{E}}^{(k)}(t_l) &= \mathbf{E}^{(k)}(t_l) + \boldsymbol{\delta}_l^{(k)}, \end{aligned} \tag{5.13}$$

for $l = 1, \ldots, L$ and $k = 1, \ldots, N_{traj}$. The added noise $\boldsymbol{\epsilon}_l^{(k)}$, $\boldsymbol{\eta}_l^{(k)}$, and $\boldsymbol{\delta}_l^{(k)}$ are i.i.d. Gaussian with zero mean and variance determined by various noise levels (5.5). We

again consider the same noise levels, 100%, 50%, 20%, and 10%.



(a) ρ, 100% noise (b) ρ, 50% noise (c) ρ, 20% noise (d) ρ, 10% noise



(e) E, 100% noise (f) E, 50% noise (g) E, 20% noise (h) E, 10% noise

Fig. 20: Comparison of the reference solution of density ρ (top) and Energy E (bottom) in Example 5.4 to the trained DNN model predictions at time $t = 1.6$ for dense ($N = 512$) and noisy observations.

The solutions for density ρ and energy E are presented in Figure 20. We observe similar behavior as was seen for Case III in Example 5.3. Specifically, all three methods yield significant diffusion in high noise environments, 100% and 50%, and cannot predict the oscillatory structure to the left of the shock front. Unlike what was observed in Example 5.3, neither the nCFN nor the nCFN-reg appear to learn the noise-related dynamics, as even in the 10% noise level case the solutions still appear diffusive. This is likely because loss function still promotes a diffuse solution as opposed to one that contains noise-related dynamics. As the noise decreases, the CFN appears to capture some of the oscillatory details in the solution. In this regard, we again see that the CFN is a more robust network with respect to noise.

We again omit figures comparing the discrete conserved quantity remainder, (4.6), of each method for Cases II and III since the methods all generate the same general behavior pattern as what is shown for Case I in Figure 18.

6. Conclusion. In this investigation we proposed a conservative form network (CFN) to learn the dynamics of unknown hyperbolic systems of conservation laws from observation data. Inspired by classical finite volume methods for hyperbolic conservation laws, our new method employs a neural network to learn the flux function of the unknown system. The predictions using CFN yield the appropriate conserved quantities and also recover the correct physical structures, including the shock speed, even outside the training domain. We validated the effectiveness and robustness of our CFN approach through a series of numerical experiments for three classic examples of one-dimensional conservation laws. Even in non-ideal environments, our results consistently demonstrate that the CFN outperforms the traditional non-conservative form network (nCFN) and its regularized version (nCFN-reg) in terms of accuracy,

efficiency, and robustness, in particular since it does not require fine-tuning of regularization parameters.

The current study does not attempt to optimize model performance for the realistic data cases, and we will attempt to do this in future investigations. For the sparse observation environment, the Mori–Zwanzig formalism [31, 48], for which memory is included in the network, may potentially enhance the overall performance. In the noisy data environment one might consider using a denoising technique such as regularization [6, 13]. Future investigations will also consider two-dimensional examples with more complex boundary conditions. Finally, we will also study mixed-form systems, where the CFN may be used for equations representing conserved quantities within the system.

REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. Software available from tensorflow.org.

[2] P. J. Baddoo, B. Herrmann, B. J. McKeon, J. N. Kutz, and S. L. Brunton, *Physics-informed dynamic mode decomposition piDMD*, arXiv preprint arXiv:2112.04307, (2021).

[3] A. D. Beck, J. Zeifang, A. Schwarz, and D. G. Flad, *A neural network based shock detection and localization approach for discontinuous galerkin methods*, Journal of Computational Physics, 423 (2020), p. 109824.

[4] T. Bertalan, F. Dietrich, I. Mezić, and I. G. Kevrekidis, *On learning hamiltonian systems from data*, Chaos: An Interdisciplinary Journal of Nonlinear Science, 29 (2019).

[5] D. A. Bezgin, S. J. Schmidt, and N. A. Adams, *A data-driven physics-informed finite-volume scheme for nonclassical undercompressive shocks*, Journal of Computational Physics, 437 (2021), p. 110324.

[6] C. M. Bishop, *Training with noise is equivalent to tikhonov regularization*, Neural Comput., 7 (1995), pp. 108–116.

[7] S. L. Brunton, J. L. Proctor, and J. N. Kutz, *Discovering governing equations from data by sparse identification of nonlinear dynamical systems*, Proc. Natl. Acad. Sci., 113 (2016), pp. 3932–3937.

[8] Z. Chen, V. Churchill, K. Wu, and D. Xiu, *Deep neural network modeling of unknown partial differential equations in nodal space*, J. Comput. Phys., 449 (2022), p. 110782.

[9] Z. Chen and D. Xiu, *On generalized residual network for deep learning of unknown dynamical systems*, Journal of Computational Physics, 438 (2021), p. 110362.

[10] Clawpack Development Team, *Clawpack software*, 2020, http://www.clawpack.org. Version 5.7.1.

[11] M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel, and S. Ho, *Lagrangian neural networks*, in ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations, 2020.

[12] X. Fu, W. Mao, L.-B. Chang, and D. Xiu, *Modeling unknown dynamical systems with hidden parameters*, J. Mach. Learn. Model. Comput., 3 (2022), pp. 79–95.

[13] G. H. Golub, P. C. Hansen, and D. P. O'Leary, *Tikhonov regularization and total least squares*, SIAM J. Matrix Anal. Appl., 21 (1999), pp. 185–194.

[14] S. Gottlieb and C.-W. Shu, *Total variation diminishing Runge-Kutta schemes*, Math. Comput., 67 (1998), pp. 73–85.

[15] S. Greydanus, M. Dzamba, and J. Yosinski, *Hamiltonian neural networks*, in Adv. Neural Inf. Process Syst., vol. 32, 2019.

[16] P. C. Hansen, *Discrete Inverse Problems: Insight and Algorithms*, SIAM, 2010.

[17] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit., 2016, pp. 770–778.

[18] J. S. Hesthaven, *Numerical Methods for Conservation Laws: From Analysis to Algorithms*, SIAM, Philadelphia, PA, 2018.

[19] A. D. Jagtap and G. E. Karniadakis, *Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations*, in Proceedings of the AAAI 2021 Spring Symposium on Combining Artificial Intelligence and Machine Learning with Physical Sciences, Stanford, CA, USA, March 22nd - to - 24th, 2021, J. Lee, E. F. Darve, P. K. Kitanidis, M. W. Mahoney, A. Karpatne, M. W. Farthing, and T. J. Hesser, eds., vol. 2964 of CEUR Workshop Proceedings, CEUR-WS.org, 2021.

[20] A. D. Jagtap, E. Kharazmi, and G. E. Karniadakis, *Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems*, Computer Methods in Applied Mechanics and Engineering, 365 (2020), p. 113028.

[21] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, *Physics-informed machine learning*, Nature Reviews Physics, 3 (2021), pp. 422–440.

[22] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, in Proc. Int. Conf. Learn. Representations, 2015.

[23] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, Nature, 521 (2015), pp. 436–444.

[24] R. J. LeVeque, *Numerical Methods for Conservation Laws*, vol. 214, Springer, 1992.

[25] R. J. LeVeque, *Finite Volume Methods for Hyperbolic Problems*, vol. 31, Cambridge University Press, 2002.

[26] Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. M. Stuart, and A. Anandkumar, *Fourier neural operator for parametric partial differential equations*, in 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021, OpenReview.net, 2021.

[27] Z. Li, M. Liu-Schiaffini, N. B. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. M. Stuart, and A. Anandkumar, *Learning chaotic dynamics in dissipative systems*, in NeurIPS, 2022.

[28] Z. Liu, V. Madhavan, and M. Tegmark, *Machine learning conservation laws from differential equations*, Physical Review E, 106 (2022), p. 045307, https://doi.org/10.1103/PhysRevE.106.045307.

[29] Z. Long, Y. Lu, X. Ma, and B. Dong, *Pde-net: Learning pdes from data*, in International conference on machine learning, PMLR, 2018, pp. 3208–3216.

[30] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, *Learning nonlinear operators via deeponet based on the universal approximation theorem of operators*, Nat. Mach. Intell., 3 (2021), pp. 218–229.

[31] H. Mori, *Transport, collective motion, and Brownian motion*, Prog. Theor. Phys., 33 (1965), pp. 423–455.

[32] T. Qin, Z. Chen, J. D. Jakeman, and D. Xiu, *Data-driven learning of nonautonomous systems*, SIAM J. Sci. Comput., 43 (2021), pp. A1607–A1624.

[33] T. Qin, Z. Chen, J. D. Jakeman, and D. Xiu, *Deep learning of parameterized equations with applications to uncertainty quantification*, Int. J. Uncertain. Quantif., 11 (2021), pp. 63–82.

[34] T. Qin, K. Wu, and D. Xiu, *Data driven governing equations approximation using deep neural networks*, J. Comput. Phys., 395 (2019), pp. 620–635.

[35] M. Raissi, *Deep hidden physics models: Deep learning of nonlinear partial differential equations*, The Journal of Machine Learning Research, 19 (2018), pp. 932–955.

[36] 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.

[37] R. Rodriguez-Torrado, P. Ruiz, L. Cueto-Felgueroso, M. C. Green, T. Friesen, S. Matringe, and J. Togelius, *Physics-informed attention-based neural network for hyperbolic partial differential equations: application to the buckley–leverett problem*, Scientific reports, 12 (2022), p. 7557.

[38] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, *Data-driven discovery of partial differential equations*, Sci. Adv., 3 (2017), p. e1602614.

[39] H. Schaeffer, *Learning partial differential equations via data discovery and sparse optimization*, Proc. R. Soc. A., 473 (2017), p. 20160446.

[40] H. Schaeffer, G. Tran, and R. Ward, *Extracting sparse high-dimensional dynamics from limited data*, SIAM Journal on Applied Mathematics, 78 (2018), pp. 3279–3295.

[41] C.-W. Shu and S. Osher, *Efficient implementation of essentially non-oscillatory shock-capturing schemes*, J. Comput. Phys., 77 (1988), pp. 439–471.

[42] B. Stevens and T. Colonius, *Enhancement of shock-capturing methods via machine learning*, Theoretical and Computational Fluid Dynamics, 34 (2020), pp. 483–496.

[43] J. H. Tu, *Dynamic mode decomposition: Theory and applications*, PhD thesis, Princeton

University, 2013.
[44] K. Wu and D. Xiu, *Data-driven deep learning of partial differential equations in modal space*, J. Comput. Phys., 408 (2020), p. 109307.
[45] S. Zhang and G. Lin, *Robust data-driven discovery of governing physical laws with error bars*, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 474 (2018), p. 20180305.
[46] Z. Zhang, Y. Shin, and G. Em Karniadakis, *GFINNs: GENERIC formalism informed neural networks for deterministic and stochastic dynamical systems*, Phil. Trans. R. Soc. A., 380 (2022), p. 20210207.
[47] Y. D. Zhong, B. Dey, and A. Chakraborty, *Symplectic ode-net: Learning hamiltonian dynamics with control*, in 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020, OpenReview.net, 2020.
[48] R. Zwanzig, *Nonlinear generalized langevin equations*, J. Stat. Phys., 9 (1973), pp. 215–220.