Identification of Nonlinear Conservation Laws Using Symbolic Neural Networks

by

Qing Li

Thesis submitted in fulfillment of
the requirements for the degree of

PHILOSOPHIAE DOCTOR
(PhD)

Faculty of Science and Technology
Department of Energy and Petroleum Engineering
2023

University of Stavanger
N-4036 Stavanger
NORWAY
www.uis.no

ISBN 978-82-8439-199-1
ISSN 1890-1387

PhD Thesis UiS no. 731

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of Philosophiae Doctor at the University of Stavanger, Norway. The doctoral research was funded by the Norwegian Ministry of Education and Research. The research lasted from January 2021 until fall 2023 and was conducted at the Department of Energy and Petroleum Engineering at the University of Stavanger, excluding a visit to Shanghai Jiaotong University from May 2023 until July 2023. My main supervisor was Professor Steinar Evje.

The outcome of this research has been given through four published papers and two manuscripts submitted during spring 2023.

Qing Li, November 2023

Abstract

Nonlinear dynamical systems are omnipresent in nature, commonly seen in many disciplines such as physics, biology, chemistry, climate science, and engineering. In this thesis, we introduce several new ideas by integrating machine learning and numerical methods, effectively tackling challenging forward and inverse problems of physical systems. According to [1], research approaches in this field can be broadly categorized based on the synergy between deep learning and domain knowledge into three groups: Supervised Methods, Physics-informed Methods, and Interleaved Methods. Supervised Methods are the classic learning approaches where a physical system produces the data, but no further interaction exists between this physical system and deep learning. In Physics-informed Methods, the physical dynamics are encoded in the loss function, typically in the form of differentiable operations. Interleaved Methods tightly integrate the physical system with the learning process, merging full simulations with deep neural network outputs. Whether addressing Ordinary Differential Equations (ODEs) or Partial Differential Equations (PDEs) in this thesis, the core of our approaches intricately unites Symbolic Neural Networks with ODE & PDE solvers, categorizing our techniques as Interleaved Methods.

We start with a slightly simpler task of trying to identify unknown ODEs with parameters from trajectory data in Paper I. Instead of directly learning ODEs using an Ordinary Neural Network (O-Net) [2], we present a novel strategy by combining Symbolic Neural Network (S-Net)—endowed with the capacity to grasp analytical expressions—with an ODE Solver to predict the dynamical system. Our numerical experiments demonstrate that our approach outperforms O-Net when applied to the Lotka-Volterra and Lorenz equations. This discovery concerning the realm of ODEs also imparts valuable insights for our future endeavors in tackling PDEs-related challenges.

Graph Neural Networks (GNNs) belonging to Supervised Methods have gained significant attention in recent years due to their ability to process and analyze data structured as graphs. By discretizing continuous spatial domains into grids or meshes, individual grid points or mesh elements can be regarded as nodes within a graph. The connections between nodes can represent the spatial relationships between these points or elements. GNNs have been widely used to solve spatially-dependent PDEs with smooth solutions. In paper II, we explore the application of GNNs to solve conservation laws with non-smooth solutions. Experimental results show that the model can predict accurately when parameters are within a specific range. However, when parameters deviate too much from that used for the training model, the model's predictive power is significantly reduced.

The achievements of the S-Net and ODE Solver in tackling ODEs, coupled with the limitations of GNNs in extending to conservation law challenges, reinforce the need to first learn the expressions of the unknown flux functions of the involved conservation law. From

this foundation, we can proceed to forecast subsequent states of the nonlinear dynamical system with greater assurance. In Paper III, we introduce ConsLaw-Net, a combination of S-Net and an entropy-satisfying discretization scheme. This work addresses the problems of one-dimensional conservation law without parameters, and empirical outcomes robustly affirm the effectiveness of our approach. However, the case with conservation laws that involve a parameter, requires that the role of the parameter must also be learned. We propose an appropriate extending to deal robustly with this situation. A two-step learning method, i.e., combining Conslaw-Net and Linear Regression Neural Network (LRNN), is proposed in Paper IV. We test it on two different systems and achieve good results. Furthermore, in paper V, we train the enhanced ConsLaw-Net through a combination of joint and alternating equation strategies, effectively addressing intricate two-dimensional conservation law scenarios demanding high precision despite limited informative data. Conclusively, in Paper VI, we unveil an upgraded ConsLaw-Net tailored for deducing the functional expressions of both flux and diffusion functions within the setting of degenerate convection-diffusion models, accommodating diverse observation modalities.

Taken together, the methodologies outlined in this thesis provide new and hopefully useful tools in the search for hidden nonlinear conservation laws behind a given set of observation data. If we should formulate the main findings of this thesis in a few sentences, it might be as follows: To uncover a possible unknown nonlinear scalar conservation law from synthetic observation data seems attainable by combining appropriate regularity imposed on the unknown function(s), as expressed by the Symbolic Neural Networks, with a "suitable" set of observation data. The meaning of "suitable" here is that small amounts of data might not be sufficient to identify the unknown flux function(s). However, by adding more observation data the proposed method is more and more likely to find the ground truth flux function.

Acknowledgements

I am profoundly grateful to have reached this milestone in my academic journey, and I owe my success to the unwavering support and guidance of numerous individuals and institutions.

First and foremost, I am greatly grateful to my supervisor, Professor Steinar Evje, for his constant support and encouragement during the last three years. Thank you for spending so much time patiently explaining numerical methods I was unfamiliar with and answering every question I had, even if they were naive. Every piece of advice you've given me has been carefully thought out by you and is very specific and doable. In my first year as a Ph.D. candidate, I tried many methods, but none of them worked. You still respected every idea of mine and gave me enough freedom to explore further. You are always enthusiastic about learning new methods, and this has influenced me so that I hope I can carry with me this attitude in the future.

I would like to thank Professor Zhi-Qin John Xu and his team at Shanghai Jiaotong University for affording me the privilege of visiting. This experience not only granted me insights into cutting-edge research on large language models but also significantly broadened the horizons of my own scholarly pursuits.

I'm deeply grateful to my roommates—Reyhaneh Banihabib, Benjamin Baroulliet, and Jie Cao—for cultivating a productive work environment. My appreciation also goes to the staff of the Faculty of Science and Technology and the Department of Energy and Petroleum, especially Øystein Arild and Norbert Puttkamer, whose pivotal role as an unsung hero in every doctoral student's journey cannot be overstated.

In addition, thanks to my friends, Prachi Vinod Wadatkar, Christin Kreutz, Arian Baloochestani Asl, Ahmad Mohammad Ahmad Faza, Saul Fuster Navarro, Luca Tomasetti, Neel Kanwal, Yao Zhang, and many others. I vividly recall my initial days in Norway when you hosted a heartwarming Chinese New Year gathering to ease my homesickness. Engaging in hiking, workouts, meaningful information exchange, and joint birthday celebrations, I believe these enduring memories will continue to foster a sense of warmth within each of us.

Undoubtedly, I'd like to express my deepest gratitude to my family. To my parents, I am indebted for instilling kindness, courage, and diligence within me. These fundamental virtues continue to guide me today and beyond. To my husband, Jiahui Geng, your kind and tender character, dedication to friendship, and enthusiasm for work infuse our lives with warmth, love, and fascination. The prospect of embarking on life's journey and exploring the world together with you fills me with genuine excitement.

Finally, I want to say thank you to myself. Thank you for following your heart and being courageous in the face of uncertainty. During my Ph.D., I did everything that I could do in every year, every month, every day, every minute, every second. So I have no regrets. No matter when you start, it is important that you do not stop after starting. No matter when you end, it is more important that you do not regret it after ended.

Qing Li, November 2023

List of publications

The main part of this dissertation is made up of the following published scientific papers:

- **Paper I**

  Learning Parameterized ODEs from Data
  Qing Li, Steinar Evje, Jiahui Geng
  IEEE Access 11 (2023): 54897 - 54909
  doi = https://ieeexplore.ieee.org/document/10143183

- **Paper II**

  Solving Nonlinear Conservation Laws of Partial Differential Equations Using Graph Neural Networks
  Qing Li, Jiahui Geng, Steinar Evje, Chunming Rong
  Proceedings of the Northern Lights Deep Learning Workshop 4 (2023)
  doi = https://doi.org/10.7557/18.6808

- **Paper III**

  Learning the Nonlinear Flux Function of a Hidden Scalar Conservation Law from Data
  Qing Li, Steinar Evje
  Networks and Heterogeneous Media 18.1 (2023): 48-79
  doi = http://www.aimspress.com/article/doi/10.3934/nhm.2023003

- **Paper IV**

  Identification of the Flux Function of Nonlinear Conservation Laws with Variable Parameters
  Qing Li, Jiahui Geng, Steinar Evje
  Physica D: Nonlinear Phenomena 451 (2023): 133773
  doi = https://doi.org/10.1016/j.physd.2023.133773

- **Paper V**

  An Alternating Flux Learning Method for Multidimensional Nonlinear Conservation Laws
  Qing Li, Steinar Evje
  Under Review in SIAM Journal on Scientific Computing

- **Paper VI**

  Learning the Flux and Diffusion Function for Degenerate Convection-Diffusion Equations Using Different Types of Observations
  Qing Li, Steinar Evje
  Under Review in BIT Numerical Mathematics

Contents

Chapter 1

Introduction

1.1 Research Background and Challenges

We model physical phenomena in science and engineering using ODEs or PDEs. These equations are derived conventionally from rigorous first principles under ideal conditions like conservation laws or knowledge-based phenomenological derivations. Our primary objective is to extract explicit expressions for these equations, including specific components like flux or diffusion functions within dynamics, from observational data. Subsequently, we employ the learned functions to predict dynamic states at a later time. One might question why we don't simply forecast the solutions of equations directly without mastering the explicit forms. Our experiments elucidated in [3] reveal that unearthing the most fitting differential equations to depict these dynamics through sensed or measured data offers greater value to scientists seeking to comprehend the intricacies of mathematical models or dynamical systems. This approach not only enhances our understanding of behavioral patterns but also leads to more precise predictions.

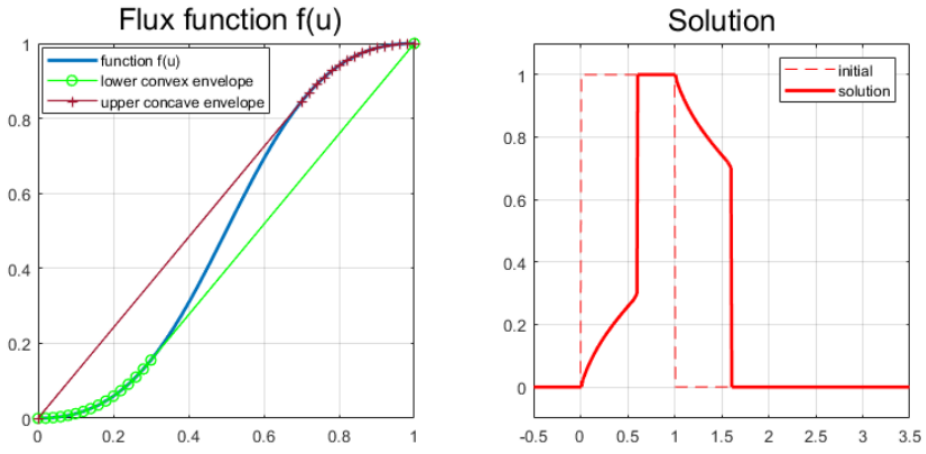


Figure 1.1: Left: Example of nonlinear flux function $f(u) = \frac{u^2}{u^2+(1-u)^2}$ (blue curve). Upper concave envelope (brown curve) and lower convex envelope (green curve) are also included. Right: The solution of (1.1) at time $T = 0.5$ is shown (red solid curve) together with its initial data $u_0(x)$ (red dashed line).

However, obtaining the analytical form of these equations proves to be a formidable task due to the inherent challenges. The scarcity and noise often present in practical observations compound this complexity. Consider the example of a general scalar nonlinear

conservation law. Here, we focus on the one-dimensional scenario given by:

$$u_t + f(u)_x = 0 \tag{1.1}$$

where $u = u(x,t)$ is the main variable and $f(u)$ is the unknown flux function. It is well known that (1.1) will generate shock wave solutions $u(x,t)$ in finite time despite the fact that initial data $u_0(x)$ is smooth [4, 5]. As jumps arise and disappear in the solution over the time period, the data may lack information about the flux function $f(u)$ that we want to identify. An illustration of this situation is given in Fig. 1.1. At time $t = 0$, the initial data $u_0(x)$ involves one jump at $x = 0$ and another jump at $x = 1$. The initial jump at $x = 0$ is instantly transformed into a solution that is a combination of a continuous wave solution and a discontinuous wave $(u_L, u_R) \approx (0.3, 1.0)$, as dictated by the lower convex envelope shown in the left panel (green curve). Similarly, the initial jump at $x = 1$ is transformed into a solution that is a combination of a continuous wave solution and a discontinuous wave $(u_L, u_R) \approx (0.7, 0)$, in accordance with the upper concave envelope illustrated in the left panel (brown curve). In addition, the concise and complete mathematical expressions describing numerous intricate real-world systems continue to be elusive or only partially understood.

Over the past decade, significant advancements in machine learning, data science, and computational power have ushered in an innovative approach to unveiling underlying equations from data: the data-driven discovery of governing equations for nonlinear dynamics. In their study [1], Thuerey and colleagues meticulously categorized these techniques into three clear groups: Supervised Methods, Physics-informed Methods, and Interleaved Methods. These classifications are dictated by how deep learning techniques intertwine with domain expertise, often taking shape as model equations expressed through ODEs or PDEs. Nevertheless, most of the work has been on equations whose solutions do not contain discontinuities. In this thesis, after a first brief visit exploring how to identify unknown functions involved in ODEs, we focus on the problems of conservation law, which is complicated by the lack of data due to the presence of discontinuous solutions.

1.2 The Parameterized ODEs

Parameterized ODEs can be expressed by

$$\frac{d\boldsymbol{u}(t;\boldsymbol{\mu})}{dt} = \boldsymbol{f}(\boldsymbol{u}(t;\boldsymbol{\mu}), t; \boldsymbol{\mu}), \tag{1.2}$$

with a parameterized initial condition $\boldsymbol{u}_0(\boldsymbol{\mu})$, where $\boldsymbol{u}(t;\boldsymbol{\mu})$ is a time-continuous representation of a hidden state, $\boldsymbol{\mu} = [\boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_{n_{\boldsymbol{\mu}}}] \in \mathbb{R}^{n_\mu}$ denotes problem-specific input parameters, $\boldsymbol{f}$ is a parameterized velocity function that defines the dynamics of hidden states over time. Inspired by the concept of parameterized ODEs, this ODE system has multiple latent trajectories that depend on the input parameters.

1.3 The Conservation Law

Nonlinear conservation laws find extensive application across fluid mechanics, biology, physics, and chemical engineering. These laws come in various forms based on factors such as dimensional space, the presence of parameters, and the inclusion of diffusive flux.

1.3.1 The One-Dimensional Scalar Conservation Law

The expression of the one-dimensional scalar conservation law is as follows

$$\begin{aligned} u_t + f(u)_x &= 0, \quad x \in (0, L) \\ u_x|_{x=0} &= u_x|_{x=L} = 0, \\ u|_{t=0} &= u_0(x), \end{aligned} \tag{1.3}$$

where u as the main variable, $f(u)$ is the nonlinear flux function and $u_0(x)$ is the initial state.

It is well known that conservation laws of the form (1.3) do not, in general, possess classical solutions. Instead, one must consider weak solutions in the sense that the following integral equality holds [4, 6, 7]

$$\int_{\Omega_x} \int_0^T \Big[u\phi_t + f(u)\phi_x \Big] \, dx \, dt + \int_{\Omega_x} u_0(x)\phi(x, t = 0) \, dx = 0 \tag{1.4}$$

for all $\phi \in C^1$ such that $\phi(x, t) : \Omega_x \times (0, T) \to \mathbb{R}$ and which is compactly supported, i.e., ϕ vanishes at $x \to \Omega_x$ and $t \to T$. It follows that if a discontinuity occurs in the solution, i.e., a left state u_L and a right state u_R, then it must propagate with the speed s given by [4, 7]

$$s = \frac{f(u_L) - f(u_R)}{u_L - u_R}. \tag{1.5}$$

However, direct calculations show that there are several weak solutions for one and the same initial data [6]. To overcome this issue of the non-uniqueness of weak solutions, this has led to the class of *entropy* solutions, which amounts to introducing an additional constraint that ensures that the unique physically relevant one is found among all the possible weak solutions.

There are different ways to express the entropy condition for scalar nonlinear conservation laws. One variant is by introducing an entropy pair (η, q) where $\eta : \mathbb{R} \to \mathbb{R}$ is any strictly convex function and $q : \mathbb{R} \to \mathbb{R}$ is constructed as [7, 8]

$$q(v) = \int_0^v f'(s)\eta'(s) \, ds \tag{1.6}$$

for any v. This implies that $q' = f'\eta'$. Then, u is an entropy solution of (1.3) if (i) u is a weak solution in the sense of (1.4); (ii) u satisfies in a weak sense $\eta(u)_t + q(u)_x \leq 0$ for any pair (η, q). This condition can also be formulated as the following characterization of a discontinuity (u_L, u_R) [7, 9]: For all numbers v between u_L and u_R,

$$\frac{f(v) - f(u_L)}{v - u_L} \geq s \geq \frac{f(v) - f(u_R)}{v - u_R} \tag{1.7}$$

where s is given by (1.5).

1.3.2 The One-Dimensional Scalar Conservation Law with Variable Parameters

The one-dimensional case is given by

$$\begin{aligned} u_t + f(u, \beta)_x &= 0, \qquad\qquad x \in (0, L) \\ u|_{t=0} &= u_0(x), \\ u_x|_{x=0} &= u_x|_{x=L} = 0, \end{aligned} \tag{1.8}$$

where $u = u(x,t)$ is the main variable and β is a parameter, typically, related to different forces that drive the process under consideration. Note that this extension to include dependence on a parameter β is highly relevant for many situations where different physical parameters naturally vary over certain intervals. Displacement of several fluids is one classical example where quantities like fluid viscosity and density may strongly influence the displacement behavior and their roles are expressed in a β-like parameter [10, 11]. In our research scenario, we seek the analytical expression of $f(u,\beta)$ that includes both the variable u and the parameter β, given a set of observations $\{u(x_j,t_i)\}$ at various time points t_i and at different values of β over an interval of interest.

1.3.3 The Two-Dimensional Scalar Conservation Law

The two-dimensional scalar conservation law is expressed by

$$\begin{aligned} u_t + f(u)_x + g(u)_y &= 0, \qquad (x,y) \in \Omega = (0,L)\times(0,L) \\ u|_{t=0} &= u_0(x,y), \\ u_x|_{x=0} &= u_x|_{x=L} = 0, \\ u_y|_{y=0} &= u_y|_{y=L} = 0. \end{aligned} \tag{1.9}$$

where u is defined as the main variable of space and time, and $f(u)$ and $g(u)$ are flux functions. The objective, in this case, is to find analytical expressions for both $f(u)$ and $g(u)$, based on a set of observations $u(x_i,y_j,t_l)$ at various time points t_l.

1.3.4 The One-Dimensional Scalar Conservation Law with Diffusion Term

The scalar nonlinear convection-diffusion PDEs flow in the following form

$$\begin{aligned} u_t + f(u)_x &= \alpha A(u)_{xx}, \quad A(u) = \int_0^u a(v)dv, \quad a(v) \geq 0 \\ u|_{t=0} &= u_0(x), \\ u_x|_{x=0} &= u_x|_{x=L} = 0, \end{aligned} \tag{1.10}$$

where $x \in [0,L]$ arise in different applications such as sedimentation of particles in liquid and various traffic flow types of problems. Here $u = u(x,t)$ is the main variable which depends on the position x and time t. The flux function $f(u)$ represents the convective transport whereas the diffusion function, denoted by $A(u) = \int_0^u a(v)dv$, is a function of u that describes the diffusive transport. For a typical situation, we have a priori information about the magnitude of the scaling factor $\alpha > 0$ but not precise information about the functional form of neither $A(u)$ nor $f(u)$. Our goal is to determine analytical expressions for both $f(u)$ and $A(u)$ using observational data of different types.

Chapter 2

Literature Review

2.1 Research Methods for Non-Machine Learning

There are several examples of non-machine approaches to nonlinear dynamics discovery from sparse data, especially for learning the flux function of a nonlinear conservation law. James and Sepúlveda formulated the inverse problem of flux identification as that of minimizing a suitable cost function [12]. Relying on the viscous approximation, it was shown that the perturbed problem converged to the original hyperbolic problem [12] by letting the viscous term vanish. Holden et al. used the front-tracking algorithm to reconstruct the flux function from observed solutions to problems with suitable initial data [13]. Several recent studies have addressed the reconstruction of the flux function for sedimentation problems that involve the separation of a flocculated suspension into a clear fluid and a concentrated sediment [14, 15]. In particular, Bürger and Diehl showed that the inverse problem of identifying the batch flux density function has a unique solution, and derived an explicit formula for the flux function [16]. This method was recently extended to construct almost the entire flux function [17] by using a cone-shaped separator. For another interesting example of the challenge of identification of the unknown flux function $f(u)$, we refer to [18]. The author explored a direct inversion method based on using linear combinations of finite element hat functions to represent unknown nonlinear function f.

2.2 Research Methods for Machine Learning

Thuerey et al. in their work [1] categorized physics-based deep learning methods into three groups based on the degree of interaction between DL techniques and domain knowledge, for example, in the sense of the generic form of the underlying PDE model.

2.2.1 Supervised Methods

Supervised methods exhibit minimal interaction with domain knowledge and align with the conventional machine learning paradigm. In these methods, data is generated by a physical system without any subsequent interaction. The advancement of machine learning, data science, and computing power has enhanced data-driven nonlinear dynamics discovery. Both [19] and [20] used Genetic Programming (GP) based Symbolic Regressors [21, 22] to deduce the differential equations representing underlying physical laws using combinations of mathematical operators. Later, this type of method inspired a series of endeavors [23–29]. Brunton et al. [30] approached dynamical system discovery through

sparse regression to prevent overfitting of GP. Specifically, they introduced a method called Sparse Identification of Nonlinear Dynamics (SINDy) that uses a sequential threshold ridge regression algorithm to iteratively find a sparse solution from a predefined basis function library until convergence. SINDy quickly emerged as one of the leading methods in this field of study, igniting significant interest [31–35]. Recently, more and more GNNs methods are being used to solve PDE problems. Gao et al. presented a novel discrete PINN framework based on Graph Convolutional Networks (GCNs) and the variational structure of PDEs in [36]. This framework could solve forward and inverse PDEs in a unified manner. Zhao et al. used GNNs in conjunction with autodecoder style priors to tackle PDE-constrained inverse problems [37]. In [38], authors build PDE solvers based on GNNs using the data generated by some classical numerical methods, such as finite differences, finite volumes, and WENO schemes. They experimentally proved that this method has fast, stable, and accurate performance across different domain topologies on various fluid-related flow problems. However, these current works mainly use GNNs on some typical equations with regular solutions. Several complex equations with discontinuous solutions still need to be researched.

Supervised Methods usually involve optimizing some criterion like the squares error to find the best parameters for the model that best fits the observed data. These methods are well-established, and there are a variety of techniques available. However, these methods require a substantial amount of labeled data. They might not capture the underlying physics of the system, especially if the data is noisy or sparse.

2.2.2 Physics-informed Methods

In the physics-informed methods, the physical dynamics are encoded in the loss function, typically in the form of differentiable operations. The learning process can repeatedly evaluate the loss and usually receives gradients from a PDE-based formulation. Raissi et al. [39] introduced physics-informed neural networks (PINN) using deep learning. They embedded the PDE residual into the loss function of fully-connected neural networks as a regularizer, facilitating training in small data scenarios. The weakly supervised approach has been utilized to solve various PDEs using limited training data in scientific domains like subsurface flows [40], vortex-induced vibrations [41], turbulent flows [42, 43], cardiovascular systems [44–47], metamaterial design [48–50], geostatistical modeling [51], and more [52–54]. To address sparse, noisy data and measure aleatoric uncertainty from noise, Sun and Wang [47] introduced a Bayesian formulation of physics-constrained learning using variational inference. Yang et al. [55] further explored Bayesian PINN, comparing both variational inference and Hamiltonian Monte Carlo approaches. While the aforementioned works need a moderate amount of training data, enforcing proper initial and boundary conditions can eliminate the data requirement altogether. The effectiveness of the data-free fully-connected neural networks based PDE solution algorithm has been demonstrated on a number of canonical PDEs [56, 57] and stochastic PDEs [58–60]. However, the neural network methods struggle in learning the nonlinear hyperbolic PDE that governs two-phase transport in porous media [11]. They experimentally indicate that this shortcoming of PINN for hyperbolic PDEs is not related to the specific architecture or to the choice of the hyperparameters but is related to the lack of regularity in the solution. The works based on the PINN approaches mentioned above rely on imposing the structure of the underlying PDE model by including an error term in the loss function. Due to the lack of regularity in the solution as illustrated in Section 1.1, it seems unclear how to implement our problems in a PINN framework.

Physics-informed Methods employ techniques like PINNs, where the network is trained not only to fit the data but also to satisfy known physical laws. By leveraging known physics, these methods can often produce more accurate and generalizable models, especially when data might be sparse or noisy. However, they require a prior understanding of the physical principles governing the system. Implementing them can also be more complex than Supervised Methods.

2.2.3 Interleaved Methods

Interleaved techniques integrate the full physical model with an output from a deep neural network. In our methods proposed in this thesis, the full physical model refers to the numerical scheme, while the deep neural network output represents the learned unknown functions. The physical simulation process is iterative step by step over time. During each step, the output of the neural network is involved. In Supervised Methods, data is first generated by the physical simulation and then used to train the neural network. This data generation is separate from the network's training, with no new data produced during training. In Physics-informed Methods, the physical dynamics are encoded in the loss function for the neural network training. Each update of the network parameters incorporates the dynamics once. However, in Interleaved Methods, each update of the neural network parameters encompasses multiple time-step iterations of the physical simulation, showcasing the tightest integration between the physical system and learning. These differentiable physics interleaving strategies are particularly crucial for temporal evolutions, enabling predictions of future dynamic behaviors. Chen et al. proposed the Symplectic Recurrent Neural Network (SRNN) that captures the dynamics of physical systems from regularly observed data [2]. This method works for Hamiltonian functions as it requires the variables of the ODE system to be separable. Long et al. [61, 62] proposed a combination of numerical approximation of differential operators by convolutions and a symbolic multi-layer neural network for model recovery. They used convolutions to approximate differential operators with properly constrained filters and to approximate the nonlinear response by deep neural networks. However, an essential difference between the problem studied in [61, 62] and the conservation law (1.1) is that the flux function $f(u)_x$ cannot be expressed by $f'(u)u_x$ as $f(u)$ is not, in general, a differentiable function in our problems.

Interleaved Methods involve alternating between data-driven identification (like Supervised Methods) and model-driven identification (like Physics-informed Methods). These methods are designed to be adaptable to varying types of data and system knowledge. For instance, one could leverage labeled data where it's available and fill in gaps with model-driven insights where data might be sparse. In addition, Interleaved Methods can incorporate the case where the system dynamics are less understood and governed by unknown physics. The significant challenges with Interleaved Methods are the increased complexity in implementation and more computational resources. Moreover, the balance between different methodologies needs to be well-managed to avoid pitfalls from either side.

ConsLaw-Net, introduced in this thesis, merges Symbolic Neural Networks with the numerical scheme, incorporating these networks into each iteration. Thus, ConsLaw-Net classifies as an Interleaved Method. Compared to non-machine learning, the primary objective of machine learning methods, including ConsLaw-Net, is to generalize from data, i.e., to learn patterns from data that can be applied to unseen data, for solving inverse problems. The focus of non-machine techniques is on finding the best solution

or parameter set that minimizes an objective function, often without an inherent goal of generalization. For instance, machine learning stops early to avoid overfitting and helps the model generalize better to unseen data. In terms of solution uniqueness, the traditional approach to solving convex optimization problems has a unique global minimum, ensuring solution consistency. However, multiple local minima might exist in machine learning, and the final model parameters can vary depending on initialization and training dynamics. The difference between machine learning methods and traditional optimization methods is also reflected in the training process, the dependence on data, and so on. The choice between the two should be based on the problem's nature, the available data, and the specific objectives and constraints of the task.

Chapter 3

Methodology

We primarily studied methods like the Interleaved Methods represented by ConsLaw-Net and the Supervised Methods typified by Graph Neural Networks (GNNs).

3.1 ConsLaw-Net

Our core method, ConsLaw-Net, primarily comprises two vital components: Symbolic Neural Networks and the Entropy Consistent Discrete Numerical Scheme.

3.1.1 Symbolic Neural Networks

We aim to decipher the analytical expression of unknown functions rather than merely fitting observations with black-box neural networks. To accomplish this, we employ an enhanced symbolic neural network, S-Net, tailored to reveal these functions.

S-Net-M and S-Net-D

In the S-Net setting, we design two network structures based on the function form we pursue: S-Net-M for multiplicative forms and S-Net-D for fractional forms, as depicted in Fig. 3.1 (top) and (bottom), respectively. Take a three layers S-Net which can learn the expression of a function f possessing a multiplication form as an example. As shown in Fig. 3.1 (top), the identity directly maps u from the input layer to the first hidden layer. The linear combination map uses parameters $\mathbf{w_1}$ and $\mathbf{b_1}$ to choose two elements from u and are denoted by α_1 and β_1

$$(\alpha_1, \beta_1)^T = \mathbf{w_1} \cdot (u) + \mathbf{b_1}, \mathbf{w_1} \in \mathbb{R}^{2\times 1}, \mathbf{b_1} \in \mathbb{R}^{2\times 1}. \tag{3.1}$$

These two elements of α_1 and β_1 are multiplied in the PDE system

$$f_1 = \alpha_1 \beta_1. \tag{3.2}$$

Apart from u gotten by the identity map, f_1 also is input to the second hidden layer

$$(\alpha_2, \beta_2)^T = \mathbf{w_2} \cdot (u, f_1)^T + \mathbf{b_2}, \mathbf{w_2} \in \mathbb{R}^{2\times 2}, \mathbf{b_2} \in \mathbb{R}^{2\times 1}. \tag{3.3}$$

Similar to the first hidden layer, we get another combination $f_2(\alpha_2, \beta_2)$

$$f_2 = \alpha_2 \beta_2. \tag{3.4}$$

Then we obtain α_3 and β_3 by means of $\mathbf{w_3}$ and $\mathbf{b_3}$ from u, f_1 and f_2

$$(\alpha_3, \beta_3)^T = \mathbf{w_3} \cdot (u, f_1, f_2)^T + \mathbf{b_3}, \mathbf{w_3} \in \mathbb{R}^{2\times 3}, \mathbf{b_3} \in \mathbb{R}^{2\times 1}. \tag{3.5}$$

f_3, which is the product of α_3 and β_3 is put into the third hidden layer

$$f_3 = \alpha_3 \beta_3. \tag{3.6}$$

Finally, we arrive at the analytic expression of the function f

$$f = \mathbf{w_4} \cdot (u, f_1, f_2, f_3)^T + \mathbf{b_4}, \mathbf{w_4} \in \mathbb{R}^{1\times 4}, \mathbf{b_4} \in \mathbb{R}. \tag{3.7}$$

The difference between S-Net-M and S-Net-D is that in the third hidden layer. we obtain the numerator part f_3 and the denominator part f_4 of the flux function $f(u)$ based on $\mathbf{w_3}$, $\mathbf{b_3}$ and $\mathbf{w_4}$, $\mathbf{b_4}$, respectively.

$$f_3 = \mathbf{w_3} \cdot (u, f_1, f_2)^T + \mathbf{b_3}, \mathbf{w_3} \in \mathbb{R}^{1\times 3}, \mathbf{b_3} \in \mathbb{R}. \tag{3.8}$$

$$f_4 = \mathbf{w_4} \cdot (u, f_1, f_2)^T + \mathbf{b_4}, \mathbf{w_4} \in \mathbb{R}^{1\times 3}, \mathbf{b_4} \in \mathbb{R}. \tag{3.9}$$

The analytic expression of the flux function f is the combination of f_3 and f_4.

$$f = \frac{f_3}{f_4}. \tag{3.10}$$

The Rational Reasons for Employing S-Net

Besides S-Net, other classical methods explicitly represent functions, such as the piecewise affine functions method described in [18] and the finite polynomial expansion method. In the piecewise affine functions method, divide the u-axis into n equidistant intervals: let u^{max} be the largest value of the data and set $u_k = ku^{max}/n$ for $k = -1, ..., n+1$. Assume that

$$f(u) = \sum_{k=0}^{n} f_k \psi_k(u) \tag{3.11}$$

where f_k is $n+1$ parameters to be determined, and the hat functions are defined by

$$\psi_k(u) = \begin{cases} \frac{u-u_{k-1}}{u_k-u_{k-1}}, & u_{k-1} < u \le u_k \\ \frac{u_{k+1}-u}{u_{k+1}-u_k}, & u_k < u \le u_{k+1} \\ 0, & \text{Otherwise} \end{cases} \tag{3.12}$$

where $k = 0, ..., n$. The polynomial expression with respect to u is given by:

$$f(u) = \frac{c_0 + c_1 u + c_2 u^2 + c_3 u^3 + ... + c_n u^{n_1}}{d_0 + d_1 u + d_2 u^2 + d_3 u^3 + ... + d_n u^{n_2}} \tag{3.13}$$

We acquire the coefficients $\{c_i\}_{i=0}^{n_1}$ and $\{d_i\}_{i=0}^{n_2}$ of (3.13) by using the observation data set. From the various methods available, we have decided to implement the S-Net approach for the following reasons:

- Experimental performance: In our work [63], we do a comparison of S-Net, the piecewise affine function representation, and the finite polynomial expansion method. It is observed that the performance of S-Net is clearly better than these two other methods for our test cases.

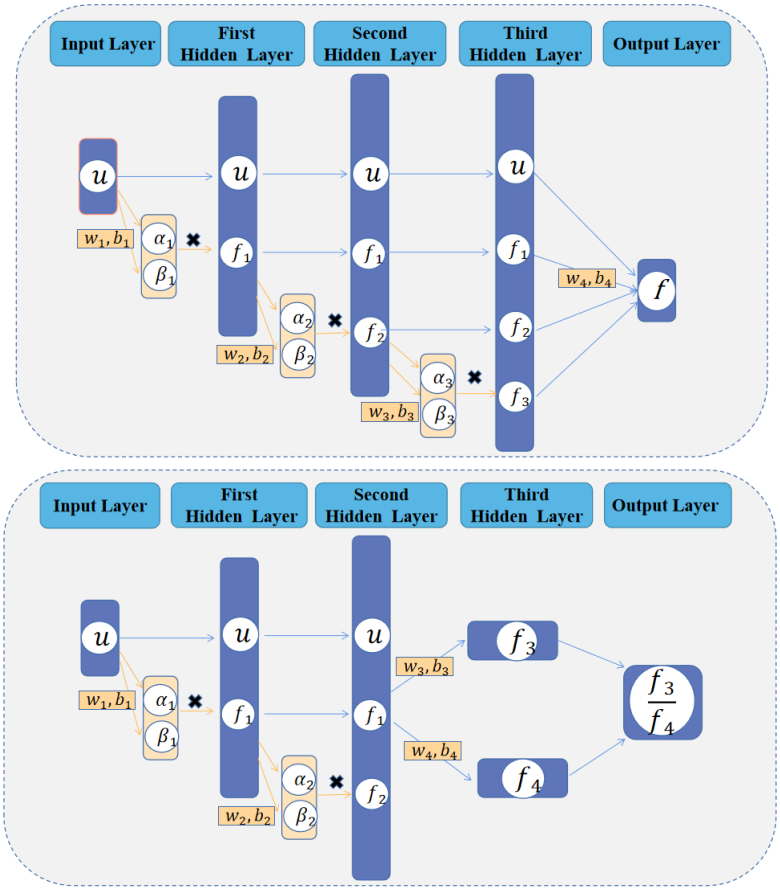


Figure 3.1: Top. The framework of S-Net-M for multiplicative function. **Bottom**. The framework of S-Net-D for division function.

- Robustness: S-Net demonstrates a high capacity to handle noisy and incomplete data [64].

- Flexibility: S-Net is flexible and powerful. Expanding functions with more variables is easier with S-Net, as it can automatically detect nonlinear relationships between input variables. However, the finite polynomial expansion method requires the manual listing of all possible combinations, which becomes an exhaustive task as the number of variables increases, potentially leading to an exponential growth in the number of combinations. Furthermore, incorporating nonlinearities such as sine and cosine into S-Net is straightforward, enabling it to learn combinations of variables that act on sine or cosine, such as $sin(u^2v)$.

3.1.2 Entropy Consistent Discrete Numerical Scheme (ECDNS)

ECDNS for (1.3)

We consider a discretization of the spatial domain $[0, L]$ in terms of $\{x_i\}_{i=0}^{N_x-1}$ where $x_i = (1/2 + i)\Delta x$ for $i = 0, \ldots, N_x - 1$ with $\Delta x = L/N_x$. Furthermore, we consider time lines $\{t^n\}_{n=0}^{N_t}$ such that $N_t \Delta t = T$. We base our discrete version of (1.3) on the Rusanov

scheme [4] which takes the form

$$\begin{aligned} u_j^{n+1} &= u_j^n - \lambda(F_{j+1/2}^n - F_{j-1/2}^n), \qquad \lambda = \frac{\Delta t}{\Delta x}, \\ u_1^{n+1} &= u_2^{n+1}, \qquad u_{N_x}^{n+1} = u_{N_x-1}^{n+1}, \end{aligned} \tag{3.14}$$

with $j = 2, \ldots, N_x - 1$ and where the Rusanov flux takes the form

$$F_{j+1/2}^n = \frac{f(u_j^n) + f(u_{j+1}^n)}{2} - \frac{M_{j+1/2}}{2}(u_{j+1}^n - u_j^n), \tag{3.15}$$

where $M_{j+1/2} = \max\{|f'(u_j^n)|, |f'(u_{j+1}^n)|\}$. The CFL condition [4] determines the magnitude of Δt for a given Δx through the relation

$$\text{CFL} := \frac{\Delta t}{\Delta x} M \leq 1, M \sim \max_u |f'(u)|. \tag{3.16}$$

ECDNS for (1.8)

We investigate a discretization of the spatial domain $[0, L]$ in terms of $\{x_i\}_{i=0}^{N_x-1}$ where $x_i = (1/2 + i)\Delta x$ for $i = 0, \ldots, N_x - 1$ with $\Delta x = L/N_x$. Furthermore, we consider time lines $\{t^n\}_{n=0}^{N_t}$ with $N_t \Delta t = T$. Our discrete version of (1.8) is based on the Rusanov scheme [4] which takes the form

$$\begin{aligned} u_j^{n+1} &= u_j^n - \lambda(F_{j+1/2}^n - F_{j-1/2}^n), \qquad \lambda = \frac{\Delta t}{\Delta x}, \\ u_1^{n+1} &= u_2^{n+1}, \qquad u_{N_x}^{n+1} = u_{N_x-1}^{n+1}, \end{aligned} \tag{3.17}$$

with $j = 2, \ldots, N_x - 1$ and where the Rusanov flux takes the form

$$F_{j+1/2}^n = \frac{f(u_j^n; \beta) + f(u_{j+1}^n; \beta)}{2} - \frac{M_{j+1/2}^n}{2}(u_{j+1}^n - u_j^n), \tag{3.18}$$

where $M_{j+1/2}^n = \max\{|f_u(u_j^n; \beta)|, |f_u(u_{j+1}^n; \beta)|\}$.The CFL condition determines the magnitude of Δt for a given Δx through the constraint

$$\text{CFL} := \frac{\Delta t}{\Delta x} M \leq 1, \qquad M = \max_u |f_u(u; \beta)|. \tag{3.19}$$

ECDNS for (1.9)

We consider a discretization of the spatial domain $[0, L] \times [0, L]$ in terms of $\{x_i\}_{i=0}^{N_x-1}$ and $\{y_j\}_{j=0}^{N_y-1}$ where $x_i = (1/2 + i)\Delta x$ for $i = 0, \ldots, N_x - 1$ with $\Delta x = L/N_x$, $y_j = (1/2 + j)\Delta y$ for $j = 0, \ldots, N_y - 1$ with $\Delta y = L/N_y$. We use the same grid number in the X and Y directions, i.e., $N_x = N_y$, denoted as N_{xy}, so $\Delta x = \Delta y = \Delta xy$. Additionally, we consider time lines $\{t^n\}_{n=0}^{N_t}$ with $N_t \Delta t = T$. Our discrete version of (1.9) is based on the Rusanov scheme [4], which takes the form

$$\begin{aligned} u^{n+1}(x_i, y_j) &= u^n(x_i, y_j) - \lambda_x(F_{i+1/2,j}^{x,n} - F_{i-1/2,j}^{x,n}) \\ &\quad - \lambda_y(G_{i,j+1/2}^{y,n} - G_{i,j-1/2}^{y,n}), \\ u^n(x_1, :) &= u^n(x_2, :), \;\; u^n(x_{N_{xy}}, :) = u^n(x_{N_{xy}-1}, :), \\ u^n(:, y_1) &= u^n(:, y_2), \;\; u^n(:, y_{N_{xy}}) = u^n(:, y_{N_{xy}-1}), \end{aligned} \tag{3.20}$$

with $\lambda_x = \lambda_y = \frac{\Delta t}{\Delta xy}$ and where $i, j = 2, \ldots, N_{xy} - 1$ and the Rusanov fluxes take the form

$$\begin{aligned} F^{x,n}_{i+1/2,j} &= \frac{f(u^n(x_i, y_j)) + f(u^n(x_{i+1}, y_j))}{2} \\ &\quad - \frac{M^n_{i+1/2,j}}{2}(u^n(x_{i+1}, y_j) - u^n(x_i, y_j)), \\ G^{y,n}_{i,j+1/2} &= \frac{g(u^n(x_i, y_j)) + g(u^n(x_i, y_{j+1})}{2} \\ &\quad - \frac{M^n_{i,j+1/2}}{2}(u^n(x_i, y_{j+1}) - u^n(x_i, y_j)). \end{aligned} \tag{3.21}$$

We adopt a local estimation by using $M^n_{i+1/2,j} = \max\{|f'(u^n_{i,j})|, |f'(u^n_{i+1,j})|\}$ and $M^n_{i,j+1/2} = \max\{|g'(u^n_{i,j})|, |g'(u^n_{i,j+1})|\}$. The CFL condition determines the magnitude of Δt for a given Δx, Δy through the constraint

$$\begin{aligned} &\text{CFL} := \frac{\Delta t}{\min\{\Delta x, \Delta y\}}(M^x + M^y) \leq 1, \\ &M^x = \max_u |f_u(u)|, \quad M^y = \max_u |g_u(u)|. \end{aligned} \tag{3.22}$$

ECDNS for (1.10)

We discretize the spatial domain $[0, L]$ into N_x points $\{x_i\}_{i=0}^{N_x-1}$, where $x_i = (1/2+i)\Delta x$ for $i = 0, \ldots, N_x - 1$, and $\Delta x = L/N_x$. Additionally, we consider a set of time steps $\{t^n\}_{n=0}^{N_t}$ with $N_t \Delta t = T$. To discretize (1.10), we use the Rusanov scheme [4], which can be written as follows:

$$\begin{aligned} u^{n+1}_j &= u^n_j - \lambda(F^n_{j+1/2} - F^n_{j-1/2}) + \lambda(D_+A^n_{j+1/2} - D_+A^n_{j-1/2}), \\ \lambda &= \frac{\Delta t}{\Delta x} u^{n+1}_1 = u^{n+1}_2, \quad u^{n+1}_{N_x} = u^{n+1}_{N_x-1}, \end{aligned} \tag{3.23}$$

with $j = 2, \ldots, N_x - 1$ and $F^n_{j+1/2}$ and $D_+A^n_{j+1/2}$ take the forms

$$F^n_{j+1/2} = \frac{f(u^n_j) + f(u^n_{j+1})}{2} - \frac{M^n_{j+1/2}}{2}(u^n_{j+1} - u^n_j), \tag{3.24}$$

and

$$D_+A^n_{j+1/2} = \frac{A(u^n_{j+1}) - A(u^n_j)}{\Delta x} = \frac{\int_0^{u^n_{j+1}} a(z)dz - \int_0^{u^n_j} a(z)dz}{\Delta x}. \tag{3.25}$$

We adopt a local estimation by using $M^n_{j+1/2} = \max\{|f'(u^n_j)|, |f'(u^n_{j+1})|\}$. The CFL condition determines the magnitude of Δt for a given Δx,

$$\text{CFL} := \frac{\Delta t}{\Delta x}(M + \frac{2K}{\Delta x}) \leq 1, \qquad M = \max_u |f'(u)|, \qquad K = \max_u a(u). \tag{3.26}$$

3.2 GNNs

Recently, GNNs with encoder processor decoder architectures have emerged as fast and precise alternatives to principled solvers for standard equations with regular solutions. In this thesis, we test the performance of GNNs on conservation law problems. Herein, We

use (3.27) as an example to demonstrate how to use GNNs to predict the solutions of PDEs based on the observation data and parameters,

$$u_t + \beta u_x = g(u), \tag{3.27}$$

where $x \in X$, $t \in [0, T]$. We discretize the region X into equally spaced points $\{x_i\}_{i=0}^{N_x}$ and the time T into $\{t_k\}_{k=0}^{N_t}$. The node c_i corresponding to the position x_i contains the solution $u(x_i, t_k)$ at the time t_k. The edge l_{ij} between neighboring nodes x_i and x_j carries the information to update $u(x_i), u(x_j)$ for subsequent times. GNNs output a new vector $\boldsymbol{d}_i = (\boldsymbol{d}_i^1, \boldsymbol{d}_i^2, ...\boldsymbol{d}_i^K)$ based on the last K solution values of node c_i, i.e., $(u(x_i, t_{k-K+1}), u(x_i, t_{k-K+2}), ...u(x_i, t_k))$.

Encoder, denoted as $\boldsymbol{\epsilon}$, is multilayer perceptrons (MLPs), computing node embedding. The node c_i at the moment t_k can be characterized by the last K solution values $(u(x_i, t_{k-K+1}), ..., u(x_i, t_{k-1}), u(x_i, t_k))$, node position x_i, current time t_k, and parameters $\boldsymbol{\beta}_{PDE}$. Herein, $\boldsymbol{\beta}_{PDE} = (\beta)$ in (3.27). As shown in Fig. 3.2, the Encoder maps these features into node embedding vector, i.e.,

$$\boldsymbol{f}_i = \boldsymbol{\epsilon}([u(x_i, t_{k-K+1}), ..., u(x_i, t_{k-1}), u(x_i, t_k), x_i, t_k, \boldsymbol{\beta}_{PDE}]). \tag{3.28}$$

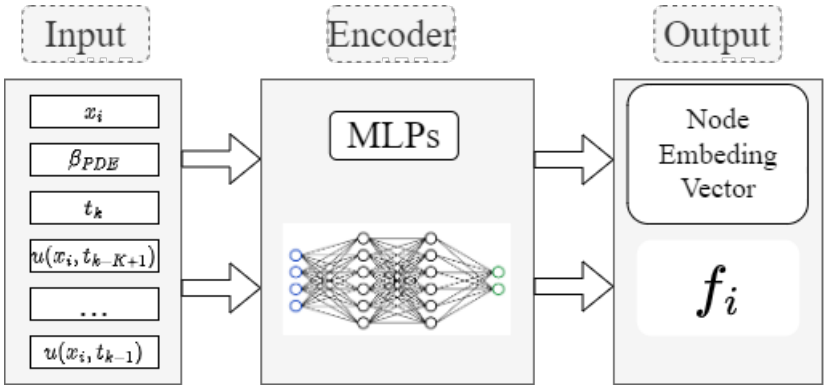


Figure 3.2: Encoder.

Processor performs the message passing

Processor performs the message passing by updating the information $\boldsymbol{m}_{ij}$ on the edge l_{ij} and the embedding vector $\boldsymbol{f}_i$ of node c_i using $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$, respectively. Processor contains M layers with intermediate graphs $\mathcal{G}^m$, $m = 1, 2, ..., M$. The messaging process is shown in Fig. 3.3.

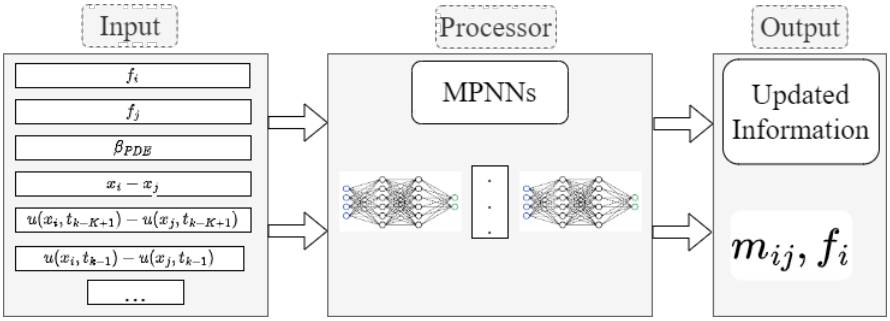


Figure 3.3: Processor.

- edge $c_j \to c_i$ message:

$$\begin{aligned} \boldsymbol{m}_{ij} = \boldsymbol{\phi}(\boldsymbol{f}_i, \boldsymbol{f}_j, u(x_i, t_{k-K+1}) - u(x_j, t_{k-K+1}) \\ , ..., u(x_i, t_k) - u(x_j, t_k), x_i - x_j, \boldsymbol{\beta}_{PDE}). \end{aligned} \tag{3.29}$$

- node c_i update:

$$\boldsymbol{f}_i = \boldsymbol{\psi}(\boldsymbol{f}_i, \sum_{j \in N(i)} \boldsymbol{m}_{ij}, \boldsymbol{\beta}_{PDE}), \tag{3.30}$$

where $N(i)$ holds the neighbors of node c_i and $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$ are MLPs.

Decoder uses a shallow 1D convolutional network to output the K next timesteps information increment $\boldsymbol{d}_i = (d(x_i, t_{k+1}), d(x_i, t_{k+2}), ..., d(x_i, t_{k+K}))$ at grid node c_i, as shown in Fig. 3.4.

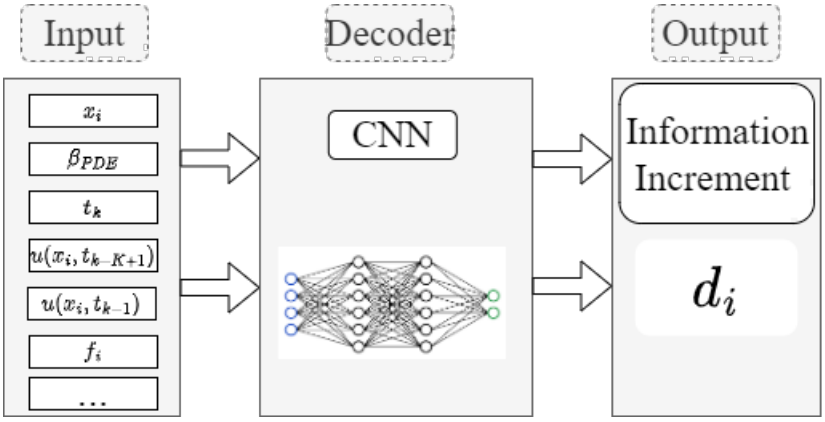


Figure 3.4: Decoder.

After getting the vector $\boldsymbol{d}_i$ from GNNs, we use it to update the next K solution values of node c_i as

$$u(x_i, t_{k+l}) = u(x_i, t_k) + (t_{k+l} - t_k) d(x_i, t_{k+l}) \tag{3.31}$$

where $1 \leq l \leq K$.

Chapter 4

Research Contributions

The main contribution of this work is six articles where four of which have been published whereas two of them are currently under review. In Paper I, we successfully addressed the parameterized ODE problem using a combination of S-Net and Euler Methods. In Paper II, we explored using GNNs—a subset of supervised methods—for conservation law problems with discontinuous solutions but found the generalization performance of GNNs is lacking. To enhance performance, in Paper III, we introduce ConsLaw-Net as a core method for the one-dimensional scalar conservation law problem. Building on ConsLaw-Net, Papers IV, V, and VI employ LRNN, Joint & Alternating equation training strategies, and multiple S-Nets to address the one-dimensional scalar conservation law with parameters, the two-dimensional version, and the one-dimensional variant with a diffusion term, respectively. Fig. 4.1 shows the relationship between the various works and the methods used.

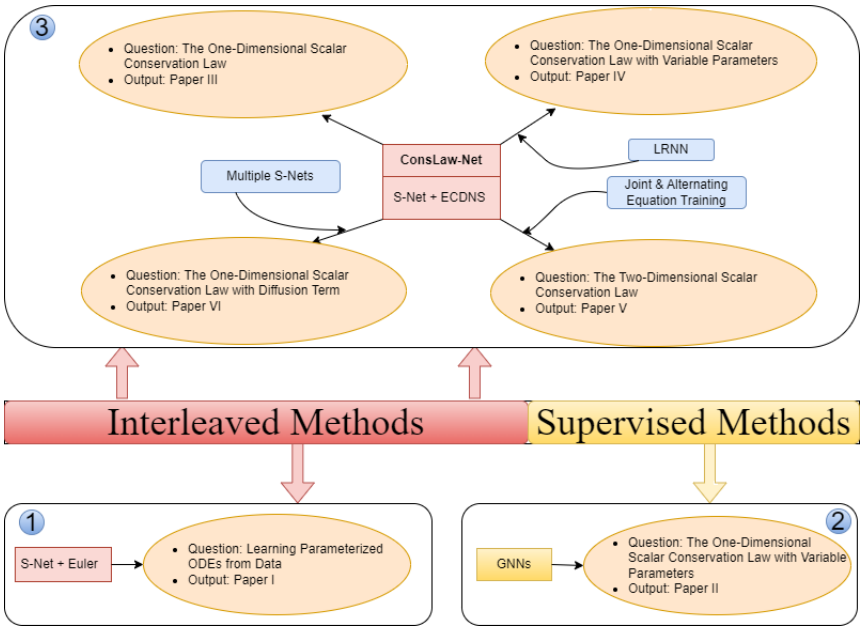


Figure 4.1: The relationship between the various works and the methods used.

4.1 Learning Parameterized ODEs from Data

In contemporary research, neural networks are being used to derive ODEs from observations. Assume that ODEs take the following generic form, which is usually used to describe some physical dynamics with different functions $f(s, p, r, \alpha, \beta, \gamma)$, $g(s, p, r, \alpha, \beta, \gamma)$ and $h(s, p, r, \alpha, \beta, \gamma)$,

$$\begin{cases} \frac{ds}{dt} = f(s, p, r, \alpha, \beta, \gamma), \\ \frac{dp}{dt} = g(s, p, r, \alpha, \beta, \gamma), \\ \frac{dr}{dt} = h(s, p, r, \alpha, \beta, \gamma). \end{cases} \tag{4.1}$$

Here, s, p, and r are independent variables and α, β, and γ are parameters. In the scenario of parameterized ODEs, we want to predict the system based on the new values of parameters. However, parameterized ODEs pose a more significant challenge than non-parameterized ODEs since the networks are required to understand the roles of the parameters, i.e., the structure of the equations. We proposed a novel approach by combining S-Net with ODE Solver to solve this issue. First, S-Net learns the structure of the parameterized ODEs and then predicts the dynamics based on the new parameters with the new initial states. Fig. 4.2 illustrates the time series generation process. Due to the ODE Solver, the system state transitions from the previous time point to the subsequent one. In our approach, S-Net serves as the ANN. To assess its performance, we compare our approach with a widely used Ordinary Neural Network (O-Net) that directly learns and predicts ODEs. Our numerical experiments demonstrate that our approach outperforms O-Net when applied to the Lotka-Volterra and Lorenz equations.

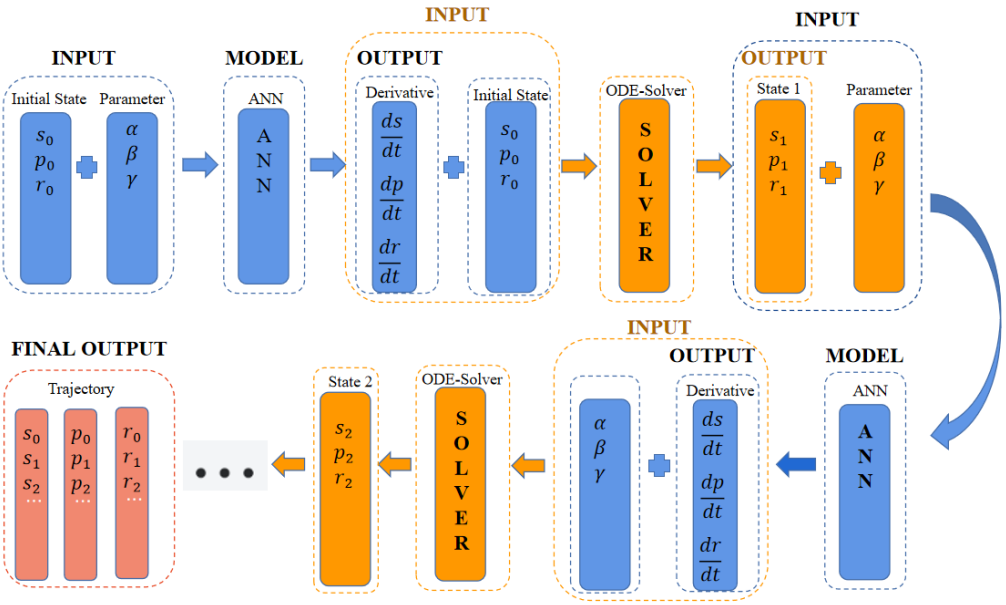


Figure 4.2: Schematic diagram of the framework. Initial states s_0, p_0, r_0 and parameters α, β, γ are fed into the ANN model to predict the derivatives of s, p, and r with respect to time. These derivatives combined with initial states s_0, p_0, r_0 will be further fed into the ODE Solver to obtain the next state s_1, p_1, r_1, and so on. Finally we will get the trajectories of $S = \{s_0, s_1, \ldots s_T\}$, $P = \{p_0, p_1, \ldots, p_T\}$ and $R = \{r_0, r_1, \ldots, r_T\}$.

4.2 Solving Nonlinear Conservation Laws of Partial Differential Equations Using Graph Neural Networks

Graph Neural Networks (GNNs) have recently been established as fast and accurate alternatives for principled solvers when applied to standard equations with regular solutions. There have been few investigations on GNNs implemented for complex PDEs with nonlinear conservation laws. Herein, we explore GNNs to solve the following problem

$$u_t + f(u, \beta)_x = 0 \tag{4.2}$$

where $f(u, \beta)$ is the nonlinear flux function of the scalar conservation law, u is the main variable, and β is the physical parameter. The main challenge of nonlinear conservation laws is that solutions typically create shocks. We leverage GNNs with encoder processor decoder structure to learn a fast-forward model that predicts solutions of PDEs. An overview of the approach that we use is shown in Fig. 4.3. Our experiments demonstrate

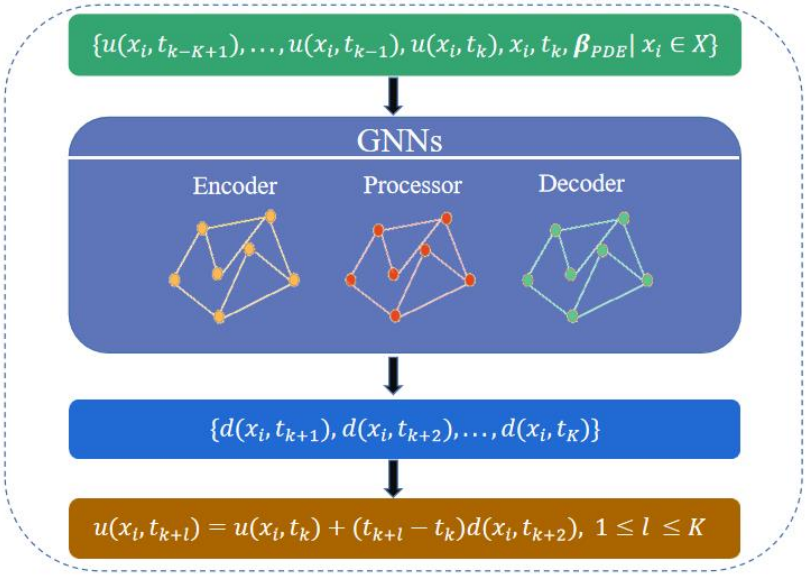


Figure 4.3: The pipeline of the approach. For each node c_i, at time point t_k, the last K solutions $(u(x_i, t_{k-K+1}), ..., u(x_i, t_{k-1}), u(x_i, t_k))$, position x_i, current time t_k, and equation parameters $\boldsymbol{\beta}_{PDE}$ are fed into GNNs. After the operators of *Encoder*, *Processor* and *Decoder*, we get the information increment of this node c_i at the next K time points: $(d(x_i, t_{k+1}), d(x_i, t_{k+2}), ..., d(x_i, t_{k+K}))$. Finally, we get solutions of the next K time points by $u(x_i, t_{k+l}) = u(x_i, t_k) + (t_{k+l} - t_k) d(x_i, t_{k+l}), 1 \leq l \leq K$.

that the GNN models can predict accurately when the parameter β is within a specific range. However, when the parameter deviates far from the value for training, the model's predictive performance drops sharply. Furthermore, the model is not very good at predicting the discontinuity of the solutions. There will be numerous small fluctuations around the discontinuity.

4.3 Learning the nonlinear flux function of a hidden scalar conservation law from data

Nonlinear conservation laws are widely used in fluid mechanics, biology, physics, and chemical engineering. However, deriving the nonlinear, unknown flux function $f(u)$ of the

following scalar conservation law is a significant and challenging problem,

$$\begin{aligned} u_t + f(u)_x &= 0, \quad x \in (0, L) \\ u_x|_{x=0} &= u_x|_{x=L} = 0, \\ u|_{t=0} &= u_0(x), \end{aligned} \tag{4.3}$$

with u as the main variable. This identification is based on using observation data $u(x_j, t_i)$ on a spatial grid $x_j,\ j = 1, \ldots, N_x$ at specified times $t_i,\ i = 1, \ldots, N_{obs}$. A main challenge with (4.3) is that the solution typically creates shocks, i.e., one or several jumps of the form (u_L, u_R) with $u_L \neq u_R$ moving in space and possibly changing over time such that information about $f(u)$ in the interval associated with this jump is sparse or not at all present in the observation data. Secondly, the lack of regularity in the solution of (4.3) and the nonlinear form of $f(u)$ hamper the use of previous proposed PINN methods where the underlying form of the sought differential equation is accounted for in the loss function. The overall architecture of the method is shown in Fig. 4.4. There are two parts involved, the generation of observed data based on the true flux function $f(u)$ and learning of the unknown flux function $f_\theta(u)$ which is assigned the S-Net structure. We proposed the method called ConsLaw-Net to circumvent this obstacle by approximating the unknown conservation law (4.3) by an entropy satisfying discrete scheme where $f(u)$ is represented through a symbolic multi-layer neural network. Numerical experiments show that the proposed method has the ability to uncover the hidden conservation law for a wide variety of different nonlinear flux functions, ranging from pure concave/convex to highly non-convex shapes. This is achieved by relying on a relatively sparse amount of observation data obtained in combination with a selection of different initial data.

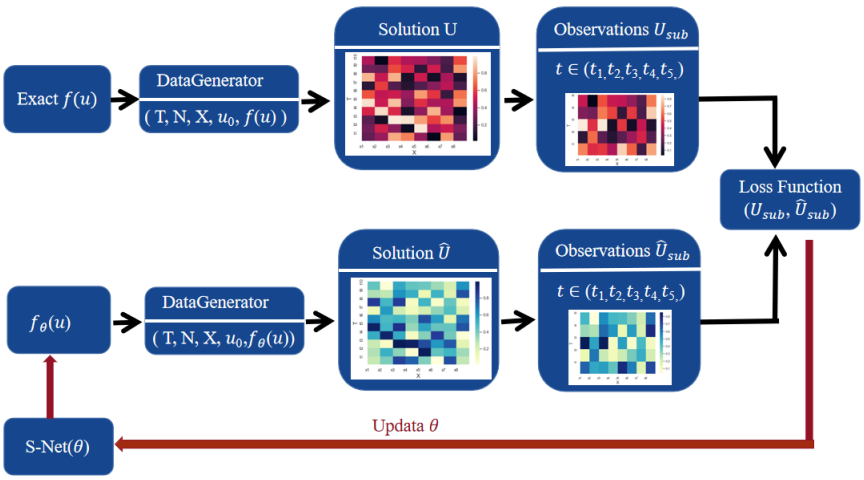


Figure 4.4: Schematic diagram of the framework.

4.4 Identification of the flux function of nonlinear conservation laws with variable parameters

The purpose of this work is to propose a method to identify the nonlinear flux function $f(u, \beta)$ with variable parameters of an unknown scalar conservation law

$$\begin{aligned} u_t + f(u, \beta)_x &= 0, \qquad x \in (0, L) \\ u|_{t=0} &= u_0(x), \\ u_x|_{x=0} &= u_x|_{x=L} = 0, \end{aligned} \tag{4.4}$$

with u as the dependent variable and β as the parameter. Learning the flux function with variable parameters, marked as $f(u, \beta)$, is more challenging than identifying $f(u; \beta)$ for various fixed β since it requires an understanding of the relationship between the variable u and parameter β as well. In this work, we couple the updated ConsLaw-Net to LRNN to learn the functional form of the two variable function $f(u, \beta)$. The framework for our approach is shown in Fig. 4.5. We experimentally demonstrate that the upgraded ConsLaw-Net integrated with LRNN is well-suited for learning tasks, achieving more accurate identification than existing learning approaches when applied to problems that involve general classes of flux functions $f(u, \beta)$. In addition, We compare our method with the GNNs method proposed in [65]. Experiments show that our method generates relatively accurate predictions whereas this GNN does not seem to have a built-in capacity to handle the problem (4.4) well.

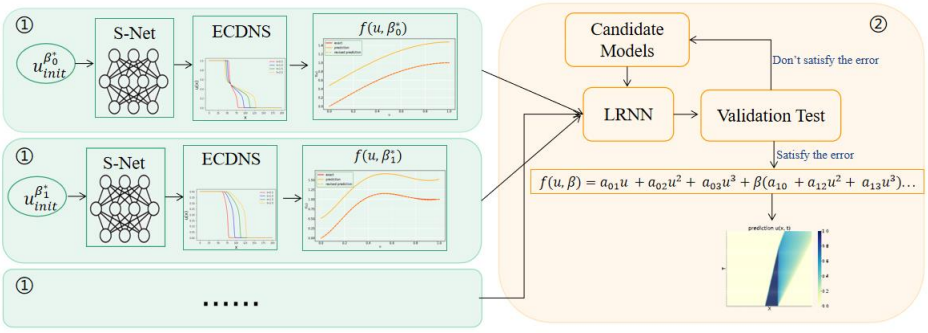


Figure 4.5: Pipeline for our approach. There are two steps: ① using the improved ConsLaw-Net to learn $f(u; \beta)$ where $\beta \in \{\beta_0^*, \beta_1^*, ..., \beta_{N_\beta}^*\}$. We use the symbol β^* to distinguish the values of β used in the first step. The learned expression $f(u; \beta_i^*)$ in each subproblem i is referring to an equation with u as variable but not β_i^*. ② Given the results of the first step, we first construct a collection of candidate models of the two variable function $f(u, \beta)$, and then use LRNN to obtain the coefficients of the selected candidate model. If this candidate model can satisfy the error criterion, we will apply it to compute the solutions of (4.4). Otherwise, we reselect the candidate model and repeat the process in the second step.

4.5 An alternating flux learning method for multidimensional nonlinear conservation laws

In this work, we focus on the problem of learning the unknown nonlinear flux functions from observations of the solution u of the two-dimensional scalar conservation law

$$\begin{aligned} u_t + f(u)_x + g(u)_y &= 0, \qquad (x, y) \in \Omega = (0, L) \times (0, L) \\ u|_{t=0} &= u_0(x, y), \\ u_x|_{x=0} &= u_x|_{x=L} = 0, \\ u_y|_{y=0} &= u_y|_{y=L} = 0. \end{aligned} \tag{4.5}$$

Given by (4.5), u is defined as the main variable of space and time, and the goal is to find analytical expressions for both $f(u)$ and $g(u)$, based on a set of observations $u(x_i, y_j, t_l)$ at various time points t_l. In addition to the lack of observational data due to the existence of some discontinuous points in the solution of the studied problem, the problem is further complicated by the need for high accuracy and the sparsity of useful information as the dimension increases. Therefore, we find that a straightforward extension of the method from the 1D to the 2D problem results in poor learning of the unknown $f(u)$ and $g(u)$.

Relying on ideas from *joint* and *alternating* equations training, we design learning strategies that enable accurate identification of the flux functions, even when 2D observations are sparse. As is shown in Fig. 4.6, it involves an *alternating flux learning* approach where a first set of candidate flux functions obtained from joint training is improved through an alternating direction-dependent learning strategy. Numerical investigations demonstrate that the method can effectively identify the true underlying flux functions f and g in the general case when they are non-convex and unequal.

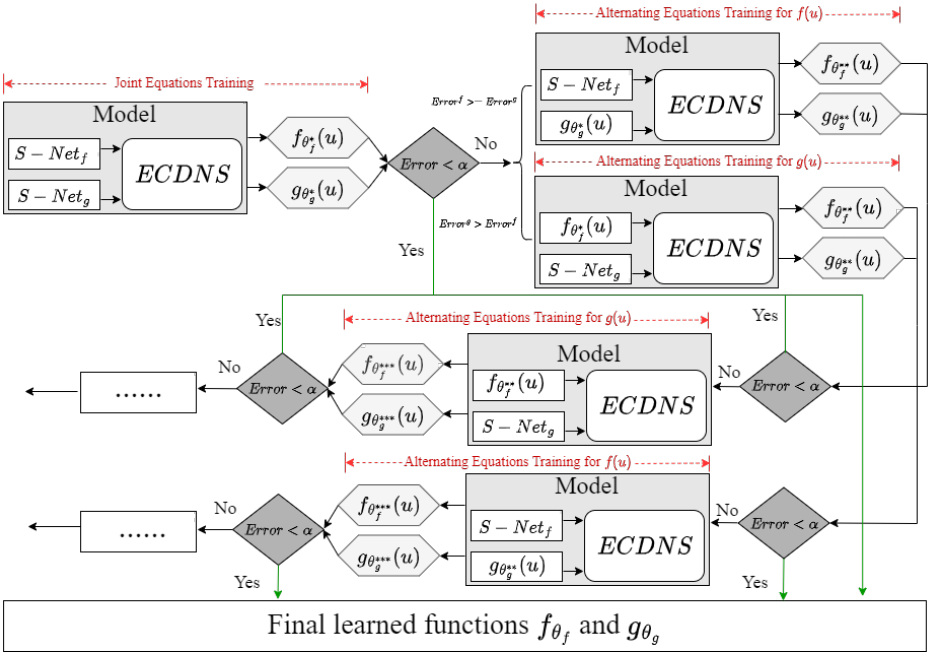


Figure 4.6: The framework based on Joint and Alternating Equations Training steps to learn our model.

4.6 Learning the flux and diffusion function for degenerate convection-diffusion equations using different types of observations

In this study, we investigate the identification of an unknown flux function and diffusion function in a one-dimensional convection-diffusion equation,

$$\begin{aligned} u_t + f(u)_x &= \alpha A(u)_{xx}, \qquad A(u) = \int_0^u a(v)dv, \qquad a(v) \geq 0 \\ u|_{t=0} &= u_0(x) \\ u_x|_{x=0} &= u_x|_{x=L} = 0 \end{aligned} \tag{4.6}$$

where $x \in [0, L]$ arises in different applications such as sedimentation of particles in liquid and various traffic flow types of problems. Here $u = u(x, t)$ is the main variable which depends on the position x and time t. The flux function $f(u)$ represents the convective transport whereas the diffusion function, denoted by $A(u) = \int_0^u a(v)dv$, is a function of u that describes the diffusive transport. For a typical situation, we have a priori information about the magnitude of the scaling factor $\alpha > 0$ but not precise information about the functional form of neither $A(u)$ nor $f(u)$. Our goal is to determine analytical expressions

for both $f(u)$ and $A(u)$ using observational data of different types. We introduce an updated ConsLaw-Net for learning the functional forms of both the flux function $f(u)$ and diffusion function $A(u)$ in the degenerate convection-diffusion model (4.6). Going beyond the reliance on the equation's solution in terms of $u(x_j, t_i)$ as observational data, we investigate the same physical system from a distinct perspective. Specifically, we consider a scenario where the system is made up of particles that follow the flow field as described by $u(x, t)$ governed by (4.6). That is, particle $z_j(t)$ is governed by

$$\dot{z_j}(t) = w(u(z_j(t), t)), \qquad z_j(t = 0) = x_j^0$$

for a given function $w(v)$. Using particle-based observational data, the proposed method generates quite remarkable outcomes despite the fact that the number of observation data points is much lower than the observations of $u(x, t)$. In addition, we conduct a comparative experiment using the piecewise affine functions method described in [18] replacing the S-Net. The experimental results demonstrate the clear advantage of neural networks in terms of effective and robust identification of the unknown functions f and A.

Chapter 5

Conclusion and Future Work

In this thesis, we introduce some new ideas for addressing complex physics problems. We present a unique framework merging the S-Net with an ODE Solver for identifying parameterized ODEs. The experiments validate the combined efficacy of the neural network and numerical scheme in determining equations in unknown dynamic systems, leading to precise state predictions. Additionally, we use the popular GNN approach, typically employed for standard equations with regular solutions, to test the conservation law problem. Jump points in these solutions create significant challenges due to incomplete information about the unknown function. However, the GNN's generalization performance fell short of our expectations. The core part of this thesis is the introduction of ConsLaw-Net, integrating the S-Net with an entropy-satisfying discrete scheme to address PDEs governed by the scalar conservation law. We later devise a framework combining the updated ConsLaw-Net with Linear Regression Neural Network to identify the conservation law's functional form with parameters. Understanding the flux function with variable parameters poses greater challenges than without, as it demands insight into the relationship between the variable and parameter. Additionally, we use joint and alternating equation strategies to train the advanced ConsLaw-Net for two-dimensional conservation law issues, which demand high accuracy amidst sparse valuable information. Finally, we present an enhanced ConsLaw-Net to learn the flux function and diffusion function at the same time in the degenerate convection-diffusion model, grounded on different types of observation data.

Physical discovery is a very ambitious and important field, and the approach of this thesis is a preliminary study of some of the fundamental issues. As far as the proposed method is concerned, there are still many areas that need to be further investigated:

a) There is a need for further testing on real-life data to what extent the proposed framework can be used to identify underlying conservation laws. The research in this thesis is based solely on synthetic data. Another interesting topic for further research is how different types of observation data (depending on the experimental system under consideration) might effect the learning. A first step in that direction is taken in Paper VI where we explore the use of particle paths that are governed by the solution of the hidden conservation law.

b) Find more flexible and less human-influenced components to replace S-Net. The parsimony of the S-Net outcome is primarily built on empirical pruning of the network weights, thus exhibiting sensitivity to user-defined network layers or even initial

parameters. Monte Carlo tree search (MCTS) agent has been used to identify the arithmetic expression of underlying physics in some studies [28, 29, 66]. It enables the flexible representation of search space with customized computational grammars to guide the search tree expansion. However, current work on MCTS mainly examines the ODE problems. Can we attempt to apply the MCTS algorithm as an alternative to S-Net for symbolic regression in ConsLaw-Net? Compared to S-Net, MCTS is able to reduce the influence of human factors on the model results, but how to encode the physical information into the selection policy to reduce the complexity of the tree search is also a challenge.

c) Theoretical insights into the field of discovery for dynamic systems are essential for future model designs. We verified experimentally that ConsLaw-Net succeeds in our scenario while GNNs don't. However, we haven't identified the model features causing this disparity. One direction of future work is to compare the difference between ConsLaw-Net and GNNs from the training process and loss landscape. In addition, the Frequency Principle proposed in [67] may be a useful tool for recognizing what kind of information GNNs can capture.

d) Most current research focuses on solving equations in low-dimensional spaces (under three dimensions). However, the solution manifold of parametric PDEs is frequently high-dimensional. One of the challenges in studying problems in high-dimensional space is the computational complexity. While deep learning can be used in this manifold, to do so effectively, an adequately expressive basis is essential to capture its features. A future research direction involves using more efficient learning strategies to train existing models. In paper V, we explore the Joint and Alternating equations training methods for problems in two-dimensional space.

Bibliography

[1] Nils Thuerey, Philipp Holl, Maximilian Mueller, Patrick Schnell, Felix Trost, and Kiwon Um. Physics-based deep learning. *arXiv preprint arXiv:2109.05237*, 2021.

[2] Zhengdao Chen, Jianyu Zhang, Martín Arjovsky, and Léon Bottou. Symplectic Recurrent Neural Networks. In *Proc. 8th Int. Conf. Learn. Represent. (ICLR), Addis Ababa, Ethiopia*, pages 1–23, Apr. 2020.

[3] Qing Li, Steinar Evje, and Jiahui Geng. Learning parameterized odes from data. *IEEE Access*, 2023.

[4] R.J. LeVeque. Finite volume methods for hyperbolic problems. *Cambridge Texts in Applied Mathematics*, 2007.

[5] H. Holden and N.H. Risebro. Front tracking for hyperbolic conservation laws. *Springer, Berlin*, 2011.

[6] J.W. Thomas. Numerical partial differential equations. conservation laws and elliptic equations. *Texts in Applied Mathematics 33*, 1999.

[7] J.S. Hesthaven. Numerical methods for conservation laws. from analysis to algorithms. *SIAM. Computational Science & Engineering*, 2017.

[8] D. Kröener. Numerical schemes for conservation laws. *Wiley-Teubner Series Advances in Numerical Mathematics*, 1997.

[9] M. Mishra, U.S. Fjordholm, and R. Abgrall. Numerical methods for conservation laws and related equations. *Lecture Notes, University in Oslo.*

[10] Hans Joakim Skadsem and Steinar Kragset. A numerical study of density-unstable reverse circulation displacement for primary cementing. *Journal of Energy Resources Technology*, 144(12):123008, 2022.

[11] Olga Fuks and Hamdi A Tchelepi. Limitations of physics informed machine learning for nonlinear two-phase transport in porous media. *Journal of Machine Learning for Modeling and Computing*, 1(1), 2020.

[12] F. James and M. Sepúlveda. Convergence results for the flux identification in a scalar conservation law. *SIAM J. Control Optim.*, 37(3):869–891, 1999.

[13] H. Holden, F.S. Priuli, and N.H. Risebro. On an inverse problem for scalar conservation laws. *Inverse Problems*, 30:035015, 2014.

[14] M. C. Bustos, F. Concha, R. Bürger, and E.M. Tory. *Sedimentation and thickening - Phenomenological Foundation and Mathematical Theory.* Kluwer Academic Publishers, 1999.

[15] S. Diehl. Estimation of the batch-settling flux function for an ideal suspension from only two experiments. *Chemical Engineering Science*, 62:4589–4601, 2007.

[16] R. Bürger and S. Diehl. Convexity-preserving flux identification for scalar conservation laws modelling sedimentation. *Inverse Problems*, 29:045008, 2013.

[17] R. Bürger, J. Careaga, and S. Diehl. Flux identification of scalar conservation laws from sedimentation in a cone. *IMA Journal of Applied Mathematics*, 83:526–552, 2018.

[18] Stefan Diehl. Numerical identification of constitutive functions in scalar nonlinear convection–diffusion equations with application to batch sedimentation. *Applied Numerical Mathematics*, 95:154–172, 2015.

[19] J. Bongard and H. Lipson. Automated reverse engineering of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 104(24):9943–9948, 2007.

[20] M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

[21] JRGP Koza. On the programming of computers by means of natural selection. *Genetic programming*, 1992.

[22] Lynne Billard and Edwin Diday. From the statistics of data to the statistics of knowledge: symbolic data analysis. *Journal of the American Statistical Association*, 98(462):470–487, 2003.

[23] Theodore Cornforth and Hod Lipson. Symbolic regression of multiple-time-scale dynamical systems. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, pages 735–742, 2012.

[24] Sébastien Gaucel, Maarten Keijzer, Evelyne Lutton, and Alberto Tonda. Learning dynamical systems using standard symbolic regression. In *Genetic Programming: 17th European Conference, EuroGP 2014, Granada, Spain, April 23-25, 2014, Revised Selected Papers 17*, pages 25–36. Springer, 2014.

[25] Daniel L Ly and Hod Lipson. Learning symbolic representations of hybrid dynamical systems. *The Journal of Machine Learning Research*, 13(1):3585–3618, 2012.

[26] Markus Quade, Markus Abel, Kamran Shafi, Robert K Niven, and Bernd R Noack. Prediction of dynamical systems by symbolic regression. *Physical Review E*, 94(1):012214, 2016.

[27] Harsha Vaddireddy, Adil Rasheed, Anne E Staples, and Omer San. Feature engineering and symbolic regression methods for detecting hidden physics from sparse sensor observation data. *Physics of Fluids*, 32(1), 2020.

[28] T Nathan Mundhenk, Mikel Landajuela, Ruben Glatt, Claudio P Santiago, Daniel M Faissol, and Brenden K Petersen. Symbolic regression via neural-guided genetic programming population seeding. *arXiv preprint arXiv:2111.00053*, 2021.

[29] Brenden K Petersen, Mikel Landajuela, T Nathan Mundhenk, Claudio P Santiago, Soo K Kim, and Joanne T Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871*, 2019.

[30] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

[31] Hayden Schaeffer. Learning partial differential equations via data discovery and sparse optimization. *Proc. Roy. Soc. Ser. A*, 473, 2017.

[32] M. Dam, M. Brøns, J. Juul Rasmussen, V. Naulin, and J. S. Hesthaven. Sparse identification of a predator-prey system from simulation data of a convection model. *Physics of Plasmas*, 24(2):022310, 2017.

[33] A. Narasingam and J. S. I. Kwon. Data-driven identification of interpretable reduced-order models using sparse regression. *Computers & Chemical Engineering*, 119:101–111, 2018.

[34] K. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton. Data-driven discovery of coordinates and governing equations. *Proceedings of the National Academy of Sciences*, 116(45):22445–22451, 2019.

[35] Z. Chen, Y. Liu, and H. Sun. Physics-informed learning of governing equations from scarce data. *Nature communications*, 12(1):6136, 2021.

[36] H. Gao, M. J. Zahr, and J. X. Wang. Physics-informed graph neural galerkin networks: A unified framework for solving pde-governed forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 390:114502, 2022.

[37] Q. Zhao, D. B. Lindell, and G. Wetzstein. Learning to solve pde-constrained inverse problems with graph networks. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 26895–26910. PMLR, 2022.

[38] Johannes Brandstetter, Daniel E. Worrall, and Max Welling. Message passing neural PDE solvers. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[39] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

[40] Nanzhe Wang, Dongxiao Zhang, Haibin Chang, and Heng Li. Deep learning of subsurface flow via theory-guided neural network. *Journal of Hydrology*, 584:124700, 2020.

[41] Maziar Raissi, Zhicheng Wang, Michael S Triantafyllou, and George Em Karniadakis. Deep learning of vortex-induced vibrations. *Journal of Fluid Mechanics*, 861:119–137, 2019.

[42] Xiaowei Jin, Shengze Cai, Hui Li, and George Em Karniadakis. Nsfnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible navier-stokes equations. *Journal of Computational Physics*, 426:109951, 2021.

[43] Maziar Raissi, Hessam Babaee, and Peyman Givi. Deep learning of turbulent scalar mixing. *Physical Review Fluids*, 4(12):124501, 2019.

[44] Georgios Kissas, Yibo Yang, Eileen Hwuang, Walter R Witschey, John A Detre, and Paris Perdikaris. Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4d flow mri data using physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 358:112623, 2020.

[45] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.

[46] Francisco Sahli Costabal, Yibo Yang, Paris Perdikaris, Daniel E Hurtado, and Ellen Kuhl. Physics-informed neural networks for cardiac activation mapping. *Frontiers in Physics*, 8:42, 2020.

[47] Luning Sun and Jian-Xun Wang. Physics-constrained bayesian neural network for fluid flow reconstruction with sparse and noisy data. *Theoretical and Applied Mechanics Letters*, 10(3):161–169, 2020.

[48] Yuyao Chen, Lu Lu, George Em Karniadakis, and Luca Dal Negro. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. *Optics express*, 28(8):11618–11633, 2020.

[49] Zhiwei Fang and Justin Zhan. Deep physical informed neural networks for metamaterial design. *Ieee Access*, 8:24506–24513, 2019.

[50] Dehao Liu and Yan Wang. Multi-fidelity physics-constrained neural network and its application in materials modeling. *Journal of Mechanical Design*, 141(12):121403, 2019.

[51] Qiang Zheng, Lingzao Zeng, and George Em Karniadakis. Physics-informed semantic inpainting: Application to geostatistical modeling. *Journal of Computational Physics*, 419:109676, 2020.

[52] Xiaoli Chen, Jinqiao Duan, and George Em Karniadakis. Learning and meta-learning of stochastic advection–diffusion–reaction systems from sparse measurements. *European Journal of Applied Mathematics*, 32(3):397–420, 2021.

[53] Zhiping Mao, Ameya D Jagtap, and George Em Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.

[54] Reda Snaiki and Teng Wu. Knowledge-enhanced deep learning for simulation of tropical cyclone boundary-layer winds. *Journal of Wind Engineering and Industrial Aerodynamics*, 194:103983, 2019.

[55] Liu Yang, Xuhui Meng, and George Em Karniadakis. B-pinns: Bayesian physics-informed neural networks for forward and inverse pde problems with noisy data. *Journal of Computational Physics*, 425:109913, 2021.

[56] Jens Berg and Kaj Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018.

[57] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. Deepxde: A deep learning library for solving differential equations. *SIAM review*, 63(1):208–228, 2021.

[58] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

[59] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

[60] Liu Yang, Dongkun Zhang, and George Em Karniadakis. Physics-informed generative adversarial networks for stochastic differential equations. *SIAM Journal on Scientific Computing*, 42(1):A292–A317, 2020.

[61] Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. Pde-net: Learning PDEs from data. In *International Conference on Machine Learning*, pages 3208–3216. PMLR, 2018.

[62] Zichao Long, Yiping Lu, and Bin Dong. PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, 399:108925, 2019.

[63] Qing Li, Jiahui Geng, and Steinar Evje. Identification of the flux function of nonlinear conservation laws with variable parameters. *Physica D: Nonlinear Phenomena*, 451:133773, 2023.

[64] Qing Li and Steinar Evje. Learning the nonlinear flux function of a hidden scalar conservation law from data. *Network Heterogeneous Media*, 18:48–79, 2023.

[65] V. Iakovlev, M. Heinonen, and H. Lähdesmäki. Learning continuous-time pdes from sparse data with graph neural networks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[66] Fangzheng Sun, Yang Liu, Jian-Xun Wang, and Hao Sun. Symbolic physics learner: Discovering governing equations via monte carlo tree search. *arXiv preprint arXiv:2205.13134*, 2022.

[67] Zhi-Qin John Xu, Yaoyu Zhang, and Tao Luo. Overview frequency principle/spectral bias in deep learning. *arXiv preprint arXiv:2201.07395*, 2022.

Paper I:
Learning Parameterized ODEs from Data

Qing Li, Steinar Evje, Jiahui Geng

IEEE Access 11 (2023): 54897 - 54909
doi = https://ieeexplore.ieee.org/document/10143183

IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 18 April 2023, accepted 27 May 2023, date of publication 2 June 2023, date of current version 7 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3282435

RESEARCH ARTICLE

Learning Parameterized ODEs From Data

QING LI[1], STEINAR EVJE[1], AND JIAHUI GENG[2], (Graduate Student Member, IEEE)
[1]Department of Energy and Petroleum Engineering, University of Stavanger, 4021 Stavanger, Norway
[2]Department of Electrical Engineering and Computer Science, University of Stavanger, 4021 Stavanger, Norway

Corresponding author: Qing Li (qing.li@uis.no)

This work was supported by the Department of Energy and Petroleum Engineering at the University of Stavanger, under Grant IN-12570.

ABSTRACT In contemporary research, neural networks are being used to derive Ordinary Differential Equations (ODEs) from observations. However, parameterized ODEs pose a more significant challenge than non-parameterized ODEs since the networks are required to understand the roles of the parameters, i.e., the structure of the equations. This paper proposes a novel approach by combining Symbolic Neural Network (S-Net) with ODE Solver to solve this issue. First, S-Net learns the structure of the parameterized ODEs and then predicts the dynamics based on the new parameters with the new initial states. To assess its performance, we compare our approach with a widely used Ordinary Neural Network (O-Net) that directly learns and predicts ODEs. Our numerical experiments demonstrate that our approach outperforms O-Net when applied to the Lotka-Volterra and Lorenz equations.

INDEX TERMS Parameterized ordinary differential equations, neural networks, ODE solver.

I. INTRODUCTION

The combination of deep learning and differential equations is a highly promising research direction. Deep learning can help uncover the underlying differential equations governing the behavior of many systems, such as electromagnetism, aerodynamics, weather prediction, and geophysics. Differential equation-guided network design can significantly improve model interpretability and generalization performance. The potential impact of this approach on scientific discovery and innovation is immense.

Chen et al. [1] introduced Neural Ordinary Differential Equations (NODEs), a new family of deep networks. They used a neural network to parameterize the derivative of the hidden state, serialized the neural network layers and parameters, and used the adjoint ODE method to optimize the neural network instead of back-propagation, which saved memory. Their work inspired research on variants of NODE methods, such as [2] and [3]. Other studies have also explored the potential of neural networks to learn and solve differential equations. For example, Chen et al. [4] proposed the Symplectic Recurrent Neural Network (SRNN) to capture the dynamics of physical systems from regularly observed data. Raissi et al. [5] introduced physics-informed neural networks (PINNs) to solve data-driven solutions and data-driven discovery problems of partial differential equations (PDEs). Long et al. [6], [7] proposed PDE-Net, a feed-forward deep network that predicts the dynamics of complex systems and uncovers the underlying hidden PDE models. PDE-Net implemented Symbolic Neural Network (S-Net) to learn the structure of PDEs and demonstrated powerful learning ability. Similarly, [8], [9] also proved the competitive generalization capability of S-Net. These works show the potential of deep learning in solving PDEs and discovering hidden models.

Parameterized ODEs are a class of ODEs with solutions that vary with parameters, representing multiple dynamics specified by input parameter instances. They have been extensively studied in computational science and engineering domains, such as fluid dynamics and the ideal pendulum system. In critical situations, computing high-fidelity solutions of parameterized ODEs is necessary, either for numerous input parameter instances or initial states. Data-driven methods have been used in recent years to estimate the evolution of dynamical systems over time, including different initial conditions [10] and system parameters [11]. To learn the latent dynamics of complex dynamical processes in computational physics, Lee and Parish [11] proposed encoder-decoder parameterized NODEs (PNODEs). Shimizu and Parish [12] presented the windowed space-time least-squares

The associate editor coordinating the review of this manuscript and approving it for publication was Derek Abbott.

Petrov-Galerkin method (WST-LSPG) for model reduction of nonlinear parameterized dynamical systems. WST-LSPG divides the time simulation into several windows and sequentially minimizes the discrete-in-time residual within its own unique low-dimensional space-time subspace. Lee and Trask [13] introduced POUNODEs, a new variant of NODEs with evolving model parameters. They modeled the evolution using partition-of-unity networks, allowing for greater flexibility in capturing the dynamics of complex systems.

To the best of our knowledge, the majority of existing approaches for solving parameterized ODEs rely on black-box methods, which employ neural networks directly to simulate its behavior and make predictions. Our proposed approach, on the other hand, aims to gain a deeper understanding of the underlying mechanics of the dynamic system by first learning the structure of the ODEs. This understanding enables us to predict the behavior of the dynamical system more accurately and efficiently for new parameters and initial states compared to black-box methods. The contributions of this work are summarized as follows:

- Our proposed framework combines S-Net with an ODE solver to learn parameterized ODEs. This novel approach enables accurate and efficient modeling of the dynamics of complex systems, even when the properties of the governing equations vary across multiple input parameters.
- To evaluate our approach, we compare its performance with that of a baseline model, O-Net, which represents a black-box approach. We empirically demonstrate the advantages of our proposed framework in terms of both accuracy and interpretability.

The remainder of this paper is organized as follows: Section II presents related works. The considered parameterized ODEs problem and the ODE Solver are briefly introduced in Section III. Section IV illustrates the proposed approach. We evaluate the performance of our method on two case studies: the Lotka-Volterra Equation in Section V and the Lorenz Lotka-Volterra Equation in Section VI. Finally, in Section VII, we summarize our findings and discuss potential avenues for future research.

II. RELATED WORK

Several studies have explored the use of Gaussian process regression to develop tailored functional representations for a given linear operator [14], [15], [16]. However, the local linearization of nonlinear terms in time and prior assumptions of Gaussian process regression limit the representation capacity of the model. Sparse regression, discussed in [17], [18], [19], and [20], overcomes this limitation by developing a dictionary of basic functions and partial derivatives that can accurately represent the data using sparsity-promoting techniques. However, the predictive and expressive capabilities of the dictionary are restricted since the sparse regression method necessitates predefining specific numerical approximations for spatial differentiation.

Mesh-based simulations have recently shown significant progress [21], [22], surpassing grid-based convolutional neural networks (CNNs) in terms of runtime and exhibiting greater adaptivity to the simulation domain. While several methods, such as AntisymmetricRNN [23] and the continuous-time Gated Recurrent Unit with a Bayesian update network [24], have leveraged the stability of underlying differential equations to capture long-term dependencies, they did not address the challenge of learning the expression of the equation to gain a deeper understanding of the underlying mechanism behind the observed data. In another work [25], the authors learned the unknown parameters of the ODE system by constructing certain time-related features, but this method did not address the expressiveness of the equation.

III. KNOWLEDGE

In this section, we'll define parameterized ODEs and differentiate them from non-parameterized ODEs. Additionally, we'll introduce the ODE Solver, a crucial component in our approach.

A. NON-PARAMETERIZED AND PARAMETERIZED ODES

Assume that ODEs take the following generic form, which is usually used to describe some physical dynamics with different functions $f(s, p, r, \alpha, \beta, \gamma)$, $g(s, p, r, \alpha, \beta, \gamma)$ and $h(s, p, r, \alpha, \beta, \gamma)$,

$$\begin{cases} \dfrac{ds}{dt} = f(s, p, r, \alpha, \beta, \gamma), \\ \dfrac{dp}{dt} = g(s, p, r, \alpha, \beta, \gamma), \\ \dfrac{dr}{dt} = h(s, p, r, \alpha, \beta, \gamma). \end{cases} \tag{1}$$

Here, we consider the nonlinear ODE system with independent variables s, p, and r, and initial states s_0, p_0, and r_0 at time t_0. The system also involves parameters α, β, and γ, where $t \in (0, T]$ represents the time interval of interest.

In non-parameterized ODEs, parameters such as α, β, and γ remain fixed during both model training and prediction, whether they are known or unknown. Thus, we assume that $\alpha = \alpha_1$, $\beta = \beta_1$, and $\gamma = \gamma_1$, and denote the observations as:

$$\begin{aligned} Z^{\text{fixed}} = \Big\{ \Big(& s(t_i; s_0^j, p_0^j, r_0^j, \alpha_1, \beta_1, \gamma_1), p(t_i; s_0^j, p_0^j, r_0^j, \alpha_1, \beta_1, \\ & \times \gamma_1), r(t_i; s_0^j, p_0^j, r_0^j, \alpha_1, \beta_1, \gamma_1) \Big) \Big| i = 1, \ldots, N_{\text{obs}}, \\ & \times j = 1, \ldots, M \Big\}. \end{aligned} \tag{2}$$

The observations are denoted by a set of $N_{\text{obs}} > 0$ time points, and the number of initial states is denoted by $M > 0$. If $(s_0^*, p_0^*, r_0^*) \in \{(s_0^j, p_0^j, r_0^j) | j = 1, \ldots M\}$, we predict $\big(s(t_*; s_0^*, p_0^*, r_0^*, \alpha_1, \beta_1, \gamma_1), p(t_*; s_0^*, p_0^*, r_0^*, \alpha_1, \beta_1, \gamma_1), r(t_*; s_0^*, p_0^*, r_0^*, \alpha_1, \beta_1, \gamma_1)\big)$ at $t_* > t_{N_{\text{obs}}}$. Otherwise, we predict the dynamics at $t_* \in (0, T]$. The non-parameterized ODEs problem has been widely studied in [2], [3], and [4].

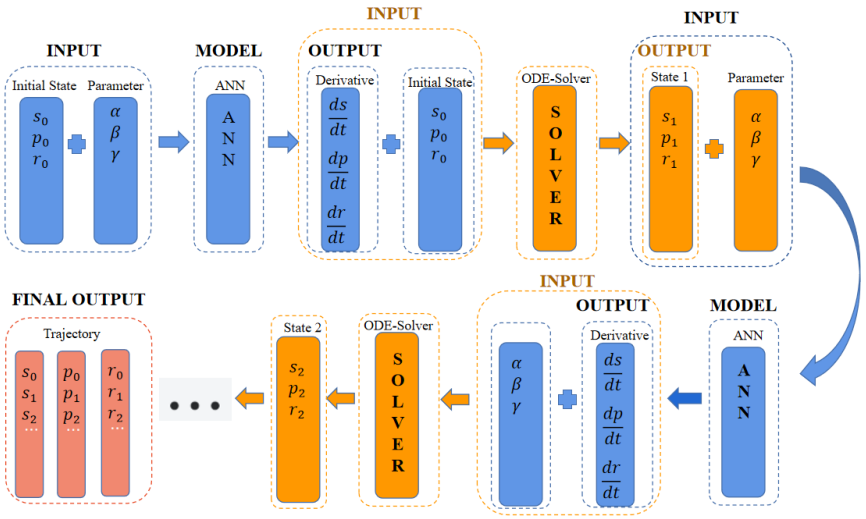


FIGURE 1. Schematic diagram of the framework. Initial states s_0, p_0, r_0 and parameters α, β, γ are fed into the ANN model to predict the derivatives of s, p, and r with respect to time. These derivatives combined with initial states s_0, p_0, r_0 will be further fed into the ODE Solver to obtain the next state s_1, p_1, r_1, and so on. Finally we will get the trajectories of $S = \{s_0, s_1, \ldots s_T\}$, $P = \{p_0, p_1, \ldots, p_T\}$ and $R = \{r_0, r_1, \ldots, r_T\}$.

Concerning the parameterized ODEs problem, the observation data

$$Z^{\text{variable}} = \Big\{\Big(s(t_i; s_0^j, p_0^j, r_0^j, \alpha_k, \beta_k, \gamma_k), p(t_i; s_0^j, p_0^j, r_0^j, \times \alpha_k, \beta_k, \gamma_k), r(t_i; s_0^j, p_0^j, r_0^j, \alpha_k, \beta_k, \gamma_k)\Big) \Big| i = 1, \times \ldots, N_{\text{obs}}, j = 1, \ldots, M, k = 1, \ldots, K\Big\}, \tag{3}$$

is generated by different parameters $(\alpha_k, \beta_k, \gamma_k)$ at some specified time t_i on different initial states (s_0^j, p_0^j, r_0^j). Here, we define $K > 0$ as the number of parameters, $N_{\text{obs}} > 0$ as the number of observation time points, and $M > 0$ as the number of initial states.

In the scenario of parameterized ODEs, if $(s_0^*, p_0^*, r_0^*) \in \{(s_0^j, p_0^j, r_0^j) | j = 1, \ldots M\}$ and $(\alpha_*, \beta_*, \gamma_*) \in \{(\alpha_k, \beta_k, \gamma_k) | k = 1, \ldots K\}$, we predict $\big(s(t_*; s_0^*, p_0^*, r_0^*, \alpha_*, \beta_*, \gamma_*), p(t_*; s_0^*, p_0^*, r_0^*, \alpha_*, \beta_*, \gamma_*), \quad r(t_*; s_0^*, p_0^*, r_0^*, \alpha_*, \beta_*, \gamma_*)\big)$ at $t_* > t_{N_{\text{obs}}}$. Otherwise, we predict the dynamics at $t_* \in (0, T]$.

Parameterized ODEs find applications in various scenarios. For instance, in [26], different parameters associated with the ODEs represent distinct strategies for regulating cancer tumor progression behavior. Given patient data describing the evolution of a tumor, it would be advantageous to obtain the functional form of the ODE system underlying this complex system, both for the state variables and parameters. Subsequently, once we have constructed the functional form of the ODE system using a neural network, we can directly compute solutions of the ODE for new initial states and parameter sets. Traffic flow problems represent another common application. Changes in traffic flow over time at two specific locations x_1, x_2 in a city can be modeled as

$$\begin{cases} \dfrac{ds}{dt} = f(s, p, \boldsymbol{w}), \\ \dfrac{dp}{dt} = g(s, p, \boldsymbol{w}). \end{cases} \tag{4}$$

In this context, the number of cars at positions x_1 and x_2 are represented by s and p, respectively, and the traffic flow changes over time at these locations are modeled using functions f and g. Here, $t \in (0, 24]$ denotes a day for a cycle, and the vector $\boldsymbol{w}$ represents external factors that influence traffic conditions, such as weather, temperature, and so on. We assume that the external factors $\boldsymbol{w}$ change daily, but the expressions for f and g remain constant as the mechanism by which each factor affects traffic flow remains the same. Once we obtain the correct analytical formulas, we can accurately predict the system.

This approach is effective for discrete chaotic maps, using the Logistic map as an example. The Logistic map is a straightforward, one-dimensional discrete-time dynamical system characterized by the following equation:

$$x_{n+1} = \gamma x_n (1 - x_n). \tag{5}$$

Here, x is the state variable at time step n, and γ is a parameter. We can employ the proposed method to learn the expression on the right side of (5), which means that the input of the S-Net consists of both γ and x.

B. ODE SOLVER

The ODE Solver, also known as a numerical integrator, iterates a numerical scheme to obtain improved approximations of the solution. There exist works dedicated to designing numerical integrators that produce more accurate solutions [27], [28]. Typically, the ODE Solver is employed to

approximate the true solution of the form $\frac{dz}{dt} = \boldsymbol{F}(z, t)$ based on the initial state z_0, where $\boldsymbol{F}(z, t)$ is a vector function. Considering the parameters are variable, parameters α, β, and γ can not be embedded in $\boldsymbol{F}$ in our problem. So $\boldsymbol{F}$ can be expressed in the form of $\boldsymbol{F}(z, t, \alpha, \beta, \gamma) = (f(z, t, \alpha, \beta, \gamma), g(z, t, \alpha, \beta, \gamma), h(z, t, \alpha, \beta, \gamma))$ and $z(t; z_0, \alpha, \beta, \gamma) = \{s(t; z_0, \alpha, \beta, \gamma), p(t; z_0, \alpha, \beta, \gamma), r(t; z_0, \alpha, \beta, \gamma)\}$.

In this paper, we begin with Euler integrator [29], a popular choice due to its simplicity, as demonstrated in previous works [1], [2], [13]. Euler method also facilitates a fair comparison with black-box methods. Given the state z_n at time point $t_n = t_0 + n\Delta t$, the state at the next time point can be computed using the following formula:

$$z_{n+1} = z_n + \Delta t \boldsymbol{F}(z_n, t_n, \alpha, \beta, \gamma). \tag{6}$$

The time step size, denoted by Δt, is a crucial parameter in numerical methods for solving ODEs. Specifically, Euler method can generate unstable solutions for stiff ODE systems unless a very small Δt is employed, as noted in [30]. To maintain solution stability in our experiments, we meticulously select an appropriate value for Δt. In Appendix A, we present a rigorous proof of the convergence of Euler method, which is essential for understanding the accuracy of numerical solutions. Our approach can also be adapted to learn unknown functions using semi-implicit solvers with appropriate modifications. In Appendix B, we provide a brief explanation of how to make the necessary adjustments to fit semi-implicit solvers.

IV. OUR APPROACH

A. PIPELINE

We use initial states $Z_0 = \left\{z_0^j | j = 1, \ldots, M\right\}$ and parameters $W = \{(\alpha_k, \beta_k, \gamma_k) | k = 1, \ldots, K\}$ for the training process. Following the work [1], we let the right side of ODEs be a parametric function $\boldsymbol{F}_{\boldsymbol{\theta}}(z, \alpha, \beta, \gamma)$, where $z = (s, p, r)$ and $\boldsymbol{\theta}$ is the vector of parameters of the neural network. After training, trajectories $\hat{Z} = \left\{\hat{z}_{\boldsymbol{\theta}}(t_i; z_0^j, \alpha_k, \beta_k, \gamma_k)\right\}_{i=1}^{N_{\text{obs}}}$ are generated based on initial state $z_0^j \in Z_0$, parameters $(\alpha_k, \beta_k, \gamma_k) \in W$ and $\boldsymbol{\theta}$, i.e.,

$$\begin{aligned}\hat{z}_{\boldsymbol{\theta}}(t_i; z_0^j, \alpha_k, \beta_k, \gamma_k) = &\hat{z}_{\boldsymbol{\theta}}(t_{i-1}; z_0^j, \alpha_k, \beta_k, \gamma_k) \\ &+ \Delta t \boldsymbol{F}_{\boldsymbol{\theta}}\left(\hat{z}_{\boldsymbol{\theta}}(t_{i-1}; z_0^j, \alpha_k, \beta_k, \gamma_k),\right. \\ &\left.\times \alpha_k, \beta_k, \gamma_k\right), \end{aligned} \tag{7}$$

where $i = 1, \ldots, N_{\text{obs}}$.

Given a series of observations $Z = \left\{z(t_i; z_0^j, \alpha_k, \beta_k, \gamma_k) | i \in \{1, \ldots, N_{\text{obs}}\}, j \in \{1, \ldots, M\}, k \in \{1, \ldots, K\}\right\}$, we estimate the parameter $\boldsymbol{\theta}$ by minimizing the error between the observed trajectories Z and predicted trajectories $\hat{Z}$, denoted as

$$L^{\text{data}} = \sum_{i=1}^{N_{\text{obs}}} \sum_{j=1}^{M} \sum_{k=1}^{K} \|\hat{z}_{\boldsymbol{\theta}}(t_i; z_0^j, \alpha_k, \beta_k, \gamma_k)\}$$

Algorithm 1 Algorithm for Solving the Parameterized ODEs Problem

Input: N_{obs}: the number of observation time points; Integrator: ODE Solver; Z_0: initial state set; W: parameter set; $\boldsymbol{F}_{\boldsymbol{\theta}}(s, p, r, \alpha, \beta, \gamma)$: neural network; $\boldsymbol{\theta}_0$: initial parameters of neural network; Z: observed trajectories of initial state Z_0 and parameters W; n_{epochs}: the number of epoch; Δt: time step; L: loss function

Output: $\hat{Z}$: estimated trajectories of initial state Z_0 and parameters W; $\boldsymbol{F}_{\boldsymbol{\theta}^*}$: the trained neural network

```
θ = θ_0
for i = 1, ..., n_epochs do
    Ẑ = []
    for j = 1, ..., M do
        Select one element (s_0^j, p_0^j, r_0^j) from Z_0
        for k = 1, ..., K do
            Select one element (α_k, β_k, γ_k) from W
            for i = 1, ..., N_obs do
                (ds/dt, dp/dt, dr/dt) =
                  F_θ(s_{i-1}^j, p_{i-1}^j, r_{i-1}^j, α_k, β_k, γ_k)
                s_i^j = integrator(s_{i-1}^j, Δt, ds/dt)
                p_i^j = integrator(p_{i-1}^j, Δt, dp/dt)
                r_i^j = integrator(r_{i-1}^j, Δt, dr/dt)
                Add (s_i^j, p_i^j, r_i^j) to Ẑ.
            end
        end
    end
    Loss = L(Ẑ, Z)
    Update parameters θ based on Loss
end
```

$$- z(t_i; z_0^j, \alpha_k, \beta_k, \gamma_k)\}\|_2. \tag{8}$$

Inspired by [7], we also add the regularization term $L^{\text{S-Net}}$ to the loss function to avoid overfitting and enhance the generalization capability of the model. $L^{\text{S-Net}}$ is defined as (9),

$$L^{\text{S-Net}} = \sum_{p \in \theta_{L^{\text{S-Net}}}} l_1^s(p), \tag{9}$$

where $l_1^s(x) = \begin{cases} |x| - \dfrac{s}{2}, & \text{if } |x| > s \\ \dfrac{1}{2s}x^2, & \text{otherwise} \end{cases}$ and $s = 0.001$.

L^{data} and $L^{\text{S-Net}}$ constitute the loss function L, that is

$$L = L^{\text{data}} + L^{\text{S-Net}}. \tag{10}$$

Figure 1 provides a detailed illustration of how to generate a trajectory based on the initial state $z_0 = (s_0, p_0, r_0) \in Z_0$ and parameters $(\alpha, \beta, \gamma) \in W$. Herein, we denote $s_i = s(t_i; z_0, \alpha, \beta, \gamma)$, $p_i = p(t_i; z_0, \alpha, \beta, \gamma)$ and $r_i = r(t_i; z_0, \alpha, \beta, \gamma)$ to represent the states of s, p and r at time t_i. Algorithm 1 illustrates the pipeline for solving the parameterized ODE problem. We back-propagate the loss and use it to update the parameter $\boldsymbol{\theta}$, which allows us to obtain the best $\boldsymbol{\theta}^*$. This value represents the ODE system and enables us to predict the dynamics based on new initial states and parameters.

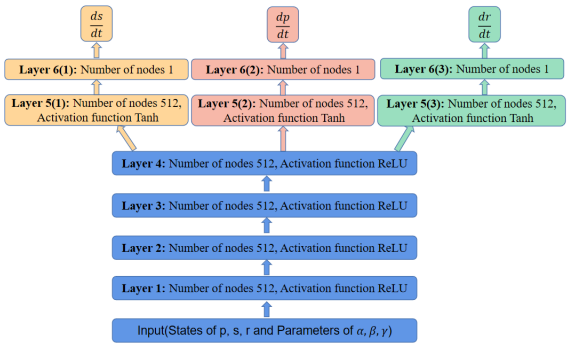


FIGURE 2. Schematic diagram of the O-Net. States of *s*, *p*, and *r* and parameters of α, β, γ are the input vector. The input vector goes through a four-layer network structure. Each layer has 512 units and a relu function. There are three parts for $\frac{ds}{dt}$, $\frac{dp}{dt}$ and $\frac{dr}{dt}$ respectively after four layers. Each part has two layers: the first layer has 512 units with the tanh activation function, and the second layer outputs the derivatives of $\frac{ds}{dt}$, $\frac{dp}{dt}$ and $\frac{dr}{dt}$.

Algorithm 2 O-Net

Input: $s, p, r, \alpha, \beta, \gamma \in \mathbb{R}$
Output: $\frac{ds}{dt}, \frac{dp}{dt}, \frac{dr}{dt}$

$y_1 = \text{relu}(\boldsymbol{w}_1^T(s, p, r, \alpha, \beta, \gamma)^T + \boldsymbol{b}_1)$,
$\boldsymbol{w}_1 \in \mathbb{R}^{6\times 512}, \boldsymbol{b}_1 \in \mathbb{R}^{512\times 1}$;
$y_2 = \text{relu}(\boldsymbol{w}_2^T y_1 + \boldsymbol{b}_2)$,
$\boldsymbol{w}_2 \in \mathbb{R}^{512\times 512}, \boldsymbol{b}_2 \in \mathbb{R}^{512\times 1}$;
$y_3 = \text{relu}(\boldsymbol{w}_3^T y_2 + \boldsymbol{b}_3)$,
$\boldsymbol{w}_3 \in \mathbb{R}^{512\times 512}, \boldsymbol{b}_3 \in \mathbb{R}^{512\times 1}$;
$y_4 = \text{relu}(\boldsymbol{w}_4^T y_3 + \boldsymbol{b}_4)$,
$\boldsymbol{w}_4 \in \mathbb{R}^{512\times 512}, \boldsymbol{b}_4 \in \mathbb{R}^{512\times 1}$;
$y_{51} = \tanh(\boldsymbol{w}_{51}^T y_4 + \boldsymbol{b}_{51})$,
$\boldsymbol{w}_{51} \in \mathbb{R}^{512\times 512}, \boldsymbol{b}_{51} \in \mathbb{R}^{512\times 1}$;
$y_{52} = \tanh(\boldsymbol{w}_{52}^T y_4 + \boldsymbol{b}_{52})$,
$\boldsymbol{w}_{52} \in \mathbb{R}^{512\times 512}, \boldsymbol{b}_{52} \in \mathbb{R}^{512\times 1}$;
$y_{53} = \tanh(\boldsymbol{w}_{53}^T y_4 + \boldsymbol{b}_{53})$,
$\boldsymbol{w}_{53} \in \mathbb{R}^{512\times 512}, \boldsymbol{b}_{53} \in \mathbb{R}^{512\times 1}$;
$\frac{ds}{dt} = \boldsymbol{w}_{61}^T y_{51} + \boldsymbol{b}_{61}, \boldsymbol{w}_{61} \in \mathbb{R}^{512\times 1}, \boldsymbol{b}_{61} \in \mathbb{R}$;
$\frac{dp}{dt} = \boldsymbol{w}_{62}^T y_{52} + \boldsymbol{b}_{62}, \boldsymbol{w}_{62} \in \mathbb{R}^{512\times 1}, \boldsymbol{b}_{62} \in \mathbb{R}$;
$\frac{dr}{dt} = \boldsymbol{w}_{63}^T y_{53} + \boldsymbol{b}_{63}, \boldsymbol{w}_{63} \in \mathbb{R}^{512\times 1}, \boldsymbol{b}_{63} \in \mathbb{R}$;

B. ARTIFICIAL NEURAL NETWORKS:O-NET AND S-NET

In Algorithm 1, we can use any form of the neural network, but we adopt the O-Net as our baseline, which is commonly used for regression tasks without exploring the underlying mechanism connecting inputs and outputs. However, to learn the combinations of different variables and parameters, we design S-Net, inspired by [6], [7], [8], and [9]. Unlike O-Net, S-Net first learns the analytic expression of ODEs and then predicts the dynamics. Notably, S-Net has no activation functions, and the most significant difference between O-Net and S-Net is the mapping between units in the layers.

1) O-NET

O-Net uses fully connected layers to learn the mapping from low to high-dimensional space and uses activation functions to learn the nonlinear relationship. Consider an O-Net with four shared layers and two unique layers for $\frac{ds}{dt}$, $\frac{dp}{dt}$ and $\frac{dr}{dt}$, as illustrated in Figure 2. To better understand O-Net, which is usually used for regression, we present a mathematical description in Algorithm 2 showing how O-Net is constructed.

O-Net is ineffective in solving variable parameters ODEs problems as it fails to capture the role of parameters. Although both parameters and variables are treated as input features, they behave differently. Specifically, at each time point, the input vector is $(z, \alpha, \beta, \gamma)$, where z changes over time while (α, β, γ) do not. As a result, it is challenging for α, β, γ to find appropriate weights in a fully connected neural network, making it difficult for O-Net to effectively learn the underlying dynamics of the system.

2) S-NET

S-Net possesses the ability to learn analytical expressions that can generalize to new domains effectively. The primary distinction between O-Net and S-Net lies in their layer-unit mapping, where S-Net is designed to capture the interaction between various variables and parameters efficiently. This is accomplished through two types of transformations: identity and linear combination maps, which enable S-Net to learn the underlying dynamics of the system accurately.

As an example, consider a two-layer S-Net that aims to learn the function $f \in \boldsymbol{F}$ in (1), as illustrated in Figure 3. With the appropriate number of layers, S-Net can represent all polynomials of the variables $(s, p, r, \alpha, \beta, \gamma)$. Herein, we only use addition and multiplication operators in S-Net. If necessary, we can add more operations to the S-Net to increase the capacity of the network. To better understand S-Net, we present an example in Algorithm 3 showing how S-Net is constructed. Particularly, we illustrate the learning process of $\alpha s - \beta r$ using S-Net in (11), (12), (13), (14) , and (15).

$$(\delta_1, \varepsilon_1)^T = \boldsymbol{w}_1 \times \left[s, p, r, \alpha, \beta, \gamma\right]^T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} s \\ p \\ r \\ \alpha \\ \beta \\ \gamma \end{bmatrix}, \tag{11}$$

$$f_1 = \delta_1 \varepsilon_1 = \alpha s, \tag{12}$$

$$(\delta_2, \varepsilon_2)^T = \boldsymbol{w}_2 \times \left[s, p, r, \alpha, \beta, \gamma, \alpha s\right]^T = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} s \\ p \\ r \\ \alpha \\ \beta \\ \gamma \\ \alpha s \end{bmatrix}, \tag{13}$$

$$f_2 = \delta_2 \varepsilon_2 = \beta r, \tag{14}$$

$$\boldsymbol{w}_3 \times \left[s, p, r, \alpha, \beta, \gamma, \alpha s, \beta r\right]^T$$

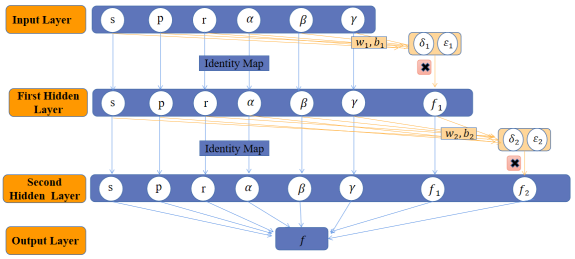


FIGURE 3. Schematic diagram of the S-Net. The identity map directly transfers $s, p, r, \alpha, \beta, \gamma$ from input layer to the first hidden layer. The linear combination chooses two elements remarked by δ_1 and ε_1 from the input vector using w_1 and b_1. The expressions $f_1 = \delta_1\varepsilon_1$ and $f_1 = \frac{\delta_1}{\varepsilon_1}$ correspond to the multiplication and division of two elements, respectively. We only implement the multiplication operation in this work. Apart from s, p, r, α, β and γ gotten by the identity map, f_1 will also be input to the second hidden layer. Similar to the first hidden layer, we get the further combination $f_2 = \delta_2\varepsilon_2$ by w_2 and b_2. Finally, we obtain the analytic expression of function f. It is necessary to enforce the sparsity of S-Net since it helps reduce overfitting and enables more robust predictions.

$$= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \times \begin{bmatrix} s \\ p \\ r \\ \alpha \\ \beta \\ \gamma \\ \alpha s \\ \beta r \end{bmatrix}$$

$$= \alpha s - \beta r. \tag{15}$$

Our analysis of $\alpha s - \beta r$ reveals that the sparsity of S-Net is a critical factor. Therefore, we introduced a regularization term into the loss function to promote model sparsity, as discussed in section IV-A.

V. NUMERICAL STUDIES: LOTKA-VOLTERRA EQUATION

The Lotka-Volterra model is frequently used to describe the dynamics of ecological systems where two species interact. [31] shows that cannibalism has both positive and negative effects on the stability of the Lotka-Volterra predator-prey model. It depends on the dynamic behaviors of the original system.

In this section, we consider Lotka-Volterra Equation (16), where different ecological systems have different parameters of α, β, δ and γ and initial states of x_0 and y_0,

$$\begin{cases} \dfrac{dx}{dt} = \alpha x - \beta xy, \\ \dfrac{dy}{dt} = -\delta y + \gamma xy. \end{cases} \tag{16}$$

The Euler integrator simulates ground truth trajectories in both training and testing stages with the time step $\Delta t = 0.1$, which empirically meets the stability requirements. The training and testing data consist of 150 and 33 trajectories, respectively, each of which starts from random initial state (x_0, y_0) in the interval $[0.6, 1.4]$ and parameters of $(\alpha, \beta, \delta, \gamma)$ in the interval $([1.0, 2.0], [0.5, 1.5], [2.5, 3.5], [0.5, 1.5])$ respectively. There is no overlap between the training and test data.

Algorithm 3 S-Net

Input: $s, p, r, \alpha, \beta, \gamma \in \mathbb{R}$
Output: F

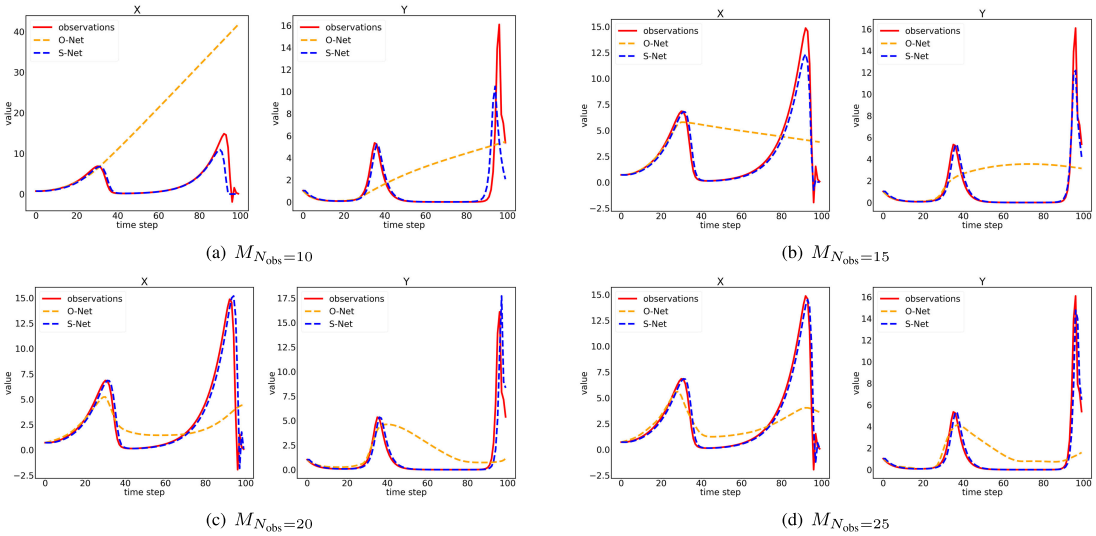
$(\delta_1, \varepsilon_1)^T = \boldsymbol{w}_1(s, p, r, \alpha, \beta, \gamma)^T + \boldsymbol{b}_1$,
$\boldsymbol{w}_1 \in \mathbb{R}^{2\times 6}, \boldsymbol{b}_1 \in \mathbb{R}^{2\times 1}$;
$f_1 = \delta_1\varepsilon_1$;
$(\delta_2, \varepsilon_2)^T = \boldsymbol{w}_2(s, p, r, \alpha, \beta, \gamma, f_1)^T + \boldsymbol{b}_2$,
$\boldsymbol{w}_2 \in \mathbb{R}^{2\times 7}, \boldsymbol{b}_2 \in \mathbb{R}^{2\times 1}$;
$f_2 = \delta_2\varepsilon_2$;
$F = \boldsymbol{w}_3(s, p, r, \alpha, \beta, \gamma, f_1, f_2)^T + b_3$,
$\boldsymbol{w}_3 \in \mathbb{R}^{1\times 8}, b_3 \in \mathbb{R}$;

The performance of O-Net and S-Net could get improved with a reasonable increase in the number of time points, i.e., N_{obs}, in the training data. We set four group experiments on training data with $N_{\text{obs}} = 10, 15, 20$, and 25. If the number of observation points is too large, it will also hurt the effectiveness of the model. Once there are too many observations, the accumulated error will be too large for one trajectory, which is not conducive to model training. We train S-Net with the L-BFGS optimizer [32] and use maximum iterations of 30000, a batch size of 150. We train O-Net with ADAM optimizer [33] for 5000 epochs using a batch size of 50, and a learning rate of 5e-3. We predict trajectories based on various initial states and parameters on $t \in (0, 10]$.

A. RESULTS AND DISCUSSIONS

Our study shows that S-Net is able to effectively recover the ODEs for the unknown variables of x and y, as well as the parameters α, β, δ, and γ. We use the notation $M_{N_{\text{obs}}}$ to represent the model trained on training data with N_{obs} time points. The results, summarized in Table 1, demonstrate that we are able to accurately recover the terms of (16). We observe that increasing the number of time points within a specific range results in higher model accuracy. Specifically, we find that when $M_{N_{\text{obs}}} = 25$, the coefficients of αx, βxy, δy, and γxy match those of the true equation. Additionally, we find that the terms not included in (16) have relatively small coefficients.

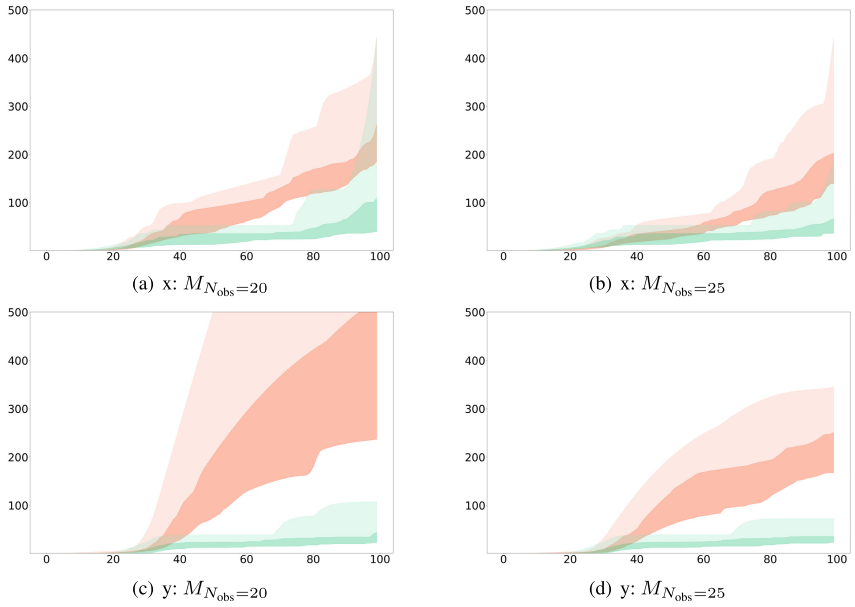
To evaluate the ability of the models to generate correct trajectories for new initial states and parameters, we feed 33 testing trajectories into the well-trained models and obtained predicted trajectories. Figure 4 shows the results of S-Net and O-Net with $N_{\text{obs}} = 10, 15, 20, 25$ for a specific test example $(x_0, y_0, \alpha, \beta, \delta, \gamma) = (0.71468, 1.01860, 1.10023, 1.17939, 2.78173, 1.18328)$. We find that S-Net can accurately predict the trajectory in the period of $t \in [0, 80\Delta t] = (0, 8]$, but its accuracy decreases when $t \in (8, 10]$. However, we observe that increasing the number of time points used in the training process improves the prediction of the S-Net of long-time dynamics. Specifically, S-Net trained on $M_{N_{\text{obs}}} = 25$ predicts with higher accuracy for $t \in (8, 10]$ than S-Net trained on $M_{N_{\text{obs}}} = 10$. On the other hand, O-Net performs poorly in all four cases, even though it performs well in the early stages



(a) $M_{N_{\text{obs}}=10}$ (b) $M_{N_{\text{obs}}=15}$

(c) $M_{N_{\text{obs}}=20}$ (d) $M_{N_{\text{obs}}=25}$

FIGURE 4. Lotka-Volterra: Testing results on $(x_0, y_0, \alpha, \beta, \delta, \gamma) = (0.71468, 1.01860, 1.10023, 1.17939, 2.78173, 1.18328)$ by S-Net and O-Net with time points 10, 15, 20 and 25 for the x-component (left) and y-component (right). In each plot, the horizontal axis indicates the time of prediction in the interval $(0, 100\Delta t] = (0, 10]$, and the vertical axis shows the values. The solid red line is the ground truth. The blue and orange dashed lines show the O-Net and S-Net results, respectively.



(a) x: $M_{N_{\text{obs}}=20}$ (b) x: $M_{N_{\text{obs}}=25}$

(c) y: $M_{N_{\text{obs}}=20}$ (d) y: $M_{N_{\text{obs}}=25}$

FIGURE 5. Lotka-Volterra: Prediction errors of the O-Net(orange) and S-Net(green) with different training time points 20 and 25. In each plot, the horizontal axis indicates the time of prediction in the interval $(0, 100\Delta t] = (0, 10]$, and the vertical axis shows the errors. The banded curves indicate the 25% – 100% percentile of the relative errors among 33 test samples. The dark regions indicate the 25% – 75% percentile of the relative error, which shows that S-Net performs significantly better than O-Net. The upper row is for variable x and the bottom low is for variable y.

for a short period, its predictions often deviate significantly from the actual trajectory. In some cases, O-Net even predicts trends that are opposite to the actual situation. Therefore, we conclude that S-Net has a stronger generalization ability than O-Net.

We define the error between observations Z and predictions $\hat{Z}$ as $\epsilon = \sqrt{||Z - \hat{Z}||^2}$. The error plots are shown in Figure 5 for two different time points, 20 and 25. In both cases, the error of O-Net is significantly larger than that of S-Net. For example, in Figure 5(c), we can observe that the error of O-Net sharply increases over time and soon reaches 500. In contrast, the error of S-Net increases at a relatively slower rate and reaches a maximum of approximately 100. Specifically, Table 2 shows the error of S-Net and O-Net for different models. The error of O-Net can be five or six times

TABLE 1. Lotka-Volterra identification with different time points.

True ODEs	$\frac{dx}{dt} = \alpha x - \beta xy,$ $\frac{dy}{dt} = -\delta y + \gamma xy,$
$M_{N_{\text{obs}}=10}$	$\frac{dx}{dt} = 0.996\alpha x - 0.988\beta xy + 0.00174x^2 - 0.00116\beta\delta y - 0.00105\beta x$ $\frac{dy}{dt} = -0.997\delta y + 0.987\gamma xy - 0.00101xy + 0.000967\gamma y + 0.000939x^2 y$
$M_{N_{\text{obs}}=15}$	$\frac{dx}{dt} = 0.998\alpha x - 0.989\beta xy - 0.00165\beta x^2 y - 0.00115\beta x - 0.000996xy$ $\frac{dy}{dt} = -0.997\delta y + 0.987\gamma xy - 0.00101xy + 0.000967\gamma y + 0.000939x^2 y$
$M_{N_{\text{obs}}=20}$	$\frac{dx}{dt} = 0.999\alpha x - 0.996\beta xy - 0.00110xy - 0.000948\alpha\beta xy - 0.000707\beta x$ $\frac{dy}{dt} = -0.998\delta y + 1.0\gamma xy - 0.000621\delta\gamma y + 0.000530y + 0.000442\alpha y$
$M_{N_{\text{obs}}=25}$	$\frac{dx}{dt} = 1.0\alpha x - 1.0\beta xy + 0.000764\alpha - 0.000683\beta y - 0.000652\beta x$ $\frac{dy}{dt} = -1.0\delta y + 1.0\gamma xy - 0.0377\gamma y + 0.0203y + 0.0112\gamma^2 y$

TABLE 2. MSE of S-Net and O-Net for equations of different models.

	Lotka–Volterra			Lorenz	
Model	O-Net	S-Net	Model	O-Net	S-Net
$M_{N_{\text{obs}}=10}$	302.70	58.99	$M_{N_{\text{obs}}=4}$	3119.18	415.95
$M_{N_{\text{obs}}=15}$	68.16	56.18	$M_{N_{\text{obs}}=6}$	402.89	8.87
$M_{N_{\text{obs}}=20}$	32.81	10.35	$M_{N_{\text{obs}}=10}$	417.71	2.29
$M_{N_{\text{obs}}=25}$	12.95	2.20	$M_{N_{\text{obs}}=20}$	1031.48	1.98

that of S-Net when $M_{N_{\text{obs}}}$ is 10 or 25, respectively. Clearly, S-Net performs significantly better than O-Net, suggesting that proper discretization is crucial when the ODE structure is unknown. Appendix C presents the findings of three other cases using different models obtained by $M_{N_{\text{obs}}} = 10$, 15, 20, and 25, respectively.

VI. NUMERICAL STUDIES: LORENZ EQUATION

In this section, we consider Lorenz Equations (17) where different ecological systems have different parameters σ, ρ and β and initial states x_0, y_0 and z_0.

$$\begin{cases} \frac{dx}{dt} = \sigma(y - x), \\ \frac{dy}{dt} = x(\rho - z) - y, \\ \frac{dz}{dt} = xy - \beta z. \end{cases} \tag{17}$$

Both the training and testing data consist of 150 and 60 trajectories, respectively, simulated by the Euler integrator with a time step of $\Delta t = 0.01$. The initial states and parameters are randomly selected from the intervals [0.8, 1.2], [0, 0.2], [0, 0.3], [5, 15], [23, 33], and [2, 3], respectively, without any overlap between the two sets. We conduct four group experiments on the training data with different numbers of time points: 4, 6, 10, and 20. For training S-Net, we use the

TABLE 3. Lorenz identification with different models.

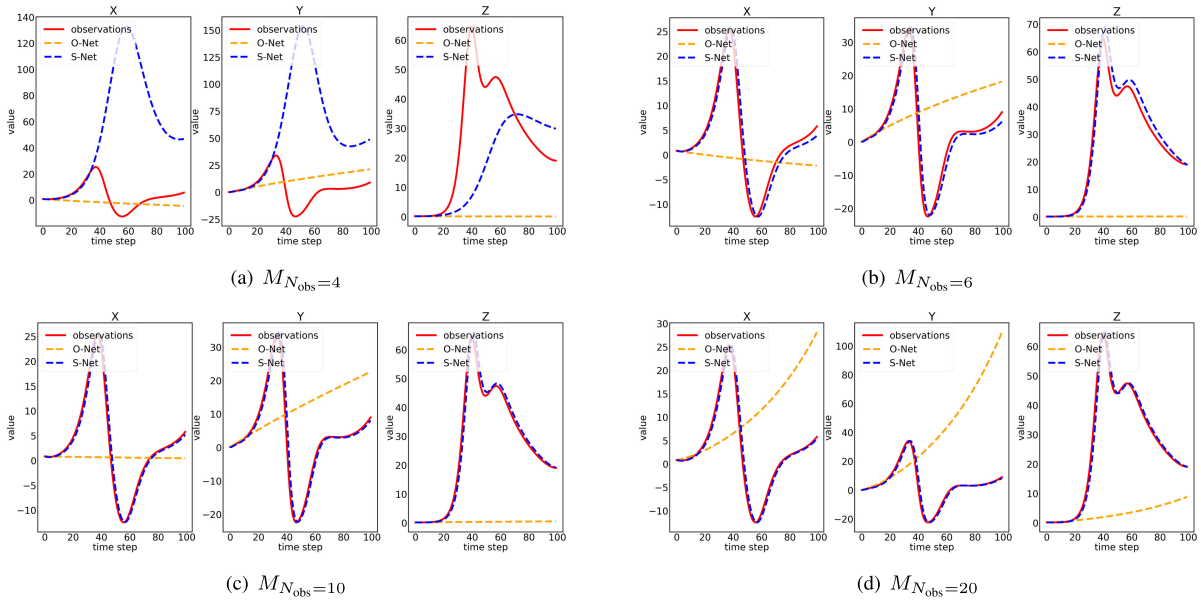
True ODEs	$\frac{dx}{dt} = \sigma y - \sigma x,$ $\frac{dy}{dt} = x\rho - xz - y,$ $\frac{dz}{dt} = xy - \beta z,$
$M_{N_{\text{obs}}=4}$	$\frac{dx}{dt} = 0.999\sigma y - 0.999\sigma x - 0.003\sigma + 0.000403\sigma z - 0.0004\beta$ $\frac{dy}{dt} = 1.0x\rho - 0.845xz - 0.579y - 0.0928xy - 0.0119\rho y$ $\frac{dz}{dt} = -2.5z + 0.595y + 0.0614x - 0.0126\beta + 0.0122\sigma$
$M_{N_{\text{obs}}=6}$	$\frac{dx}{dt} = 0.999\sigma y - 0.999\sigma x - 0.00183\sigma - 0.000529\beta + 0.000296\sigma z$ $\frac{dy}{dt} = 1.0x\rho - 0.929xz - 0.904y - 0.0183xy - 0.00282\rho y$ $\frac{dz}{dt} = -0.966\beta z + 0.365\beta xy + 0.358\beta y + 0.172\beta x - 0.130\beta^2 y$
$M_{N_{\text{obs}}=10}$	$\frac{dx}{dt} = 0.997\sigma y - 0.996\sigma x - 0.00158\sigma - 0.000392\sigma x^2 - 0.000362x$ $\frac{dy}{dt} = 1.0x\rho - 0.979xz - 0.980y - 0.00403xy - 0.00124\beta x$ $\frac{dz}{dt} = -0.993\beta z + 0.988xy + 0.00173\beta xy + 0.00101y + 0.000909y^2$
$M_{N_{\text{obs}}=20}$	$\frac{dx}{dt} = 1.0\sigma y - 1.0\sigma x - 0.000905y - 0.000392 - 0.000282xy$ $\frac{dy}{dt} = 1.0x\rho - 0.998xz - 0.997y - 0.000986x - 0.000708xy$ $\frac{dz}{dt} = 0.999xy - 0.997\beta z + 0.000818x^2 + 0.000562yz - 0.000405z$

L-BFGS optimizer with a maximum of 50,000 iterations and a batch size of 150. For training O-Net, we use the ADAM optimizer with a learning rate of 5e-3, a batch size of 100, and 5,000 epochs. We predict trajectories for various initial states and parameters on the time interval $t \in (0, 1]$.

A. RESULTS AND DISCUSSIONS

Table 3 presents the capability of the trained S-Net to identify the underlying ODE model, showing the top five terms of coefficient weights recovered by S-Net with certain accuracy. Notably, when $M_{N_{\text{obs}}} = 20$, the coefficients of σy, σx, and $x\rho$ match those of the true equation with high precision. The coefficients of the remaining four terms, namely xz, y, xy, and βz, deviate only slightly from the true values, with a maximum difference of 0.01. These results demonstrate the effectiveness of S-Net in recovering the underlying ODE model.

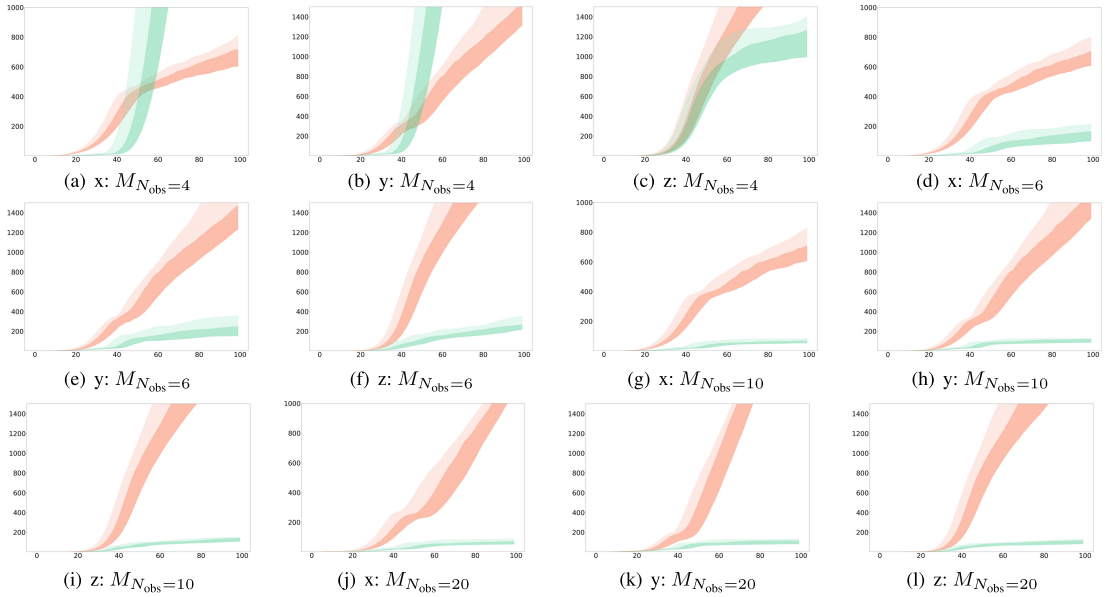
We evaluate the predictive performance of S-Net and O-Net on (17) using 60 testing trajectories and obtain corresponding predictions from the trained models. Figure 6 shows the results of S-Net and O-Net with $N_{\text{obs}} = 4, 6, 10, 20$ for a specific test example $(x_0, y_0, z_0, \sigma, \rho, \beta) = (0.82841, 0.14785, 0.17099, 12.39551, 32.03720, 2.67205)$. We predict the trajectories of x, y and z on $t \in (0, 100\Delta t] = (0, 1]$ using the learned models. When $M_{N_{\text{obs}}} = 4$, both O-Net and S-Net predictions are very poor. As the number



(a) $M_{N_{\text{obs}}=4}$ (b) $M_{N_{\text{obs}}=6}$

(c) $M_{N_{\text{obs}}=10}$ (d) $M_{N_{\text{obs}}=20}$

FIGURE 6. Lorenz: Testing results on $(x_0, y_0, z_0, \sigma, \rho, \beta) = (0.82841, 0.14785, 0.17099, 12.39551, 32.03720, 2.67205)$ by S-Net and O-Net with time points 4, 6, 10 and 20 for the x-component (left), y-component (middle) and z-component (right). In each plot, the horizontal axis indicates the time of prediction in the interval $(0, 100\Delta t] = (0, 1]$, and the vertical axis shows the values. The solid red line is the ground truth. The blue and orange dashed lines show the O-Net and S-Net results, respectively.



(a) x: $M_{N_{\text{obs}}=4}$ (b) y: $M_{N_{\text{obs}}=4}$ (c) z: $M_{N_{\text{obs}}=4}$ (d) x: $M_{N_{\text{obs}}=6}$

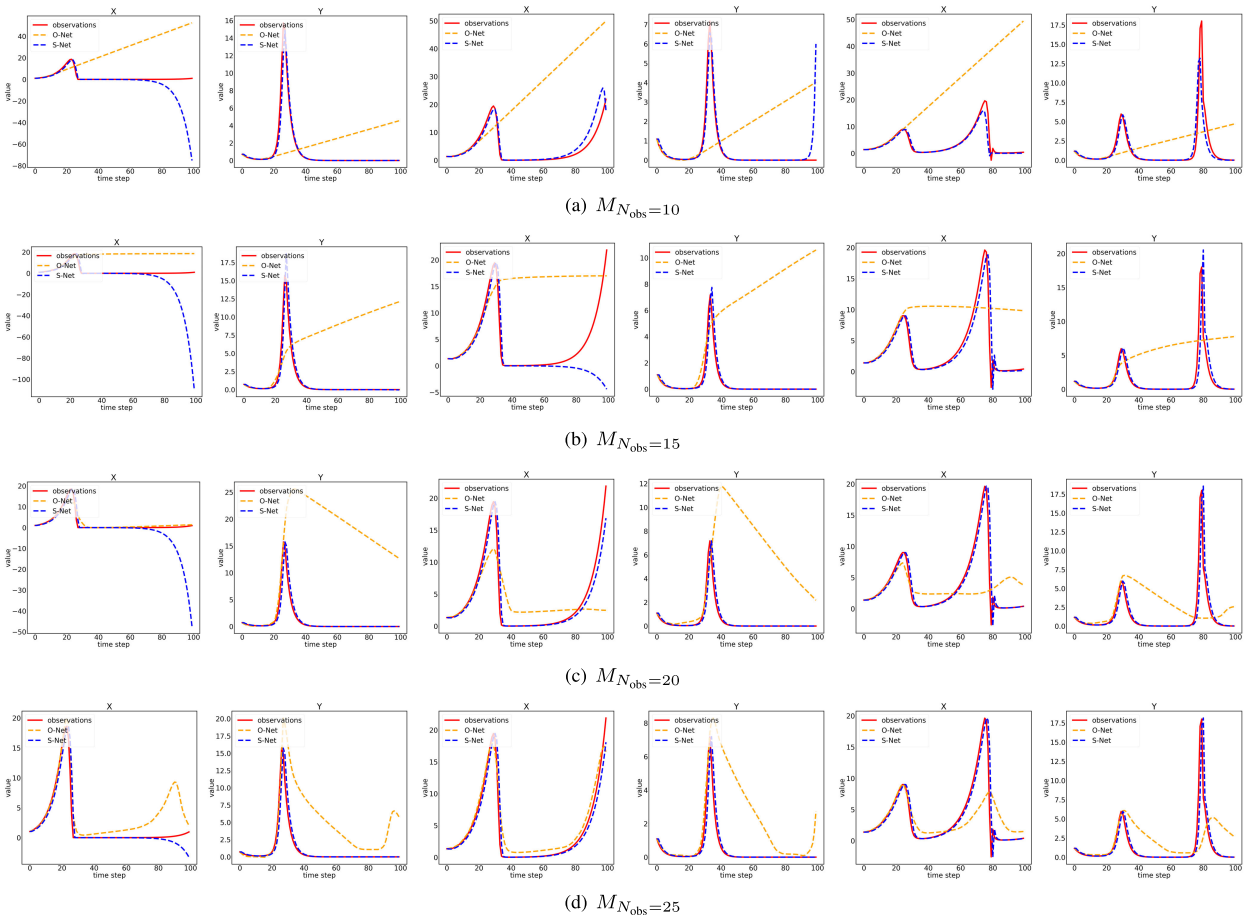
(e) y: $M_{N_{\text{obs}}=6}$ (f) z: $M_{N_{\text{obs}}=6}$ (g) x: $M_{N_{\text{obs}}=10}$ (h) y: $M_{N_{\text{obs}}=10}$

(i) z: $M_{N_{\text{obs}}=10}$ (j) x: $M_{N_{\text{obs}}=20}$ (k) y: $M_{N_{\text{obs}}=20}$ (l) z: $M_{N_{\text{obs}}=20}$

FIGURE 7. Lorenz: Prediction errors of the O-Net(orange) and S-Net(green) with different training time points 4, 6, 10, and 20. In each plot, the horizontal axis indicates the time of prediction in the interval $(0, 100\Delta t] = (0, 1]$, and the vertical axis shows the errors. The banded curves indicate the 25% – 100% percentile of the relative errors among 60 test samples and the dark regions indicate the 25% – 75% percentile of the relative error.

of time points increased, the performance of S-Net improve significantly, but the performance of O-Net remain almost unchanged, as shown in Figures 6(b), 6(c), and 6(d). Even in the initial stages of each subfigure, O-Net was unable to perform well.

The error plots for four different time points (4, 6, 10, and 20) are presented in Figure 7. In each case, O-Net exhibits significantly larger errors than S-Net. For instance, consider Figure 7(l). The error of O-Net increases sharply with time and soon reaches 1400, while the error of S-Net increases relatively slowly with time and reaches a maximum of about 100. Table 2 shows the errors of S-Net and O-Net of various time points. The error of O-Net can be up to a thousand times larger than that of S-Net. Clearly, S-Net outperforms



(a) $M_{N_{\text{obs}}=10}$

(b) $M_{N_{\text{obs}}=15}$

(c) $M_{N_{\text{obs}}=20}$

(d) $M_{N_{\text{obs}}=25}$

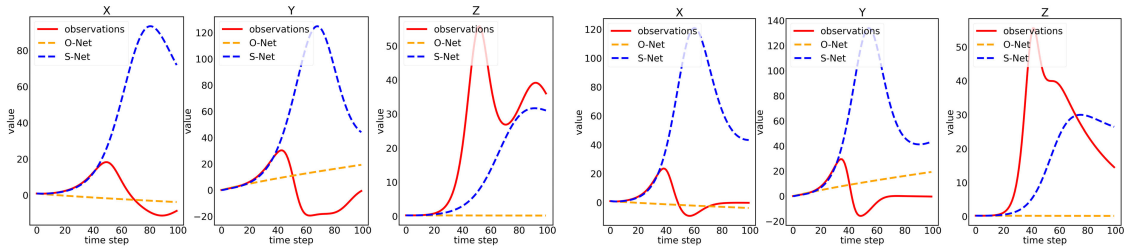
FIGURE 8. **Testing results on $(x_0, y_0, \alpha, \beta, \delta, \gamma) \in \{(1.03591, 0.71055, 1.64399, 0.73789, 2.63207, 0.58110), (1.34048, 1.08388, 1.29828, 1.44437, 3.24866, 0.58960), (1.39070, 1.15958, 1.20000, 1.01820, 3.32212, 0.93040)\}$ by S-Net and O-Net with time points 10, 15, 20 and 25 for the *x*-component (left) and *y*-component (right).**

O-Net significantly. We provide the findings of two other cases using different models obtained by $N_{\text{obs}} = 4$, 6, 10, and 20, respectively, in Appendix D.

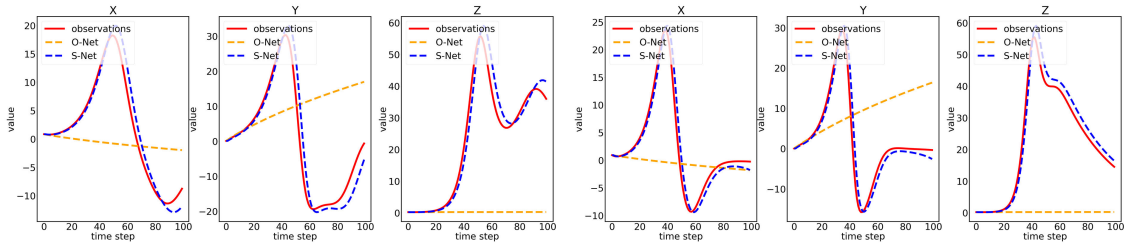
VII. CONCLUSION

Exploring parameterized ODEs is a broad area in computational science and engineering. Parameterized ODEs problems are particularly challenging to solve because the solutions to these problems can vary with the parameters used. Black-box methods offer some insights, but their generalization performance is often subpar due to a limited understanding of underlying mechanisms. To address this issue, we propose a novel framework that combines the S-Net with an ODE Solver to learn parameterized ODEs. Unlike the black-box O-Net method that simply fits the observed data, S-Net learns the expressions of the equations based on the observed data. We use the Euler method as the ODE Solver due to its simplicity. Experiments show the S-Net framework surpasses the O-Net method in learning parameterized ODEs, evidenced by superior efficiency and accuracy in tests involving the Lotka-Volterra and Lorenz Equations. The proposed framework marks a significant advancement in studying parameterized ODEs in computational science and engineering.

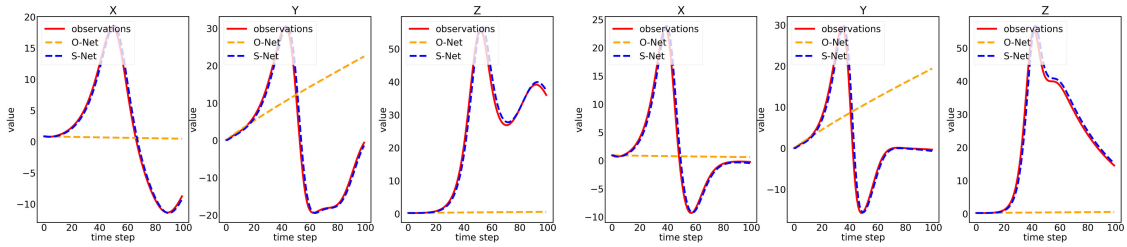
However, there are several limitations to address in future research. Firstly, the current version of S-Net only supports basic operators such as addition, subtraction, and multiplication, which limits its application to complex systems. To broaden its applicability, it is necessary to incorporate more advanced operators such as division, trigonometric functions, powers, and fractional calculation operators. Second, enhancing simulation accuracy requires implementing other ODE Solvers. Thirdly, the present approach is not applicable to arbitrary order systems, which present a more complex challenge. The combinatorial relationship between variables and parameters must be known, and the observations are supplied based on the information on variables. An additional step may be required to predict the number of variables or parameters based on the current method. Finally, while S-Net has proven resilient against noisy data in [9], testing with real data is necessary to establish its applicability to diverse domains.



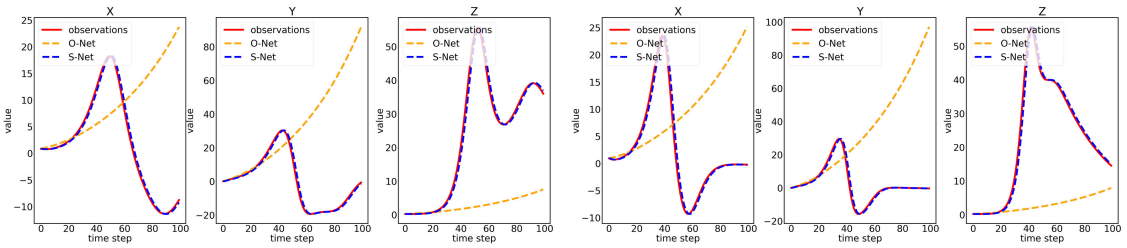
(a) $M_{N_{\text{obs}}=4}$



(b) $M_{N_{\text{obs}}=6}$



(c) $M_{N_{\text{obs}}=10}$



(d) $M_{N_{\text{obs}}=20}$

FIGURE 9. Testing results on $(x_0, y_0, z_0, \sigma, \rho, \beta) \in \{(0.84731, 0.11332, 0.24186, 5.24679, 29.69917, 2.26212), (0.92617, 0.04462, 0.21900, 14.47371, 27.97391, 2.57754)\}$ by S-Net and O-Net with time points 4, 6, 10 and 20 for the x-component (left), y-component (middle) and z-component (right).

APPENDIX A
CONVERGENCE OF THE EULER METHOD

Our subsequent analysis establishes the convergence of the Euler method when solving the initial-value problem of a first-order differential equation, as stated in (18)

$$\begin{cases} \dfrac{dy}{dx} = f(x, y), x > x_0, \\ y(x_0) = y_0. \end{cases} \tag{18}$$

Here, the unknown function is denoted by $y(x)$, the known function by $f(x, y)$, and the initial data by y_0.

Theorem 1: Assume the following one-step method corresponding to the initial value problem (18) be the p-order accuracy

$$y_{n+1} = y_n + h\phi(x_n, y_n, h)$$

and the function ϕ satisfies the Lipschitz condition for y, i.e., $\exists L > 0$,

$$|\phi(x, y_1, h) - \phi(x, y_2, h)| \leq L|y_1 - y_2|, \quad \forall y_1, y_2$$

and $y_0 = y(x_0)$, then the one-step method is convergent and $y(x_n) - y_n = O(h^p)$.

Proof: Let $e_n = y(x_n) - y_n$, then

$$y(x_{n+1}) = y(x_n) + h\phi(x_n, y(x_n), h) + T_{n+1},$$

i.e.,

$$e_{n+1} = e_n + h[\phi(x_n, y(x_n), h) - \phi(x_n, y_n, h)] + T_{n+1}.$$

Since the one-step method is the p-order accuracy, then $\exists h_0, 0 < h \leq h_0$, satisfy $|T_{n+1}| \leq Ch^{p+1}$ where C is a constant. That is

$$|e_{n+1}| \leq |e_n| + hL|e_n| + Ch^{p+1} = \alpha|e_n| + \beta,$$

where $\alpha = 1 + hL$, $\beta = Ch^{p+1}$. So we can get

$$\begin{aligned} |e_n| &\leq \alpha|e_{n-1}| + \beta \leq \alpha^2|e_{n-2}| + \alpha\beta + \beta \\ &\leq \alpha^3|e_{n-3}| + \beta(1 + \alpha + \alpha^2) \leq \ldots \leq . \end{aligned} \tag{19}$$

From the known conditions, we get

$$\begin{aligned} |e_n| &\leq \exp(L(x_n - x_0))|e_0| + Ch^pL^{-1}(\exp(L(x_n - x_0)) - 1) \\ &= Ch^pL^{-1}(\exp(L(x_n - x_0)) - 1). \end{aligned} \tag{20}$$

From (20), we know when $h \to 0$, then $|e_n| \to 0$. ■

APPENDIX B
THE SEMI-IMPLICIT SOLVERS

Our approach can be adapted to learn unknown functions utilizing semi-implicit solvers with suitable modifications. For instance, consider the semi-implicit Euler method, which can be employed for a pair of differential equations with the form

$$\begin{aligned} \frac{dx}{dt} &= f(t, y), \\ \frac{dy}{dt} &= g(t, x), \end{aligned} \tag{21}$$

where f and g are unknown functions that we want to learn. The semi-implicit Euler method produces an approximateÂ discreteÂ solution by iterating

$$\begin{aligned} y_{n+1} &= y_n + g(t_n, x_n)\Delta t, \\ x_{n+1} &= x_n + f(t_n, y_{n+1})\Delta t, \end{aligned} \tag{22}$$

where Δt is the time step. The difference with the standard Euler method is that the semi-implicit Euler method uses y_{n+1} in the equation for x_{n+1}, while the standard Euler method uses y_n. Given that the expressions for f and g are represented by S-Nets, S-Net$_f$ and S-Net$_g$, it is necessary to provide them with known information. In the semi-implicit Euler method, we use a positive time step to compute x_{n+1} from y_{n+1} generated by S-Net$_g$ based on starting points x_0, y_0, that is

$$x_{n+1} = x_n + \text{S-Net}_f(t_n, \text{S-Net}_g(t_n, x_n)\Delta t). \tag{23}$$

In this context, y is not utilized as an observable variable; instead, it serves as a component to generate the observable variable x. With this approach, as the expression for g must be learned through the x variable, we require additional observations on the x variable and no longer need observations on the y variable.

APPENDIX C
LOTKA-VOLTERRA

In Figure 8, we present the outcomes of three distinct models obtained by utilizing $N_{\text{obs}} = 10, 15, 20$, and 25, respectively. The rows indicate the prediction results of the different models for three test samples. The first two columns in each row demonstrate the evolution of the two variables x and y for the first test sample $(x_0, y_0, \alpha, \beta, \delta, \gamma) = (1.03591, 0.71055, 1.64399, 0.73789, 2.63207, 0.58110)$. The center two columns illustrate the changes in the variables x and y over time for the second test sample $(x_0, y_0, \alpha, \beta, \delta, \gamma) = (1.34048, 1.08388, 1.29828, 1.44437, 3.24866, 0.58960)$. Finally, the last two columns show how the variables x and y varied over time for the third test sample $(x_0, y_0, \alpha, \beta, \delta, \gamma) = (1.39070, 1.15958, 1.20000, 1.01820, 3.32212, 0.93040)$. Each subplot displays the time of prediction on the horizontal axis within the interval $(0, 100\Delta t] = (0, 10]$, while the vertical axis indicates the corresponding values. The solid red line represents the ground truth, while the orange and blue dashed lines show the O-Net and S-Net prediction results, respectively. Notably, within a specific range, the prediction accuracy of the model increases with a higher number of observation points used to train the model.

APPENDIX D
LORENZ

In Figure 9, we present the outcomes of two examples utilizing distinct models derived from $N_{\text{obs}} = 4, 6, 10$, and 20, respectively. Each row represents the prediction results of different models on two test samples. The first three columns of each row demonstrate the progression of the three variables x, y, and z for the first test sample $(x_0, y_0, z_0, \sigma, \rho, \beta) = (0.84731, 0.11332, 0.24186, 5.24679, 29.69917, 2.26212)$, while the last three columns depict how the three variables x, y, and z evolve over time for the third test sample $(x_0, y_0, z_0, \sigma, \rho, \beta) = (0.92617, 0.04462, 0.21900, 14.47371, 27.97391, 2.57754)$. Each plot shows the prediction time in the interval $(0, 100\Delta t] = (0, 1]$ on the horizontal axis and the corresponding values on the vertical axis. The solid red line in each subplot represents the actual ground truth, while the orange and blue dashed lines show the O-Net and S-Net prediction results, respectively. Notably, the prediction accuracy of the model improves with an increase in the number of observation points used to train the model.

REFERENCES

[1] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, Dec. 2018, pp. 6572–6583.
[2] Y. Rubanova, R. T. Chen, and D. K. Duvenaud, "Latent ordinary differential equations for irregularly-sampled time series," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, Dec. 2019, pp. 5321–5331.
[3] C. Herrera, F. Krach, and J. Teichmann, "Neural jump ordinary differential equations: Consistent continuous-time prediction and filtering," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Austria, May 2021, pp. 1–44. [Online]. Available: https://openreview.net/forum?id=JFKR3WqwyXR
[4] Z. Chen, J. Zhang, M. Arjovsky, and L. Bottou, "Symplectic recurrent neural networks," in *Proc. 8th Int. Conf. Learn. Represent. (ICLR)*, Addis Ababa, Ethiopia, Apr. 2020, pp. 1–23. [Online]. Available: https://openreview.net/forum?id=BkgYPREtPr

[5] 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.*, vol. 378, pp. 686–707, Feb. 2019.
[6] Z. Long, Y. Lu, X. Ma, and B. Dong, "PDE-Net: Learning PDEs from data," in *Proc. Mach. Learn. Res.*, Jul. 2018, pp. 3208–3216.
[7] 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.*, vol. 399, Dec. 2019, Art. no. 108925.
[8] S. Sahoo, C. Lampert, and G. Martius, "Learning equations for extrapolation and control," in *Int. Conf. Mach. Learn.*, Jul. 2018, pp. 4442–4450.
[9] Q. Li and S. Evje, "Learning the nonlinear flux function of a hidden scalar conservation law from data," *Netw. Heterogeneous Media*, vol. 18, no. 1, pp. 48–79, 2022.
[10] R. Maulik, K. Fukami, N. Ramachandra, K. Fukagata, and K. Taira, "Probabilistic neural networks for fluid flow surrogate modeling and data recovery," *Phys. Rev. Fluids*, vol. 5, no. 10, Oct. 2020, Art. no. 104401.
[11] K. Lee and E. J. Parish, "Parameterized neural ordinary differential equations: Applications to computational physics problems," *Proc. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 477, no. 2253, Sep. 2021, Art. no. 20210162.
[12] Y. S. Shimizu and E. J. Parish, "Windowed space–time least-squares Petrov–Galerkin model order reduction for nonlinear dynamical systems," *Comput. Methods Appl. Mech. Eng.*, vol. 386, Dec. 2021, Art. no. 114050.
[13] K. Lee and N. Trask, "Parameter-varying neural ordinary differential equations with partition-of-unity networks," 2022, *arXiv:2210.00368*.
[14] H. Owhadi, "Bayesian numerical homogenization," *Multiscale Model. Simul.*, vol. 13, no. 3, pp. 812–828, Jan. 2015.
[15] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Inferring solutions of differential equations using noisy multi-fidelity data," *J. Comput. Phys.*, vol. 335, pp. 736–746, Apr. 2017.
[16] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Machine learning of linear differential equations using Gaussian processes," *J. Comput. Phys.*, vol. 348, pp. 683–693, Nov. 2017.
[17] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proc. Nat. Acad. Sci. USA*, vol. 113, no. 15, pp. 3932–3937, Mar. 2016.
[18] H. Schaeffer, "Learning partial differential equations via data discovery and sparse optimization," *Proc. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 473, no. 2197, Jan. 2017, Art. no. 20160446.
[19] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Data-driven discovery of partial differential equations," *Sci. Adv.*, vol. 3, no. 4, Apr. 2017, Art. no. e1602614.
[20] Z. Wu and R. Zhang, "Learning physics by data for the motion of a sphere falling in a non-Newtonian fluid," *Commun. Nonlinear Sci. Numer. Simul.*, vol. 67, pp. 577–593, Feb. 2019.
[21] V. Iakovlev, M. Heinonen, and H. Lähdesmäki, "Learning continuous-time PDEs from sparse data with graph neural networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Austria, May 2021, pp. 1–15.
[22] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia, "Learning mesh-based simulation with graph networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Austria, May 2021, pp. 1–18.
[23] B. Chang, M. Chen, E. Haber, and E. H. Chi, "AntisymmetricRNN: A dynamical system view on recurrent neural networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, New Orleans, LA, USA, May 2019, pp. 1–15.
[24] E. D. Brouwer, J. Simm, A. Arany, and Y. Moreau, "GRU-ODE-Bayes: Continuous modeling of sporadically-observed time series," in *Proc. Adv. Neural Inf. Process. Syst.*, Vancouver, BC, Canada, Dec. 2019, pp. 7377–7388.
[25] A. Yazdani, L. Lu, M. Raissi, and G. E. Karniadakis, "Systems biology informed deep learning for inferring parameters and hidden dynamics," *PLOS Comput. Biol.*, vol. 16, no. 11, Nov. 2020, Art. no. e1007575.
[26] C. K. Buhler, R. S. Terry, K. G. Link, and F. R. Adler, "Do mechanisms matter? Comparing cancer treatment strategies across mathematical models and outcome objectives," *Math. Biosci. Eng.*, vol. 18, no. 5, pp. 6305–6327, 2021.
[27] K. Ohta and H. Ishida, "Comparison among several numerical integration methods for Kramers–Kronig transformation," *Appl. Spectrosc.*, vol. 42, no. 6, pp. 952–957, Aug. 1988.
[28] G. R. W. Quispel and D. I. McLaren, "A new class of energy-preserving numerical integration methods," *J. Phys. A, Math. Theor.*, vol. 41, no. 4, Feb. 2008, Art. no. 045206.
[29] K. Atkinson, *An Introduction to Numerical Analysis*. Hoboken, NJ, USA: Wiley, 2008.
[30] J. D. Lambert, *Numerical Methods for Ordinary Differential Systems*, vol. 146. New York, NY, USA: Wiley, 1991.
[31] H. Deng, F. Chen, Z. Zhu, and Z. Li, "Dynamic behaviors of Lotka–Volterra predator–prey model incorporating predator cannibalism," *Adv. Difference Equ.*, vol. 2019, no. 1, pp. 1–17, Dec. 2019.
[32] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Math. Program.*, vol. 45, nos. 1–3, pp. 503–528, Aug. 1989.
[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

QING LI received the B.S. degree in applied mathematics from Qingdao University, China, in 2015, with a focus on solid foundation in mathematical theory and its practical applications, and the M.S. degree in mathematics from the South China University of Technology, China, with a focus on advanced mathematical concepts and numerical methods. She is currently pursuing the Ph.D. degree with the Department of Energy and Petroleum Technology, University of Stavanger, Norway, with a focus on developing novel machine learning algorithms that leverage her expertise in differential equations, both ordinary and partial, and numerical calculation. Her research interests include the intersection of these fields, with a particular emphasis on developing innovative solutions to complex problems in energy and petroleum technology.

STEINAR EVJE received the M.S. and Ph.D. degrees in applied mathematics from the University of Bergen, in 1992 and 1998, respectively. He is currently a Professor in applied and computational mathematics with the Department of Energy and Petroleum Engineering, University of Stavanger. With more than two decades of experience in the field, he has developed a keen interest in the development of mathematical models that can be used to gain insight into fundamental mechanisms for various multiphase transport and reaction processes within fluid mechanics and medical engineering. In addition to his work on mathematical modeling, he has also focused on leveraging the power of data-driven modeling, including machine learning methods to solve problems related to partial differential equations. He has published numerous papers and articles that have helped advance the state of the art in both machine learning and mathematical modeling.

JIAHUI GENG (Graduate Student Member, IEEE) received the B.S. degree from the School of Automation, Southeast University, China, in 2015, and the M.S. degree from the Department of Computer Science, RWTH Aachen University, Germany, in 2018. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Computer Engineering, University of Stavanger, Norway. His research interests include robustness, privacy, and security, as well as the development of blockchain systems and the application of dynamic systems in machine learning. With a passion for exploring the cutting edge of his field, he is committed to making meaningful contributions to the development of robust, secure, and privacy-preserving machine learning systems that can be used to drive innovation and progress in a wide range of industries and applications.

Paper II:
Solving Nonlinear Conservation Laws of Partial Differential Equations Using Graph Neural Networks

Qing Li, Jiahui Geng, Steinar Evje, Chunming Rong

Proceedings of the Northern Lights Deep Learning Workshop 4 (2023)
doi = https://doi.org/10.7557/18.6808

Solving Nonlinear Conservation Laws of Partial Differential Equations Using Graph Neural Networks

Qing Li §*[1], Jiahui Geng §[2], Steinar Evje[1], and Chunming Rong[2]

[1]University of Stavanger, Department of Energy and Petroleum, Group of Computational Engineering, Stavanger, Norway
[2]University of Stavanger, Department of Electrical Engineering and Computer Science, Stavanger, Norway

Abstract

Nonlinear Conservation Laws of Partial Differential Equations (PDEs) are widely used in different domains. Solving these types of equations is a significant and challenging task. Graph Neural Networks (GNNs) have recently been established as fast and accurate alternatives for principled solvers when applied to standard equations with regular solutions. There have been few investigations on GNNs implemented for complex PDEs with nonlinear conservation laws. Herein, we explore GNNs to solve the following problem

$$u_t + f(u,\beta)_x = 0 \qquad (*)$$

where $f(u,\beta)$ is the nonlinear flux function of the scalar conservation law, u is the main variable, and β is the physical parameter. The main challenge of nonlinear conservation laws is that solutions typically create shocks. That is, one or several jumps in the form (u_L, u_R) with $u_L \neq u_R$ moving in space and probably changing over time such that information about $f(u)$ in the interval associated with this jump is not present in the observation data. We demonstrate that GNNs could achieve accurate estimates of PDEs solutions based on new initial conditions and physical parameters within a specific parameter range.

*Corresponding Author: qing.li@uis.no
§The authors contributed equally to this work.

1 Introduction

Machine learning methods have been widely used to solve PDEs in science and engineering, for example, aerodynamics [16, 3], electromagnetism [13], geophysics [17] and weather prediction [1], etc. According to [18, 7, 14, 10], GNNs have recently been introduced and made much progress in this area, offering faster runtimes than principled solvers. Compared to grid-based convolutional neural networks(CNNs) [20, 21], GNNs demonstrated better adaptivity in the simulation scenarios [22, 2, 5]. However, most currently published papers use GNNs to resolve PDEs with regular solutions, such as the Wave equation, the Poisson's equation, and the Navier-Stokes equations. These tests have demonstrated that GNNs can resolve these partial differential equations with high efficiency and precision. How would GNNs perform if we try to use it in the context of nonlinear conservation laws?

In this paper, we implement GNN variants to solve PDEs with nonlinear flux function $f(u,\beta)$ that is involved in general scalar nonlinear conservation laws. We restrict to the one-dimensional case given by

$$u_t + f(u,\beta)_x = 0 \qquad (1)$$

where $u = u(x,t)$ is the main variable and β is a parameter of a physical phenomenon. We train GNNs based on observation data $u(x_j, t_i, \beta_k)$ on a spatial grid $x_j,\ j = 1,\ldots,N_x$ at specified times $t_i,\ i = 1,\ldots,N_{obs}$ with some specific parameters $\beta_k,\ k = 1,\ldots,N_\beta$. Using learned GNNs, we pre-

https://doi.org/10.7557/18.6808

dict the solutions of Eq. (1) given new values of parameter β and initial state u_0.

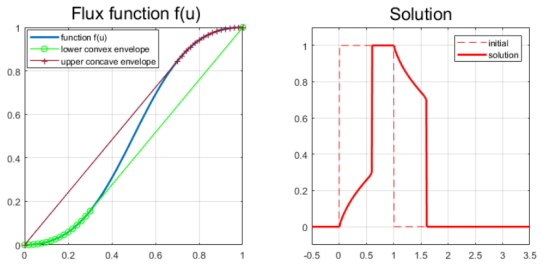


Figure 1: Left: Example of nonlinear flux function $f(u) = \frac{u^2}{u^2+(1-u)^2}$ (blue curve). Upper concave envelope (brown curve) and lower convex envelope (green curve) are also included. Right: The solution of Eq. (1) at time $T = 0.5$ is shown (red solid curve) together with its initial data $u_0(x)$ (red dashed line).

Compared to equations with regular solutions, Eq. (1) has some characteristics that make it more challenging to solve. Typically, Eq. (1) generates shock wave solutions $u(x,t)$ in finite time, i.e., solutions that contain one or more discontinuities expressed as a jump (u_L, u_R) with $u_L \neq u_R$, though the initial value $u_0(x)$ is smooth [8, 6]. In particular, the specific form of $f(u)$ in the interval $[\min(u_L, u_R), \max(u_L, u_R)]$ is not used in the construction of the entropy solution, only the slope $s = \frac{f(u_L)-f(u_R)}{u_L-u_R}$. As jumps arise and disappear in the solution over the period for which observation data is collected, the data may lack information about $f(u)$. This situation is illustrated in Fig. 1. In the left panel, we plot the flux function $f(u) = u^2/(u^2 + (1-u)^2)$. In the right panel, the entropy solution is shown after a time $T = 0.5$. At the time $t = 0$, the initial data $u_0(x)$ involves one jump at $x = 0$ and another at $x = 1$. The initial jump at $x = 0$ is instantly transformed into a solution that is a combination of a continuous wave solution (rarefaction wave) and a discontinuous wave $(u_L, u_R) \approx (0.3, 1.0)$, as dictated by the lower convex envelope shown in the left panel (green curve) [8]. Similarly, the initial jump at $x = 1$ is transformed into a solution that is a combination of a continuous wave solution (rarefaction wave) and a discontinuous wave $(u_L, u_R) \approx (0.7, 0)$, by the upper concave envelope illustrated in the left panel (brown curve) [8]. From this example, we see that we have no observation data that directly can reveal the shape of $f(u)$ in the interval $u \in [0.3, 0.7]$ (approximately).

2 Related Works

Several machine learning approaches are proposed to solve the PDEs. Raissi et al. [15] introduced the physics informed neural networks (PINNs) for solving solutions of PDEs and learning the parameters in PDEs. However, the neural network methods struggle to learn the nonlinear hyperbolic PDE that governs two-phase transport in porous media [4]. They experimentally indicated that this shortcoming of PINNs for hyperbolic PDEs is not due to the specific architecture or to the choice of the hyperparameters, but rather to the lack of regularity in the solution. Long et al. [11, 12] proposed a PDE-Net that combines numerical approximations of differential operators and a symbolic multi-layer neural network. They employed convolutions to approximate differential operators and deep networks to approximate the nonlinear response. However, in our case, $f(u)_x$ can not be written by $f'(u)u_x$ as $f(u)$ is not in general a differentiable function in our problem. Recently, more and more GNNs methods are being used to solve PDE problems. Gao et al. presented a novel discrete PINN framework based on Graph Convolutional Networks (GCNs) and the variational structure of PDEs in [5]. This framework could solve forward and inverse PDEs in a unified manner. Zhao et al. used GNNs in conjunction with autodecoder style priors to tackle PDE-constrained inverse problems [22]. In [2], authors combined GNNs with some classical numerical methods, such as finite differences, finite volumes, and WENO schemes to solve PDEs problems. They experimentally proved that this method has fast, stable, and accurate performance across different domain topologies on various fluid-related flow problems. However, these current works mainly use GNNs on some typical equations with regular solutions. Several complex equations with discontinuous solutions still need to be researched. In a recent work [9], we introduced a framework coined ConsLaw-Net that combines a symbolic multi-layer neural network and an entropy-satisfying discrete scheme to learn the non-

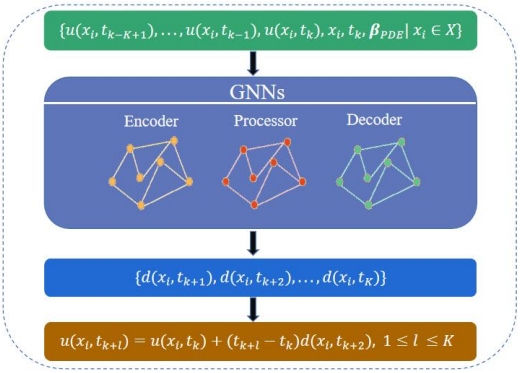


Figure 2: The pipeline of the approach. For each node c_i, at time point t_k, the last K solutions $(u(x_i,t_{k-K+1}),...,u(x_i,t_{k-1}),u(x_i,t_k))$, position x_i, current time t_k, and equation parameters $\boldsymbol{\beta}_{PDE}$ are fed into GNNs. After the operators of *Encoder*, *Processor* and *Decoder*, we get the information increment of this node c_i at the next K time points: $(d(x_i,t_{k+1}),d(x_i,t_{k+2}),...,d(x_i,t_{k+K}))$. Finally, we get solutions of the next K time points by $u(x_i,t_{k+l})=u(x_i,t_k)+(t_{k+l}-t_k)d(x_i,t_{k+l}),1\le l\le K$.

linear, unknown flux function $f(u)$ without parameter β.

3 Method

3.1 Graph Neural Networks (GNNs)

In this paper, we leverage GNNs with encoder processor decoder structure to learn a fast-forward model that predicts solutions of PDEs. We model the domain X as a graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$ with node $c_i\in\mathcal{V}$, edges $l_{ij}\in\mathcal{E}$. The features of node c_i are remarked by $\boldsymbol{f}_i\in R^c$ and the edges l_{ij} define local neighborhoods. An overview of the approach that we use is shown in Fig. 2.

Encoder *Encoder* computes node embeddings. For each node c_i, the operator of *Encoder* maps the last K solution values $(u(x_i,t_{k-K+1}),...,u(x_i,t_{k-1}),u(x_i,t_k))$, node position x_i, current time t_k, and equation parameters $\boldsymbol{\beta}_{PDE}$ to node embedding vector

$$\begin{aligned}\boldsymbol{f}_i=&\boldsymbol{\epsilon}([u(x_i,t_{k-K+1}),...,u(x_i,t_{k-1}),u(x_i,t_k)\\&,x_i,t_k,\boldsymbol{\beta}_{PDE}])\end{aligned}\tag{2}$$

where $\boldsymbol{\epsilon}$ representing the operator of *Encoder*.

Processor *Processor* contains M layers with intermediate graphs $\mathcal{G}^m$, $m=1,2,...,M$. Eq. (3) and Eq. (4) is the messaging of edges and the information update of node c_i, respectively.

- edge $c_j\to c_i$ message:

$$\begin{aligned}\boldsymbol{m}_{ij}^m=\boldsymbol{\phi}(&\boldsymbol{f}_i^m,\boldsymbol{f}_j^m,u(x_i,t_{k-K+1})-u(x_j,t_{k-K+1})\\&,...,u(x_i,t_k)-u(x_j,t_k),x_i-x_j,\boldsymbol{\beta}_{PDE})\end{aligned}\tag{3}$$

- node c_i update:

$$\boldsymbol{f}_i^{m+1}=\boldsymbol{\psi}\left(\boldsymbol{f}_i^m,\sum_{j\in N(i)}\boldsymbol{m}_{ij}^m,\boldsymbol{\beta}_{PDE}\right)\tag{4}$$

where $N(i)$ holds the neighbors of node c_i, and $\boldsymbol{\phi}$ and $\boldsymbol{\psi}$ are multilayer perceptrons (MLPs). Using relative positions x_i-x_j can be justified by the translational symmetry of the PDEs we consider. Solution differences u_i-u_j make sense by thinking of the message passing as a local difference operator, like a numerical derivative operator.

Decoder After *Processor*, we use a shallow convolutional network to output the K next timesteps information increment at grid point $x_i\in X$. The result is a new vector $\boldsymbol{d}_i=(d(x_i,t_{k+1}),d(x_i,t_{k+2}),...,d(x_i,t_{k+K}))$ with each element $d(x_i,t_{k+l}),1\le l\le K$ corresponding to different time point t_{k+l}.

Readout After GNNs with the structure of encoder processor decoder, we get solutions of the next K time points by

$$u(x_i,t_{k+l})=u(x_i,t_k)+(t_{k+l}-t_k)d(x_i,t_{k+l})\tag{5}$$

where $1\le l\le K$.

3.2 Data Generator

We investigate a discretization of the spatial domain $[0,L]$ in terms of $\{x_i\}_{i=0}^{N_x-1}$ where $x_i=(1/2+i)\Delta x$ for $i=0,\ldots,N_x-1$ with $\Delta x=L/N_x$.

Algorithm 1: CFL

Input: L: length of the spatial domain; N_x: the number of spatial grid cells; $f(u)$: the nonlinear flux function; T: computational time period;

Output: Δt: local time interval

Function `CFL`$(L, N_x, f(u), T)$**:**

$\Delta x = L/N_x$

$M = \max_u |f'(u)|$

$dt = (\frac{3}{4}\Delta x)/(M + 0.0001)$

$n\_time = \lfloor T/dt \rfloor$

$\Delta t = T/n\_time$

return Δt;

End Function

Furthermore, we consider time lines $\{t^n\}_{n=0}^{N_t}$ with $N_t \Delta t = T$. The discretization of Eq. (1) is based on the Rusanov scheme [8] which is expressed as

$$u_j^{n+1} = u_j^n - \lambda(F_{j+1/2}^n - F_{j-1/2}^n), \qquad \lambda = \frac{\Delta t}{\Delta x},$$
$$u_1^{n+1} = u_2^{n+1}, \qquad u_{N_x}^{n+1} = u_{N_x-1}^{n+1} \tag{6}$$

with $j = 2, \ldots, N_x - 1$ and the Rusanov flux takes the form

$$F_{j+1/2}^n = \frac{f(u_j^n) + f(u_{j+1}^n)}{2} - \frac{\max\{|f'(u_j^n)|, |f'(u_{j+1}^n)|\}}{2}(u_{j+1}^n - u_j^n).$$

The Courant–Friedrichs–Lewy(CFL) condition [8] determines the magnitude of Δt for a given Δx. We detail the CFL condition in Algorithm 1. We illustrate how to learn the solution $U = \{u(x_j, t^n)\}$ of the discrete conservation law Eq. (6) in Algorithm 2.

Algorithm 2: DataGenerator

Input: T: computational time period; N_x: the number of spatial grid cells; L: length of the spatial domain; $u_0 = \{u_0(x_j)\}_{j=1}^{N_x}$: initial state set of dimension N_x; $f(u)$: the flux function;

Output: $U = \{u_j^n\}$: the solution based on initial state u_0;

Δt = CFL(L, N_x, $f(u)$, T)

$\Delta x = L/N_x$

$U[0] = u_0$

$\tilde{u} = u_0$

for n = 1,...,$T/\Delta t$ **do**

  for j = 1,...,N_x - 1 **do**

    $F_{j+1/2} = \frac{1}{2}(f(\tilde{u}_j) + f(\tilde{u}_{j+1})) - \frac{\max\{|f'(\tilde{u}_j)|, |f'(\tilde{u}_{j+1})|\}}{2}(\tilde{u}_{j+1} - \tilde{u}_j)$

  end

  for j = 2,...,N_x - 1 **do**

    $u_j = \tilde{u}_j - \frac{\Delta t}{\Delta x}\left(F_{j+1/2} - F_{j-1/2}\right)$

  end

  $u_1 = u_2$

  $u_{N_x} = u_{N_x - 1}$

  $\tilde{u} = u$

  $U[i] = u$

end

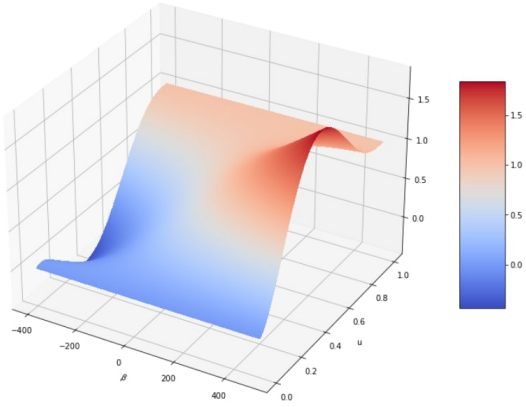
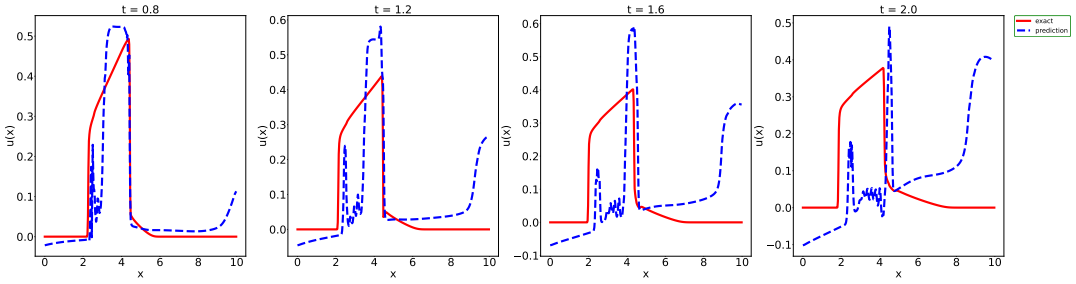


Figure 3: Flux function Eq. (7).

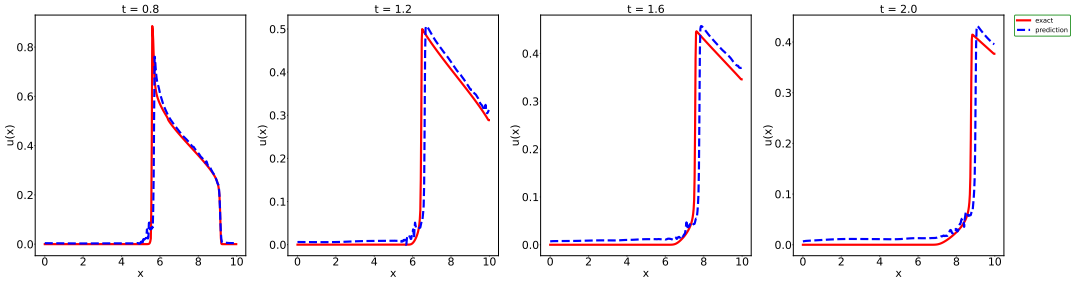
4 Experiments

Experiment Setup

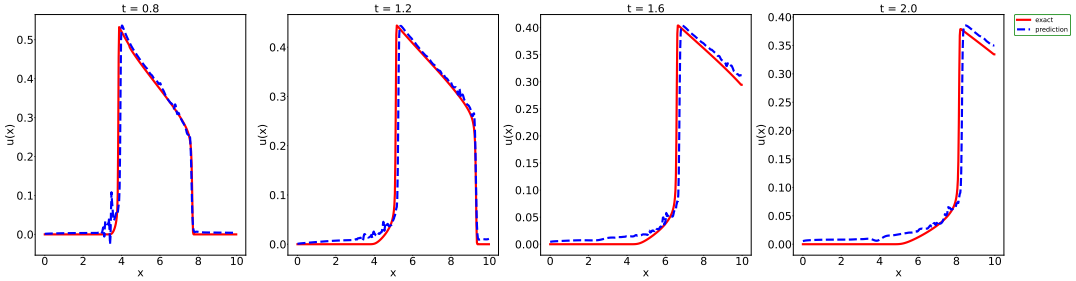
In this section, we study a class of nonlinear conservation laws that are naturally from the problems where one fluid is displaced by another fluid in a vertical domain. The displacement process involves a balance between buoyancy and viscous forces. Depending on the properties of the used fluids, there could be various displacement processes. One can derive a family of flux functions $f(u, \beta)$ in



(a) $\beta = -252$ in flux function (7). Solutions of (1) based on initial state $u_0 = \begin{cases} 0.5, & \text{if } x \in [2.575, 4.3] \\ 0.0, & \text{otherwise} \end{cases}$.



(b) $\beta = 264$ in flux function (7). Solutions of (1) based on initial state $u_0 = \begin{cases} 0.88, & \text{if } x \in [4.675, 6.4] \\ 0.0, & \text{otherwise} \end{cases}$.



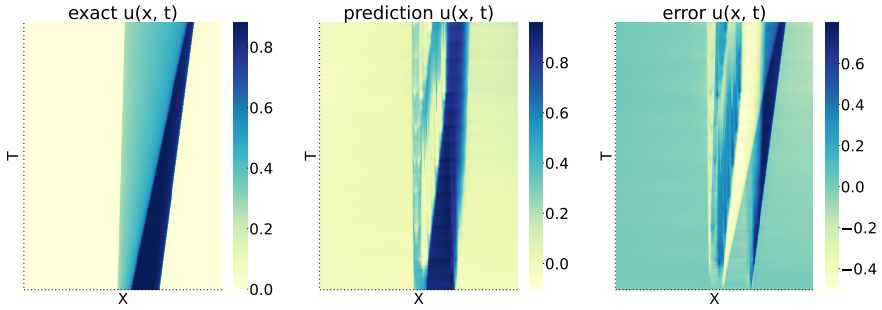
(c) $\beta = 360$ in flux function (7). Solutions of (1) based on initial state $u_0 = \begin{cases} 0.86, & \text{if } x \in [2.675, 4.4] \\ 0.0, & \text{otherwise} \end{cases}$.

Figure 4: Solutions of (1) based on different initial states and values of β at time point $t = 0.8, 1.2, 1.6$ and 2.0, respectively. In each subplot, the solid red line is the true solution obtained by Algorithm 2, and the blue dashed line is the solution predicted by the trained GNNs.

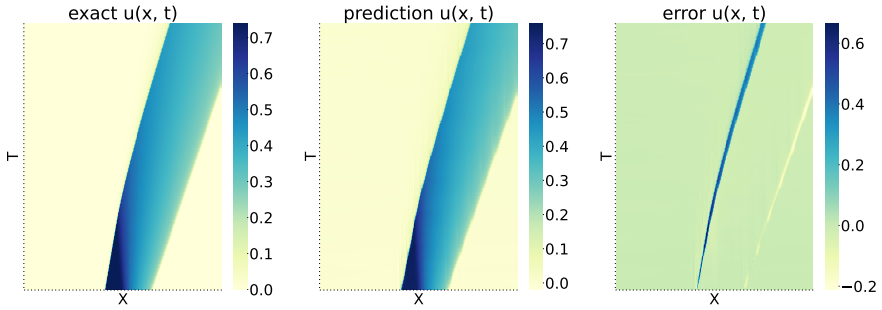
Eq. (1) which takes the form [19]

$$f(u, \beta) = \frac{1}{2}u(3-u^2)+\frac{\beta}{12}u^2\left(\frac{3}{4} - 2u + \frac{3}{2}u^2 - \frac{1}{4}u^4\right), \tag{7}$$

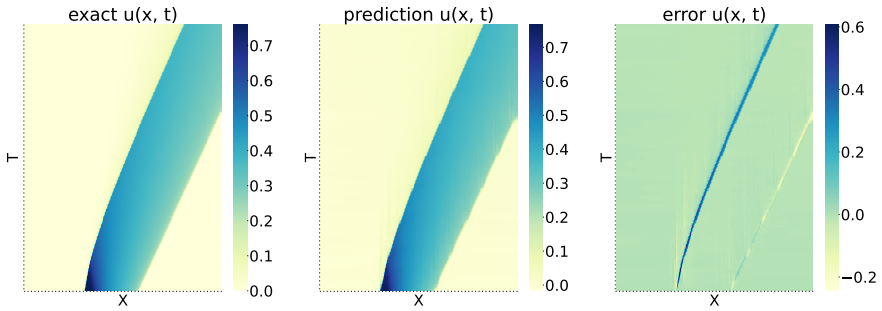
where the parameter $\beta \in (-400, 400)$ represents the balance between gravity (bouyancy) and viscous forces. We study the solution of Eq. (1) at $t \in [0, 2]$ in $x \in [0, 10]$. As shown in Fig. 3, different values of β result in different types of flux



(a) $\beta = -172$ in flux function (7). Solutions of (1) based on initial state $u_0 = \begin{cases} 0.88, & \text{if } x \in [4.675, 6.4] \\ 0.0, & \text{otherwise} \end{cases}$.



(b) $\beta = 192$ in flux function (7). Solutions of (1) based on initial state $u_0 = \begin{cases} 0.74, & \text{if } x \in [3.55, 5.275] \\ 0.0, & \text{otherwise} \end{cases}$.



(c) $\beta = 336$ in flux function (7). Solutions of (1) based on initial state $u_0 = \begin{cases} 0.76, & \text{if } x \in [2.475, 4.2] \\ 0.0, & \text{otherwise} \end{cases}$.

Figure 5: Solutions of (1) based on different initial states and values of β at time $t \in [0.0, 2.0]$. The left subplot is the real solutions of Eq. (1). The middle subplot is the solutions generated by GNNs, and the right subplot is the error of real and predicted $u(x, t)$.

functions.

We use Algorithm 2 to generate 500 samples with different values of β and initial states. The values of β are randomly selected from $(0, 300)$ and initial states are from $[0.0, 1.0]$. We consider the observation in terms of x-dependent data at fixed times $\{t_i^*\}_{i=1}^{N_{\text{obs}}}$ extracted from the solution U as follows:

$$U_{\text{sub}} = \left\{ u(x_j, t_1^*), u(x_j, t_2^*), \ldots, u(x_j, t_{N_{\text{obs}}}^*) \right\} \quad (8)$$

where $j = 1, \ldots, N_x$, $N_x = 400$ with $\Delta x = 0.025$ and $N_{\text{obs}} = 250$ with $\Delta t^{\text{obs}} = 0.008$.

For training, we split U_{sub} into five parts of length 50 in terms of time. Values of $u(x)$ for the first 25 time points are used to predict $u(x)$ for the following 25 time points, i.e., $K = 25$ in Eq. (5). In the testing, we use the first stage, which contains 25 time points of $u(x)$ to predict the solutions of the second stage, and then predict $u(x)$ at the third phase based on the predicted value of the second stage. This process is repeated until $T = 2$.

Results and discussions

The total number of trainable parameters is 0.28M. We set *epoch* = 30 and divide the 100 test samples into three groups according to β, (i)G_1: 50 examples with $\beta \in (-400, 0)$, (ii)G_2: 30 examples with $\beta \in (0, 300)$,(iii)G_3: 20 examples with $\beta \in (300, 400)$.

We randomly select one sample $\beta = -252, 264, 360$ from each group. Fig. 4 displays the exact and predicted $u(x, t)$ of these selected examples with $x \in [0, 10]$ at time $t = 0.8, 1.2, 1.6, 2.0$. All the examples show that the error between exact and predicted values of $u(x)$ increases with time. Notably, the model cannot predict the underlying trend at $t = 1.6$ and $t = 2.0$ on the case selected from G_1. While, it performs very well on G_2 and G_3. The result is also reflected in Table 1 that counts MSE of G_1, G_2, G_3. The MSE of G_1 is 10 times higher than the MSE of G_2 and G_3.

Fig. 5 shows another set of samples from G_1, G_2, G_3 with $x \in [0, 10)$ and $t \in [0.4, 2.0]$. When the solution fluctuates, the model is prone to large prediction errors. This phenomenon is also reflected in Fig. 4. This phenomenon is probably related to the discontinuity of the solutions of 1.

Experimental results show that the model can predict more accurately when β is within a specific range. However, when β deviates too much from that used for the training model, the model's predictive power is significantly reduced.

Group	G_1	G_2	G_3
MSE	0.0268	0.004	0.0015

Table 1: The value of MSE for each group. We calculate MSE of the true and predicted $u(x, t)$ of all samples with $x \in [0, 10]$ and $t \in [0.4, 2.0]$in each group.

5 Conclusion

In this work, we apply GNNs to solve PDEs, which are unique since their solutions usually contain shocks. Our experiments demonstrate that the GNN models can predict accurately when the parameter β is within a specific range. However, when the parameter deviates far from the value for training, the model's predictive performance drops sharply. Furthermore, the model is not very good at predicting the discontinuity of the solutions. There will be numerous small fluctuations around the discontinuity. One of our future work directions is to explore how to improve GNN's behavior when handling discontinuous solutions or develop new methods to deal with PDEs containing shocks.

References

[1] P. Bauer, A. Thorpe, and G. Brunet. The quiet revolution of numerical weather prediction. *Nature*, 525(7567):47–55, 2015. doi: https://doi.org/10.1038/nature14956.

[2] J. Brandstetter, D. E. Worrall, and M. Welling. Message passing neural PDE solvers. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. doi: https://doi.org/10.48550/arXiv.2202.03376. URL `https://openreview.net/forum?id=vSix3HPYKSU`.

[3] T. D. Economon, F. Palacios, S. R. Copeland, T. W. Lukaczyk, and J. J. Alonso. Su2:

An open-source suite for multiphysics simulation and design. *Aiaa Journal*, 54(3):828–846, 2016. doi: https://doi.org/10.2514/1.J053813.

[4] O. Fuks and H. A. Tchelepi. Limitations of physics informed machine learning for nonlinear two-phase transport in porous media. *Journal of Machine Learning for Modeling and Computing*, 1(1), 2020. doi: 10.1615/JMachLearnModelComput.2020033905.

[5] H. Gao, M. J. Zahr, and J.-X. Wang. Physics-informed graph neural galerkin networks: A unified framework for solving pde-governed forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 390:114502, 2022. doi: https://doi.org/10.1016/j.cma.2021.114502.

[6] H. Holden and N. Risebro. Front tracking for hyperbolic conservation laws. *Springer, Berlin*, 2011.

[7] V. Iakovlev, M. Heinonen, and H. Lähdesmäki. Learning continuous-time pdes from sparse data with graph neural networks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. doi: https://doi.org/10.48550/arXiv.2006.08956. URL `https://openreview.net/forum?id=aUX5Plaq7Oy`.

[8] R. LeVeque. Finite volume methods for hyperbolic problems. *Cambridge Texts in Applied Mathematics*, 2007. doi: https://doi.org/10.1017/CBO9780511791253.

[9] Q. Li and S. Evje. Learning the nonlinear flux function of a hidden scalar conservation law from data. *Network Heterogeneous Media*, 18, 2023. doi: 10.3934/nhm.2023003.

[10] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020.

[11] Z. Long, Y. Lu, X. Ma, and B. Dong. Pde-net: Learning PDEs from data. In *International Conference on Machine Learning*, pages 3208–3216. PMLR, 2018. doi: https://doi.org/10.48550/arXiv.1710.09668.

[12] Z. Long, Y. Lu, and B. Dong. PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, 399:108925, 2019. doi: https://doi.org/10.1016/j.jcp.2019.108925.

[13] D. Pardo, L. Demkowicz, C. Torres-Verdin, and M. Paszynski. A self-adaptive goal-oriented hp-finite element method with electromagnetic applications. part ii: Electrodynamics. *Computer methods in applied mechanics and engineering*, 196(37-40):3585–3597, 2007. doi: https://doi.org/10.1016/j.cma.2006.10.016.

[14] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020. doi: https://doi.org/10.48550/arXiv.2010.03409.

[15] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse prvoblems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. doi: https://doi.org/10.1016/j.jcp.2018.10.045.

[16] R. Ramamurti and W. Sandberg. Simulation of flow about flapping airfoils using finite element incompressible flow solver. *AIAA journal*, 39(2):253–260, 2001. doi: https://doi.org/10.2514/2.1320.

[17] C. Schwarzbach, R.-U. Börner, and K. Spitzer. Three-dimensional adaptive higher order finite element simulation for geo-electromagnetics—a marine csem example. *Geophysical Journal International*, 187(1):63–74, 2011. doi: https://doi.org/10.1111/j.1365-246X.2011.05127.x.

[18] S. Seo, C. Meng, and Y. Liu. Physics-aware difference graph networks for sparsely-observed dynamics. In *International Conference on Learning Representations*, 2019.

[19] H. Skadsem and S. Kragset. A numerical study of density-unstable reverse circulation displacement for primary cementing. *J. Energy Resour. Technol*, 144, 2022. doi: https://doi.org/10.1115/1.4054367.

[20] N. Thuerey, K. Weißenow, L. Prantl, and X. Hu. Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows. *AIAA Journal*, 58(1):25–36, 2020. doi: https://doi.org/10.2514/1.j058291.

[21] N. Wandel, M. Weinmann, and R. Klein. Learning incompressible fluid dynamics from scratch - towards fast, differentiable fluid models that generalize. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. doi: https://doi.org/10.48550/arXiv.2006.08762. URL `https://openreview.net/forum?id=KUDUoRsEphu`.

[22] Q. Zhao, D. B. Lindell, and G. Wetzstein. Learning to solve pde-constrained inverse problems with graph networks. In K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 26895–26910. PMLR, 2022. doi: https://doi.org/10.48550/arXiv.2206.00711. URL `https://proceedings.mlr.press/v162/zhao22d.html`.

Paper III:
Learning the Nonlinear Flux Function of a Hidden Scalar Conservation Law from Data

Qing Li, Steinar Evje

Networks and Heterogeneous Media 18.1 (2023): 48-79
doi = http://www.aimspress.com/article/doi/10.3934/nhm.2023003

Networks and Heterogeneous Media

http://www.aimspress.com/journal/nhm

NHM, 18(1): 48–79.
DOI: 10.3934/nhm.2023003
Received: 8 July 2022
Revised: 24 August 2022
Accepted: 31 August 2022
Published: 20 October 2022

Research article

Learning the nonlinear flux function of a hidden scalar conservation law from data

Qing Li and Steinar Evje*

University of Stavanger, Department of Energy and Petroleum, Group of Computational Engineering, Stavanger, Norway

* **Correspondence:** Email: steinar.evje@uis.no.

Abstract: Nonlinear conservation laws are widely used in fluid mechanics, biology, physics, and chemical engineering. However, deriving such nonlinear conservation laws is a significant and challenging problem. A possible attractive approach is to extract conservation laws more directly from observation data by use of machine learning methods. We propose a framework that combines a symbolic multi-layer neural network and a discrete scheme to learn the nonlinear, unknown flux function $f(u)$ of the scalar conservation law

$$u_t + f(u)_x = 0 \qquad (*)$$

with u as the main variable. This identification is based on using observation data $u(x_j, t_i)$ on a spatial grid $x_j,\ j = 1, \ldots, N_x$ at specified times $t_i,\ i = 1, \ldots, N_{obs}$. A main challenge with Eq $(*)$ is that the solution typically creates shocks, i.e., one or several jumps of the form (u_L, u_R) with $u_L \neq u_R$ moving in space and possibly changing over time such that information about $f(u)$ in the interval associated with this jump is sparse or not at all present in the observation data. Secondly, the lack of regularity in the solution of $(*)$ and the nonlinear form of $f(u)$ hamper use of previous proposed physics informed neural network (PINN) methods where the underlying form of the sought differential equation is accounted for in the loss function. We circumvent this obstacle by approximating the unknown conservation law $(*)$ by an entropy satisfying discrete scheme where $f(u)$ is represented through a symbolic multi-layer neural network. Numerical experiments show that the proposed method has the ability to uncover the hidden conservation law for a wide variety of different nonlinear flux functions, ranging from pure concave/convex to highly non-convex shapes. This is achieved by relying on a relatively sparse amount of observation data obtained in combination with a selection of different initial data.

Keywords: nonlinear conservation law; flux function; machine learning method; discrete scheme; entropy condition

1. Introduction

1.1. Background

Inspired by the vigorous development of AI and big data technology in recent decades, researchers currently pay much attention to deriving partial differential equations (PDEs) based on neural networks combined with observation data. Earlier attempts on data-driven discovery of hidden physical laws include [1] and [2]. They used symbolic regression to learn multiple models from basic operators and operands to explain observed behavior and then chose the best model from candidate models by the advantage of new sets of initial conditions. In the more recent studies of [3–5], authors employed Gaussian process regression [6] to devise functional representations that are tailored to a given linear differential operator. They were able to accurately infer solutions and provide uncertainty estimates for several prototype problems in mathematical physics. However, local linearization of any nonlinear term in time and certain prior assumptions of the Bayesian nature of Gaussian process regression limit the representation capacity of the model. Other researchers represented by [7–10] have proposed an approach using sparse regression. They constructed a dictionary of simple functions and partial derivatives that were likely to appear in the unknown governing equations. Then, they took advantage of sparsity promoting techniques to select candidates that most accurately represent the data. Raissi et al., [11] introduced physics informed neural network (PINN) for solving two main classes of problems: data-driven solution and data-driven discovery of partial differential equations. They suggested that if the considered PDE is well-posed and its solution is unique, then the PINN method is capable of achieving good predictive accuracy given a sufficiently expressive neural network architecture and a sufficient number of collocation points. The method was explored for Schrödinger equation, Allen-Cahn equation, and Korteweg-de Vries (KdV) in one dimension (1D) and Navier-Stokes in two dimensions (2D). However, the neural network methods struggle in learning the nonlinear hyperbolic PDE that governs two-phase transport in porous media [12]. They experimentally indicate that this shortcoming of PINN for hyperbolic PDEs is not related to the specific architecture or to the choice of the hyperparameters but is related to the lack of regularity in the solution. Long et al., [13, 14] proposed a combination of numerical approximation of differential operators by convolutions and a symbolic multi-layer neural network for model recovery. They used convolutions to approximate differential operators with properly constrained filters and to approximate the nonlinear response by deep neural networks. Models that are explored include Burgers equation

$$u_t + \lambda_1 u \cdot \nabla u = \lambda_2 \Delta u \tag{1.1}$$

where the constant parameters λ_1 and λ_2 must be learned. Moreover, the advection-diffusion equation

$$u_t + k(x) \cdot \nabla u = \lambda_2 \Delta u \tag{1.2}$$

was also considered. Herein, the parameter λ_2 is known whereas $k(x) = (a(x), b(x))^\top$ is the unknown space-dependent coefficient to be learned. Furthermore, the diffusion-reaction problem given by Eq (1.3),

$$u_t = \lambda_2 \Delta u + g(u) \tag{1.3}$$

where $g(u)$ is unknown and must be found by means of the observation data, was also successfully demonstrated.

In this work we focus on the problem of learning the unknown *nonlinear* flux function $f(u)$ that is involved in the general scalar nonlinear conservation law, here restricted to the one-dimensional case, given by Eq (1.4)

$$u_t + f(u)_x = 0 \tag{1.4}$$

where $u = u(x,t)$ is the main variable. Burgers equation (1.1) amounts to the case with flux function $f(u) = \frac{\lambda_1}{2}u^2$ in Eq (1.4). Hence, the main challenge in Eq (1.4) is that the flux function $f(u)$, which is a nonlinear function of u, itself is unknown and there is no viscous term in the form u_{xx} that can regularize the solution.

1.2. Problem statement and novelty

The learning of the nonlinear flux function $f(u)$ in Eq (1.4) involves several new aspects as compared to the other PDE models Eqs (1.1)–(1.3).

(i) Lack of observation data. It is well known that Eq (1.4) will generate shock wave solutions $u(x,t)$ in finite time, i.e., solutions that contain one or sevel discontinuities expressed as a jump (u_L, u_R) with $u_L \neq u_R$, despite the fact that initial data $u_0(x)$ is smooth [15, 16]. In particular, the specific form of $f(u)$ in the interval $[\min(u_L,u_R), \max(u_L,u_R)]$ is not used in the construction of the entropy solution, only the slope $s = \frac{f(u_L)-f(u_R)}{u_L-u_R}$. As jumps arise and disappear in the solution over the time period for which observation data is collected, the data may lack information about $f(u)$. An illustration of this situation is given in Figure 1. In the left panel we plot the flux function $f(u) = u^2/(u^2 + (1-u)^2)$. In the right panel the entropy solution after a time $T = 0.5$ is shown. At time $t = 0$, the initial data $u_0(x)$ involves one jump at $x = 0$ and another jump at $x = 1$. The initial jump at $x = 0$ is instantly transformed into a solution that is a combination of a continuous wave solution (rarefaction wave) and a discontinuous wave $(u_L, u_R) \approx (0.3, 1.0)$, as dictated by the lower convex envelope shown in the left panel (green curve) [15]. Similarly, the initial jump at $x = 1$ is transformed into a solution that is a combination of a continuous wave solution (rarefaction wave) and a discontinuous wave $(u_L, u_R) \approx (0.7, 0)$, in accordance with the upper concave envelope illustrated in left panel (brown curve) [15]. From this example, we see that we have no observation data that directly can reveal the shape of $f(u)$ in the interval $u \in [0.3, 0.7]$ (approximately).

(ii) Lack of regularity. Previous work based on the PINN approaches mentioned above relies on imposing the structure of the underlying PDE model by including an error term in the loss function. This would amount to computing the left-hand-side of Eq (1.4). Due to the lack of regularity in the solution as illustrated by the example in Figure 1 (right panel), it seems not clear how to implement this in a PINN framework. We refer to [12] for investigations related to this point which found that it was necessary to consider the viscous approximation $u_t + f(u)_x = \varepsilon u_{xx}$ with a small value $\varepsilon > 0$ to learn the forward solution.

1.3. Our approach

Our aim is to learn the unknown nonlinear function $f(u)$ from observation data in terms of solution behavior collected at different points (x_j, t_i) in space and time. The approach we explore in this work relies on the two following building blocks: (i) We represent the unknown function $f(u)$ by a symbolic

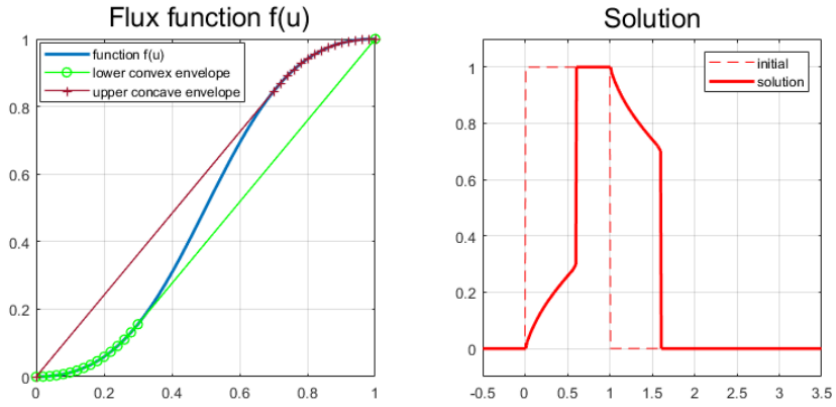


Figure 1. Left: Example of nonlinear flux function $f(u) = \frac{u^2}{u^2+(1-u)^2}$ (blue curve). Upper concave envelope (brown curve) and lower convex envelope (green curve) are also included. Right: The solution of Eq (1.4) at time $T = 0.5$ is shown (red solid curve) together with its initial data $u_0(x)$ (red dashed line).

multi-layer neural network; (ii) Instead of including the form of the conservation law Eq (1.4) explicitly in the loss function as in PINN, we use a standard simple entropy-satifying discrete scheme associated with Eq (1.4) to account for this information during the learning process. The numerical scheme allows us to evolve the given initial data over the relevant time interval and collect predicted data which is accounted for in the loss function.

Regarding point (i), inspired by the symbolic neural network that is used in [17, 18], we can learn an analytic expression that has a derivative similar to the true $f'(u)$. The reason why it is attractive to learn the analytic expression is that, unlike black box learning, system identification has explanatory value. That is, once we have extracted an analytical expression of the hidden flux function $f(u)$, e.g., based on data from a more or less complex system, the shape of the flux function provides precise insight into finer wave propagation mechanisms involved in the system under consideration. In particular, predictions can then be made for any other initial state. Our approach bears similarity to the underlying idea employed in the recent work [13, 14]. However, an essential difference is that the flux function $f(u)_x$ cannot be expressed by $f'(u)u_x$ as $f(u)$ is not in general a differentiable function in our problem. Therefore, we rely on using an entropy satifying discrete scheme, which is guaranteed to converge to the entropy solution of Eq (1.4) [15], to identify an analytical expression of the flux function $f(u)$ which is present in the numerical scheme in the form of a symbolic neural network.

1.4. Related work

James and Sepúlveda formulated the inverse problem of flux identification as that of minimizing a suitable cost function [19]. Relying on the viscous approximation, it was shown that the perturbed problem converged to the original hyperbolic problem [19] by letting the viscous term vanish. Holden et al used the front-tracking algorithm to reconstruct the flux function from observed solutions to problems with suitable initial data [20]. Several recent studies have addressed the reconstruction of the flux function for sedimentation problems that involve the separation of a flocculated suspension into a clear fluid and a concentrated sediment [21, 22]. In particular, Bürger and Diehl showed that

the inverse problem of identifying the batch flux density function has a unique solution, and derived an explicit formula for the flux function [23]. This method was recently extended to construct almost the entire flux function [24] by using a cone-shaped separator. For another interesting example of the challenge of identification of the unknown flux function $f(u)$, we refer to [25]. The author explored a direct inversion method based on using linear combinations of finite element hat functions to represent unknown nonlinear function f. Finally, for an example with neural networks used in combination with discrete schemes to optimize computations of a nonlinear conservation law, see [26].

The remainder of this paper is organized as follows: Section 2 gives a presentation of the approach that we explore. In Section 3 and Section 4 we conduct numerical studies where synthetic data has been generated from a general class of flux functions ranging from purely concave to highly non-convex functions. Concluding thoughts are given in Section 5.

2. Framework

2.1. Nonlinear conservation laws and entropy satisfying solutions

It is well known that conservation laws of the form Eq (1.4) do not in generall possess classical solutions. Instead one must consider weak solutions in the sense that the following integral equality holds [15, 27, 28]

$$\int_{\Omega_x}\int_0^T \Big[u\phi_t + f(u)\phi_x\Big]\,dx\,dt + \int_{\Omega_x} u_0(x)\phi(x,t=0)\,dx = 0 \tag{2.1}$$

for all $\phi \in C^1$ such that $\phi(x,t) : \Omega_x \times (0,T) \to \mathbb{R}$ and which is compactly supported, i.e., ϕ vanishes at $x \to \Omega_x$ and $t \to T$. It follows that if a discontinuity occurs in the solution, i.e., a left state u_L and a right state u_R, then it must propagate with the speed s given by [15, 28]

$$s = \frac{f(u_L) - f(u_R)}{u_L - u_R}. \tag{2.2}$$

This follows from mass conservation and, thus, must be satified across any discontinuity [15, 28]. However, direct calculations show that there are several weak solutions for one and the same initial data [27]. To overcome this issue of non-uniqueness of weak solutions, we need criteria to determine whether a proposed weak solution is admissible or not. This has led to the class of *entropy* solutions, which amounts to introducing an additional constraint which ensures that the unique physically relevant one is found among all the possible weak solutions.

There are different ways to express the entropy condition for scalar nonlinear conservation laws. One variant is by introducing an entropy pair (η, q) where $\eta : \mathbb{R} \to \mathbb{R}$ is any strictly convex function and $q : \mathbb{R} \to \mathbb{R}$ is constructed as [28, 29]

$$q(v) = \int_0^v f'(s)\eta'(s)\,ds \tag{2.3}$$

for any v. This implies that $q' = f'\eta'$. Then, u is an entropy solution of Eq (1.4) if (i) u is a weak solution in the sense of Eq (2.1); (ii) u satisfies in a weak sense $\eta(u)_t + q(u)_x \le 0$ for any pair (η, q). This condition can also be formulated as the following characterization of a discontinuity (u_L, u_R) [28, 30]: For all numbers v between u_L and u_R,

$$\frac{f(v) - f(u_L)}{v - u_L} \ge s \ge \frac{f(v) - f(u_R)}{v - u_R} \tag{2.4}$$

where s is given by Eq (2.2). This entropy condition can naturally be accounted for by introducing the upper concave envelope and lower convex envelope, as indicated in Figure 1 (left panel) [15, 30]. In particular, it gives a tool for constructing exact solutions.

From this characterization of the physically relevant solution of Eq (1.4), it is clear that there are special challenges pertaining to identification of the unknown flux function $f(u)$ from observation data. Firstly, it is a challenge with the more indirect characterization of the correct weak solution since it involves formulations like Eq (2.1) and Eq (2.4). This may hamper the use of PINN-based approaches. The approach we take in this work is to rely on a discrete scheme that represents an approximation to the entropy solution described above. A convenient feature of an entropy-consistent numerical scheme is that the entropy condition is automatically built into the scheme. I.e., as the grid is refined, the numerical solution converges to the admissible solution [15, 28, 29]. Secondly, it follows from the entropy condition Eq (2.4) that observation data that involves one or several discontinuities, may not contain information about the unknown flux function $f(u)$ in intervals that correpond to discontinuities in u. The example shown in Figure 1 shows an approximation to the entropy solution and obeys the entropy condition Eq (2.4). As mentioned above, the example indicates that we lack information about $f(u)$ in the interval $\approx [0.3, 0.7]$.

In this work we explore how we can deal with this situation by a proper combination of two different aspects: (i) we add a priori regularity to the unknown flux function $f(u)$ by representing it as a symbolic multi-layer neural network; (ii) we collect observation data by considering a set of different initial data that can help detecting the finer details of $f(u)$.

2.2. Entropy consistent discrete numerical scheme

Based on the given observation data, our aim is to identify a conservation law Eq (1.4) for $(x, t) \in [0, L] \times [0, T]$, written in the form Eq (2.5),

$$\begin{aligned} u_t + f(u)_x &= 0 \\ u_x|_{x=0} &= u_x|_{x=L} = 0 \\ u|_{t=0} &= u_0(x) \end{aligned} \tag{2.5}$$

where $f(u)$ is the unknown, possible nonlinear flux function and $u_0(x)$ is the initial state which is assumed known.

We consider a discretization of the spatial domain $[0, L]$ in terms of $\{x_i\}_{i=1}^{N_x}$ where $x_i = (1/2 + i)\Delta x$ for $i = 0, \ldots, N_x - 1$ with $\Delta x = L/N_x$. Furthermore, we consider time lines $\{t^n\}_{n=0}^{N_t}$ such that $N_t \Delta t = T$. We base our discrete version of Eq (1.4) on the Rusanov scheme [15] which takes the form Eq (2.6),

$$\begin{gathered} U_j^{n+1} = U_j^n - \lambda(F_{j+1/2}^n - F_{j-1/2}^n), \qquad \lambda = \frac{\Delta t}{\Delta x}, \\ U_1^{n+1} = U_2^{n+1}, \qquad U_{N_x}^{n+1} = U_{N_x-1}^{n+1} \end{gathered} \tag{2.6}$$

with $j = 2, \ldots, N_x - 1$ and where the Rusanov flux takes the form Eq (2.7),

$$F_{j+1/2}^n = \frac{f(U_j^n) + f(U_{j+1}^n)}{2} - \frac{M}{2}(U_{j+1}^n - U_j^n), \qquad M \sim \max_u |f'(u)|. \tag{2.7}$$

We use a slightly modified version of the Rusanov flux by relying on a global estimate of $|f'(u)|$ instead of a local estimate of M in terms of $M_{j+1/2} = \max\{|f'(U_j^n)|, |f'(U_{j+1}^n)|\}$ [15]. The CFL condition [15] determines

the magnitude of Δt for a given Δx through the relation Eq (2.8),

$$CFL := \frac{\Delta t}{\Delta x} M \leq 1. \tag{2.8}$$

We have used $\frac{\Delta t}{\Delta x} M = CFL \leq \frac{3}{4} < 1$ when we compute solutions involved in the learning process. The training process involves repeated use of the discrete scheme Eq (2.6) for different flux functions $f(u)$. This requires repeated estimation of the parameters Δt and M that will be used for calculation of predicted data based on Eq (2.6), according to the CFL condition Eq (2.8). Finally, we note that the Rusanov flux falls within the class of monotone schemes and therefore is guaranteed to converge to the entropy solution [28–30].

2.3. Observation data set

We consider observation data in terms of x-dependent data at fixed times $\{t_i^*\}_{i=1}^{N_{\text{obs}}}$ extracted from the solution $U(x_j, t^n) = U_j^n$ as follows:

$$U_{\text{sub}} = \left\{U(x_j, t_1^*), U(x_j, t_2^*), \ldots, U(x_j, t_{N_{\text{obs}}}^*)\right\}, \qquad j = 1, \ldots, N_x. \tag{2.9}$$

We consider a domain of length L and consider simulations over the time period $[0, T]$. We apply a numerical grid composed of N_x grid cells when we compute numerical solutions of Eq (2.5) based on the numerical scheme Eq (2.6) and Eq (2.7). This is used both for obtaining the true solution and corresponding synthetic observation data (which we denote by U_{sub}) as well as when we compute predictions based on the ensemble of flux functions brought forth through training (which we denote by $\hat{U}_{\text{sub}}$). We specify times for collecting the time dependent data

$$T_{\text{obs}} = \{t_i^* = i\Delta t^{\text{obs}} : \quad i = 1, \ldots, N_{\text{obs}}\}. \tag{2.10}$$

We typically use $N_{\text{obs}} = 9$, with $T = 1$ i.e., $\Delta t^{\text{obs}} = 0.1$. In particular, the number N_{obs} of collected spatial-dependent data is relatively sparse. Also the number of local time steps (of length Δt) we need to compute numerical solutions through the discrete scheme Eq (2.6) and Eq (2.7) is much higher than the number of observation data, i.e., $\Delta t << \Delta t^{\text{obs}}$. Since Δt is dictated by the CFL condition for the given choice of the flux function f (which will vary during the training), we do not known that $\Delta t K_i = t_i^*$ for $i = 1, \ldots, N_x$ for some integer K_i. In that case, we choose the one that is closest to t_i^*.

2.4. Main algorithms

Specifically, we use Algorithm 1 to calculate the parameters Δt and M which are needed as input to Algorithm 2. Then, we use Algorithm 2 to extract the solution $U(x_j, t^n) = U_j^n$ of the discrete conservation law Eq (2.6). Finally, from $\{U_j^n\}$ we extract the observation data set U_{sub} according to Eq (2.9).

2.5. Symbolic Multi-layer Neural Network to represent $f(u)$

We want to learn the analytical expression of the flux function $f(u)$, not just fit observations using neural networks as a black box. For that purpose we suggest to use the symbolic neural network (called S-Net) proposed in [17] and [18] to learn the unknown function $f(u)$ instead of fully connected

Algorithm 1: CFL Algorithm

Input: L: length of the spatial domain; N_x: the number of spatial grid cells; $f(u)$: the nonlinear flux function; T: computational time period;

Output: Δt: local time interval used in Eq (2.6); M: upper limit of $|f'(u)|$

$\Delta x = L/N$
$dfdu = grad(\alpha f, u)$
$max\_fprime = \max |dfdu|$
$dt = (\frac{3}{4}\Delta x)/(max\_fprime + 0.0001)$
$n\_time = round(T/dt, 0)$
$\Delta t = T/n\_time$
$M = max\_fprime$

Algorithm 2: DataGenerator

Input: T: computational time period; N_x: the number of spatial grid cells; L: length of the spatial domain; $u_0 = \{u_0(x_j)\}_{j=1}^{N_x}$: initial state vector of dimension N_x; $f(u)$: the flux function;

Output: $U = \{U_j^n\}$: the solution based on initial state u_0;

$(\Delta t, M)$ = CFL Algorithm(L, N_x, $f(u)$, T)
$time\_steps = T/\Delta t$
$\Delta x = L/N_x$
$u\_old = u_0$
$U = []$
$U.append(u\_old)$
for n = 1,...,*time_steps* **do**
  for j = 1,...,N_x - 1 **do**
    $f = f(u\_old)$
    $F\_half[j] = \frac{1}{2}\Big(f[j] + f[j+1]\Big) - \frac{M}{2}\Big(u\_old[j+1] - u\_old[j]\Big)$
  end
  for j = 2,...,N_x - 1 **do**
    $u[j] = u\_old[j] - \frac{\Delta t}{\Delta x}\Big(F\_half[j] - F\_half[j-1]\Big)$
  end
  $u[1] = u[2]$
  $u[N_x] = u[N_x - 1]$
  $u\_old = u$
  $U.append(u\_old)$
end

neural networks that have been used in, e.g., [11]. We also tested a graph neural network (GNN) method proposed in [31] to learn the hidden conservation law. The GNN method is based on using an evolution scheme of the form $U_j^{n+1} = U_j^n + \Delta t(\Delta U)_j^n$ where $(\Delta U)_j^n$ must be learned at each grid point. The authors of [31] employed GNN in the context of learning convection-diffusion equations.

However, this approach did not work well for the problem Eq (2.5). The reason may be related to the fact that, in our case, the solutions of the hyperbolic conservation law become discontinuous, whereas the PDE models studied in [31] have regular solutions.

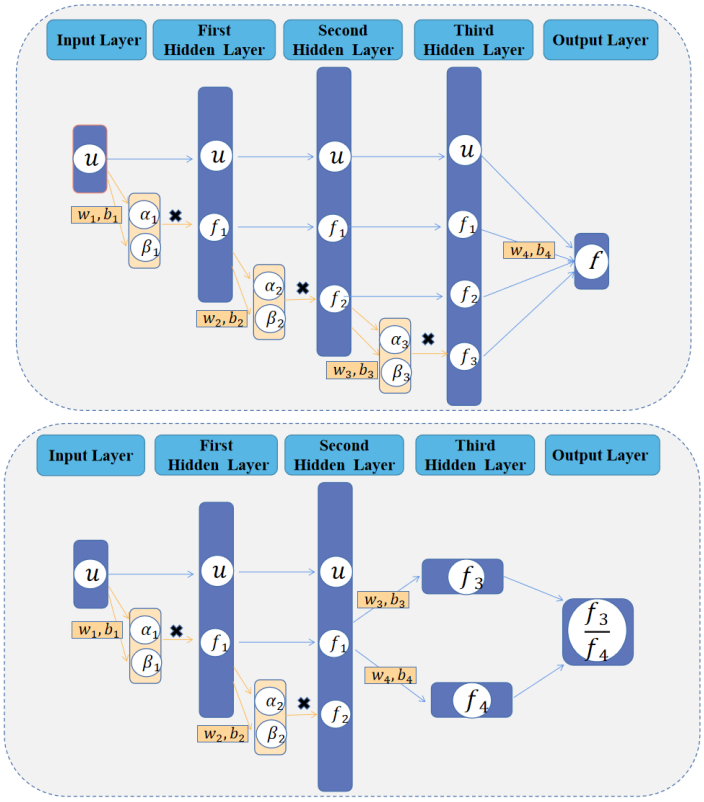


Figure 2. Top. The framework of S-Net-M for multiplicative function. **Bottom**. The framework of S-Net-D for division function.

In the S-Net setting, depending on whether we seek a function that takes a multiplicative form or a fractional form, we design two types of network structures illustrated, respectively, in Figure 2 (top) and Figure 2 (bottom). Take a three layers S-Net which can learn the expression of a function f possessing a multiplication form as an example. As shown in Figure 2 (top), the identity directly maps u from input layer to the first hidden layer. The linear combination map uses parameters $\mathbf{w_1}$ and $\mathbf{b_1}$ to choose two elements from u and are denoted by α_1 and β_1.

$$(\alpha_1, \beta_1)^T = \mathbf{w_1} \cdot (u) + \mathbf{b_1}, \mathbf{w_1} \in \mathbb{R}^{2\times 1}, \mathbf{b_1} \in \mathbb{R}^{2\times 1} \tag{2.11}$$

These two elements of α_1 and β_1 are multiplied in the PDE system.

$$f_1 = \alpha_1 \beta_1 \tag{2.12}$$

Apart from u gotten by the identity map, f_1 also is input to the second hidden layer.

$$(\alpha_2, \beta_2)^T = \mathbf{w_2} \cdot (u, f_1)^T + \mathbf{b_2}, \mathbf{w_2} \in \mathbb{R}^{2\times 2}, \mathbf{b_2} \in \mathbb{R}^{2\times 1} \tag{2.13}$$

Similarly with the first hidden layer, we get another combination $f_2(\alpha_2, \beta_2)$.

$$f_2 = \alpha_2 \beta_2 \tag{2.14}$$

Then we obtain α_3 and β_3 by means of $\mathbf{w_3}$ and $\mathbf{b_3}$ from u, f_1 and f_2.

$$(\alpha_3, \beta_3)^T = \mathbf{w_3} \cdot (u, f_1, f_2)^T + \mathbf{b_3}, \mathbf{w_3} \in \mathbb{R}^{2\times 3}, \mathbf{b_3} \in \mathbb{R}^{2\times 1} \tag{2.15}$$

f_3, which is the product of α_3 and β_3 is put into the third hidden layer.

$$f_3 = \alpha_3 \beta_3 \tag{2.16}$$

Finally, we arrive at the analytic expression of the function f.

$$f = \mathbf{w_4} \cdot (u, f_1, f_2, f_3)^T + \mathbf{b_4}, \mathbf{w_4} \in \mathbb{R}^{1\times 4}, \mathbf{b_4} \in \mathbb{R} \tag{2.17}$$

The difference between S-Net for multiplicative function (denoted S-Net-M) and S-Net for division function (denoted S-Net-D) is that in the third hidden layer in Figure 2 (bottom), we obtain the numerator part f_3 and the denominator part f_4 of the flux function $f(u)$ based on $\mathbf{w_3}$, $\mathbf{b_3}$ and $\mathbf{w_4}$, $\mathbf{b_4}$, respectively.

$$f_3 = \mathbf{w_3} \cdot (u, f_1, f_2)^T + \mathbf{b_3}, \mathbf{w_3} \in \mathbb{R}^{1\times 3}, \mathbf{b_3} \in \mathbb{R} \tag{2.18}$$

$$f_4 = \mathbf{w_4} \cdot (u, f_1, f_2)^T + \mathbf{b_4}, \mathbf{w_4} \in \mathbb{R}^{1\times 3}, \mathbf{b_4} \in \mathbb{R} \tag{2.19}$$

The analytic expression of the flux function f is the combination of f_3 and f_4.

$$f = \frac{f_3}{f_4} \tag{2.20}$$

The parameters involved in the network described above is denoted by θ and the resulting function according to Eq (2.17) or Eq (2.20) is denoted by $f_\theta(u)$. Herein, for the case above the ensemble of parameters used in the multi-layer symbolic neural network is given by

$$\theta_{\text{S-Net}} = \{\mathbf{w_1}, \mathbf{w_2}, \mathbf{w_3}, \mathbf{w_4}, \mathbf{b_1}, \mathbf{b_2}, \mathbf{b_3}, \mathbf{b_4}\}. \tag{2.21}$$

2.6. General architecture

The overall architecture of the method is shown in Figure 3. There are two parts involved, the generation of observed data based on the true flux function $f(u)$ and learning of the unknown flux function $f_\theta(u)$ which is assigned the S-Net structure. For the synthetic observation data, we use Algorithm 2 to obtain the approximate solution U of Eq (2.5) based on the exact flux function $f(u)$ combined with the scheme Eq (2.6). We select data U_{sub} at times as given by Eq (2.9). Concerning the learning process, firstly, we use S-Net to represent the function $f_\theta(u)$. $f_\theta(u)$, together with T, N_x, L, u_0 are fed into the DataGenerator to get the predicted solution $\hat{U}$. We also choose data $\hat{U}_{sub}$ at the same time points Eq (2.9). The difference between U_{sub} and $\hat{U}_{sub}$ is denoted as loss, and we use the second-order quasi-Newton method, L-BFGS-B ([32, 33]), to update the parameters θ of the S-Net involved in $f_\theta(u)$. This updating process iterates until we reach the number of epoch that we set or the process can't be optimized anymore. The learning process is shown in Algorithm 3 which we denote as ConsLaw-Net. Finally, we get the best flux function $f_{\theta^*}(u)$ and use it to represent the learned conservation law for further predictions.

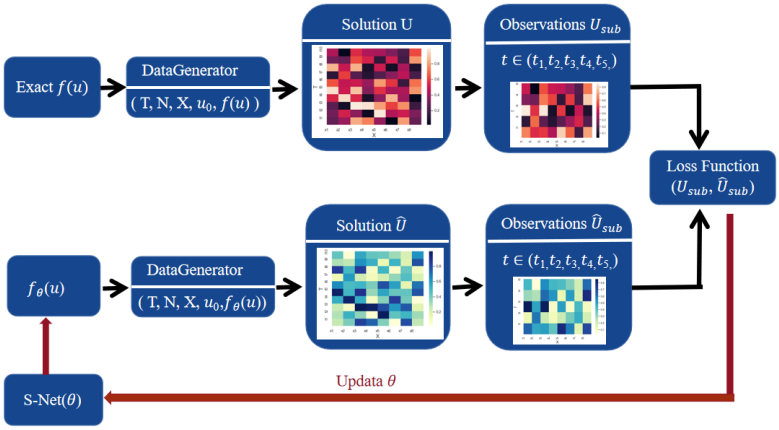


Figure 3. Schematic diagram of the framework.

Algorithm 3: ConsLaw-Net

Input: T: computational time period; N_x: the number of spatial grid cells; L: length of the spatial domain; u_0: initial state vector of dimension N_x; $f(u)$: true flux function; θ_0: initial parameters of S-Net; θ: parameters of S-Net; $f_\theta(u)$: flux function generated by S-Net; T_obs: observation time points Eq (2.10); mse: mean squared error Eq (2.23); $epoch$: the number of epochs; $DataGenerator$: Algorithm 2

Output: $f_{\theta^*}(u)$: the best flux function generated by the S-Net based on parameter vector θ^*;

$U = DataGenerator(T, N_x, L, u_0, f(u))$
$U\_sub = \{u \in U | t \in T\_obs\}$
$\theta = \theta_0$
for i = 1,...,$epoch$ **do**
  $\hat{U} = DataGenerator(T, N_x, L, u_0, f_\theta(u))$
  $\hat{U}\_sub = \{u \in \hat{U} | t \in T\_obs\}$
  $loss = mse(U\_sub, \hat{U}\_sub)$
  Updating θ by optimizer L-BFGS-B and loss;
end
$\theta^* = \theta$

2.6.1. Loss Function

We adopt the following loss function for the training of the S-Net function:

$$L = L^{data} \tag{2.22}$$

where data approximation L^{data} is obtained as follows: Assume that we have K different initial states used for the training process, and each predicted solution is described on a grid of $N = N_x$ grid cells and at $I = N_{\text{obs}}$ different times, as given by Eq (2.10). Through Algorithm 3 (ConsLaw-Net) the observation data set is first obtained, which is denoted by $\{U_{\text{sub},k}(x_j, t_i^*) : \ 1 \le k \le K;\ 1 \le j \le N;\ 1 \le i \le I\}$. Then, through an iterative loop in Algorithm 3, the predicted data is generated and is denoted by $\{\hat{U}_{\text{sub},k}(x_j, t_i^*) : \ 1 \le k \le K;\ 1 \le j \le N;\ 1 \le i \le I\}$. So we define the data approximation term L^{data} as:

$$L^{data} = \frac{1}{NKI} \sum_{k=1}^{K} \sum_{j=1}^{N} \sum_{i=1}^{I} \left\| U_{\text{sub},k}(x_j, t_i^*) - \hat{U}_{\text{sub},k}(x_j, t_i^*) \right\|^2 \tag{2.23}$$

3. Learning of nonlinear flux functions $f(u;\beta)$ involved in complex fluid displacement

In this section, we consider a class of nonlinear conservation laws that naturally arise from the problem of studying displacement of one fluid by another fluid in a vertical domain. The resulting displacement process involves a balance between buoyancy and viscous forces. Depending on the property of the fluids that are used, there is room for a whole range of different type of displacement processes. This is expressed by the fact that one can derive a family of flux functions $f(u;\beta)$ which takes the form [34]

$$f(u;\beta) = \frac{1}{2}u(3-u^2) + \frac{\beta}{12}u^2\left(\frac{3}{4} - 2u + \frac{3}{2}u^2 - \frac{1}{4}u^4\right). \tag{3.1}$$

The parameter β represents the balance between gravity (bouyancy) and viscous forces and, typically, $\beta \in [-200, 300]$. Different values of β result in different types of flux functions. As shown in Figure 4, the shape of $f(u;\beta)$ varies over a broad spectrum with $\beta \in \{-200, -100, 10, 100, 120, 200, 300\}$. In particular, we see that $f(u;\beta)$ can be purely concave ($\beta = 0$), but also have both one and two inflection points where the sign of f'' changes.

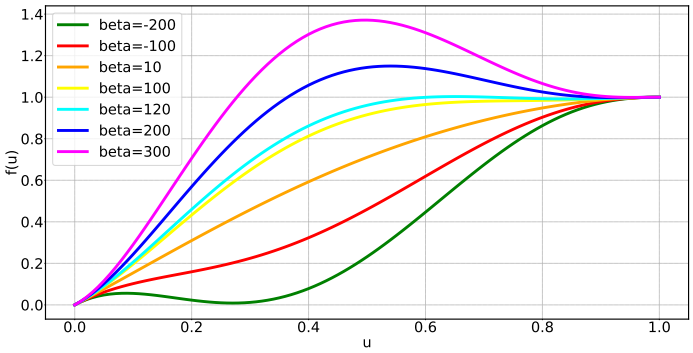


Figure 4. $f(u)$ with different values of β. Green, red, orange, yellow, cyan, blue and magenta line are generated by $\beta = -200$, $\beta = -100$, $\beta = 10$, $\beta = 100$, $\beta = 120$, $\beta = 200$, and $\beta = 300$, respectively.

In the following we generate synthetic data by specifying β and a class of initial data $u_0(x)$. We consider a spatial domain $L = 10$ such that $x \in [0, 10]$ and consider solutions in the time interval $[0, T]$ with $T = 2$. We collect observation data in the form Eq (2.9). The aim is to identify the unknown $f(u)$ for $u \in [0, 1]$. Since the solution of Eq (1.4) is TVD (total variation diminishing) [16, 29], we know that the solution $u(x, t)$ at any time $t > 0$ does not contain any new maxima or minima as compared to the initial data $u_0(x)$, i.e.,

$$\min u_0(x) \le u(x, t) \le \max u_0(x).$$

This feature is inherited by the discrete scheme Eq (2.6) we use [29, 30]. In order to learn $f(u)$ for $u \in [0, 1]$, we therfore consider a set of initial data $\{u_0^k\}_{k=1}^K$ such that $0 \le u_0^k(x) \le 1$. As the solution

$u(x,t)$ evolves over time, and corresponding observation data are collected in the form Eq (2.9), we hopefully can extract data which is sufficient to learn a reliable approximation to the true flux function.

As initial data we choose box-like states that give rise to Riemann problems, one at each initial discontinuity. Some of the questions we are interested in are:

(a) How much data do we need for learning the unknown flux function $f(u)$?

(b) How is the result of the learning of $f(u)$ sensitive to noise in the observation data?

(c) How is the question in (a) and (b) sensitive to different flux functions, i.e., to different $\beta \in [-200, 300]$ in light of Eq (3.1)?

(d) What is the role of using S-Net-M versus S-Net-D when we seek to identify the unknown flux function?

In the following we apply a numerical grid composed of $N_x = 200$ grid cells. We test effect of using finer grid in Section 4. We consider observation data Eq (2.9) with

$$\{t_i^*\} = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}. \tag{3.2}$$

3.1. Example with concave flux $f'' < 0$ corresponding to $\beta = 10$ in Eq (3.1)

We use the S-Net-M given by Eq (2.17) to represent the unknown flux function $f_\theta(u)$. We use three hidden layers, and the total number of trainable parameters is then 23. (Note that we obtain the same type of result by choosing S-Net-D since this is a special case of S-Net-M.)

3.1.1. The case with noise-free observations and one initial state

(a) Simulated observation data

We use Algorithm 2 to generate the (synthetic) observations which are sampled at times Eq (3.2) based on the following initial state:

$$u_0(x) = \begin{cases} 1.0, & \text{if } x \in [0,3] \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

The distribution of observation data is shown in Figure 5 (top) where right plot is a zoomed in version of the left plot. This plot shows that essentially the whole interval $u \in [0,1]$ is represented in the data suggesting that a good learning of $f(u)$ in this interval is possible.

(b) Training and testing

By applying Algorithm 3 (ConsLaw-Net), we obtain after training a flux function which we denote by $f_{\theta^*}(u)$. The analytical expression of it is given in Table 1. Apparently, $f_{\theta^*}(u)$ differs from the true one. However, we recall that what matters is the derivative $f'_{\theta^*}(u)$. In order to compare with the true flux function, we plot the translated function $f_{\theta^*}(u) - f_{\theta^*}(0)$ (revised) in Figure 5 (bottom). Clearly, ConsLaw-Net has the ability to identify the true flux function with good accuracy for the flux $f(u;\beta = 10)$. This may not be a surprise since the nonlinearity is somewhat "weak" for this case.

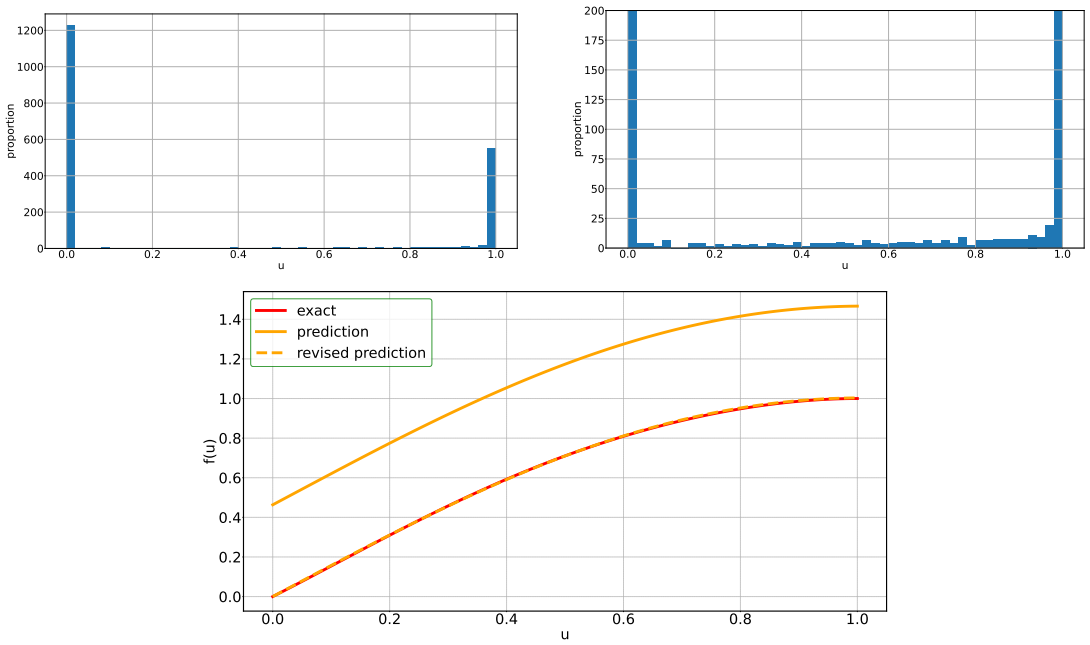


Figure 5. Top. The distribution of noise-free observations generated from combining $f(u;\beta = 10)$ with initial state Eq (3.3). Left: The distribution of all observation data. Right: The distribution between 0 and 200 (number of times values occur in the data). **Bottom**. The flux function $f(u;\beta = 10)$ (red solid line), $f_{\theta^*}(u)$ generated by ConsLaw-Net (orange solid line), and the translated function $f_{\theta^*}(u) - f_{\theta^*}(0)$ (orange dashed line).

Table 1. The identification of flux function $f(u;\beta = 10)$.

Correct $f(u)$	$f(u) = \frac{1}{2}u(3-u^2) + \frac{10}{12}u^2\left(\frac{3}{4} - 2u + \frac{3}{2}u^2 - \frac{1}{4}u^4\right)$
$f_{\theta^*}(u)$ generated by Algorithm 3	$f_{\theta^*}(u) = (1.5727)u + (-0.8902)u^3 + (0.4633) + (0.2512)u^4$ $+ (0.0779)u^2 + (-0.0085)u^5 + (0.0001)u^6$

3.1.2. What is the effect of adding noise to observation data?

(a) Simulated noisy observation data

To test the robustness of ConsLaw-Net, we add 5% noise on the data U generated by the initial state Eq (3.3) based on sampling times Eq (3.2). That is, we replace U by $U+\epsilon$, where $\epsilon \in [-0.05, +0.05]$ and ϵ is generated from a uniform distribution. Since U varies within $[0, 1]$ we refer to this as 5% noise. The distribution of observations is similar to the one shown in Figure 5 (not shown). Specifically, Figure 6 shows a comparison of noise-free and noisy data at three time points: 0.3, 0.6 and 0.9.

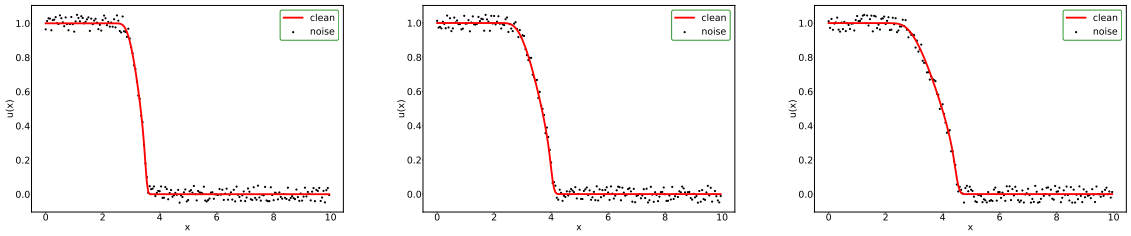
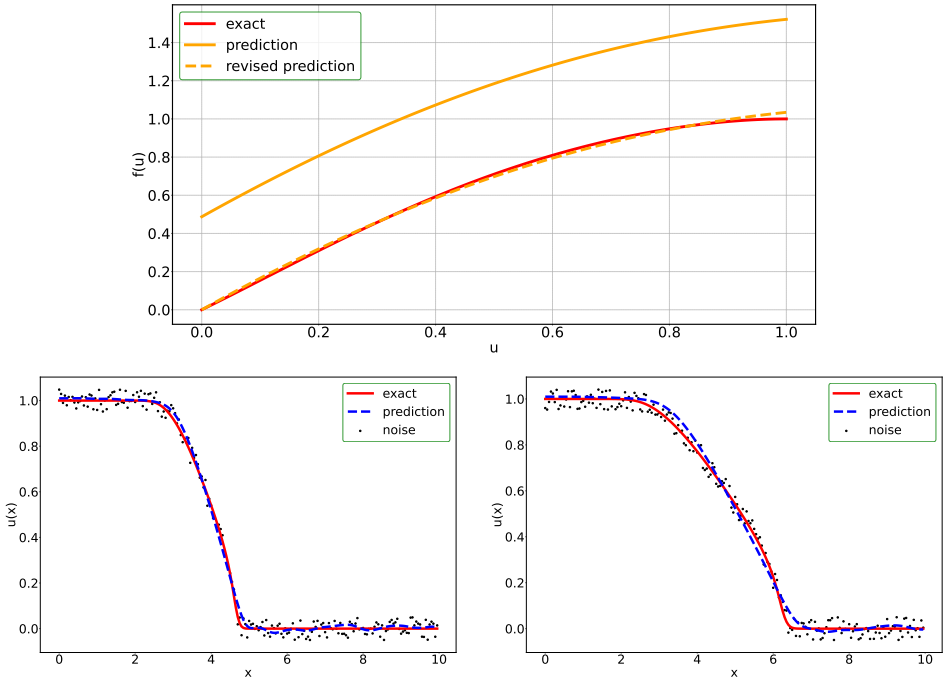


Figure 6. Noise-free and noisy data generated by the initial state (3.3) combined with the flux function $f(u;\beta = 10)$ at three time points: 0.3 (left), 0.6 (middle) and 0.9 (right).



(a) $t = 1.0$ **(b)** $t = 2.0$

Figure 7. Top. The true flux function $f(u;\beta = 10)$ (red solid line), $f_{\theta^*}(u)$ generated by ConsLaw-Net based on noisy data (orange solid line), and the revised function $f_{\theta^*}(u) - f_{\theta^*}(0)$ (orange dashed line). **Bottom**. Predicted solution $u(x,t)$ by using Algorithm 2 based on, respectively, true $f(u;\beta = 10)$ (red solid line) and the learned $f_{\theta^*}(u)$ with noisy data (blude dashed line). (a) Solutions at $t = 1.0$. (b) Solutions at $t = 2.0$.

(b) Training and testing

In Figure 7 (top), we show the learned function $f_{\theta^*}(u)$ generated by ConsLaw-Net as well at the translated $f_{\theta^*}(u) - f_{\theta^*}(0)$ (revised). Comparison with the true $f(u;\beta = 10)$ reveals that the noisy data has made the identification slightly less accurate. Figure 7 (bottom) presents a comparison of the

solution based on $f(u;\beta = 10)$ and the predicted solution at later times $t = 1.0$, $t = 2.0$ by using Algorithm 2 combined with $f_{\theta^*}(u)$. Noise has been added to the initial data Eq (3.3) and then used with the learned $f_{\theta^*}(u)$ as input to Algorithm 2. This gives rise to the blue dashed line which contains some smaller oscillations due to noisy initial data. From Figure 7 we see that the noisy data combined with just one initial state Eq (3.3) leads to some loss of the predictive ability of ConsLaw-Net. Next, we test how the learning can be improved for the case with noisy data by adding more initial data, thereby, more observation data.

3.1.3. Can we improve the learning when observations are noisy by using 3 initial states?

(a) Simulated noisy observation data

We use Algorithm 2 to generate observations based on the 3 initial states given in Table 2. We have no preferences other than that we want to generate observation data over a broader spectrum by selecting box-functions of different heights as initial states. We consider observation data at times Eq (3.2) and add 5% noise. The distribution of the resulting observation data is shown in Figure 8 (top). Compared to the distribution corresponding to initial data Eq (3.3) and shown in Figure 5, we see that all values of u in $[0, 1]$ are to a larger extent represented.

Table 2. Three initial states used for case with $f(u;\beta = 10)$.

$u^0_{\beta=10} = \begin{cases} 1.0, & \text{if } x \in [2,6] \\ 0, & \text{otherwise} \end{cases}$	$u^1_{\beta=10} = \begin{cases} 0.6, & \text{if } x \in [3,6] \\ 0, & \text{otherwise} \end{cases}$
$u^2_{\beta=10} = \begin{cases} 0.25, & \text{if } x \in [2.5,5.5] \\ 0, & \text{otherwise} \end{cases}$	

(b) Training and testing

Table 3 shows the analytical expression of the trained $f_{\theta^*}(u)$ obtained by ConsLaw-Net. In Figure 8 (bottom) we see from the plot of $f_{\theta^*}(u) - f_{\theta^*}(0)$ (revised) that the increased observation data set resolves the problem with loss of accuracy due to noisy data.

Table 3. Identification of $f(u;\beta = 10)$ based on noisy data from set of inital data in Table 2.

Correct $f(u)$	$f(u) = \frac{1}{2}u(3-u^2) + \frac{10}{12}u^2\left(\frac{3}{4} - 2u + \frac{3}{2}u^2 - \frac{1}{4}u^4\right)$
$f_{\theta^*}(u)$ generated by Algorithm 3	$f(u) = (1.5457)u + (-0.9621)u^3 + (0.3557) + (0.24967)u^4$ $+ (0.1620)u^2 + (-0.0049)u^5 + (3.2293e-05)u^6$

3.2. Example with $\beta = 120$ in Eq (3.1)

We consider now the situation when synthetic data is generated from Eq (3.1) with $\beta = 120$. From Figure 4 it is clear that the shape is more complex. We use the same S-Net-M as for the previous example to represent $f_\theta(u)$, i.e., three hidden layers and 23 trainable parameters.

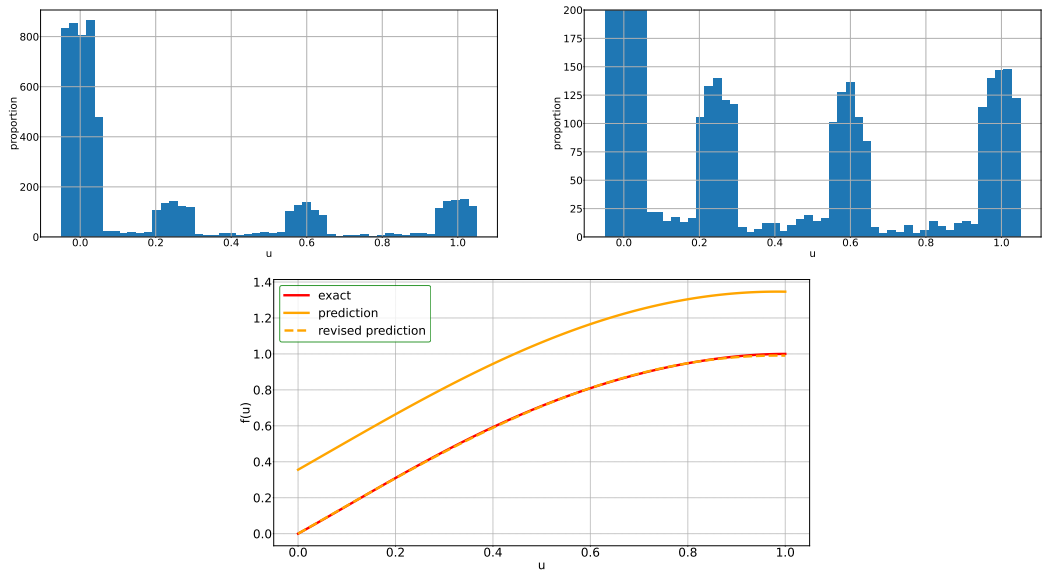


Figure 8. Top. The distribution of noisy observations generated by combining $f(u;\beta = 10)$ with initial states given in Table 3. Left: all observation data. Right: The distribution between 0 and 200. **Bottom**. True flux function $f(u;\beta = 10)$ (red solid line), $f_{\theta^*}(u)$ generated from ConsLaw-Net based on noisy data (orange solid line) and revised function $f_{\theta^*}(u) - f_{\theta^*}(0)$ (orange dashed line).

3.2.1. Noise-free observation data and one initial state

(a) Simulated observation data

We use Algorithm 2 to generate observations based on the following initial state

$$u_0(x) = \begin{cases} 1.0, & \text{if } x \in [2,4] \\ 0, & \text{otherwise} \end{cases} \tag{3.4}$$

Observations are extracted at times Eq (3.2). The distribution of observations is shown in Figure 9.

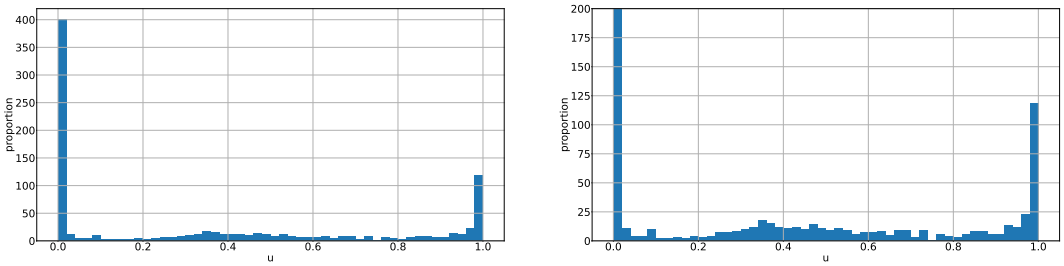
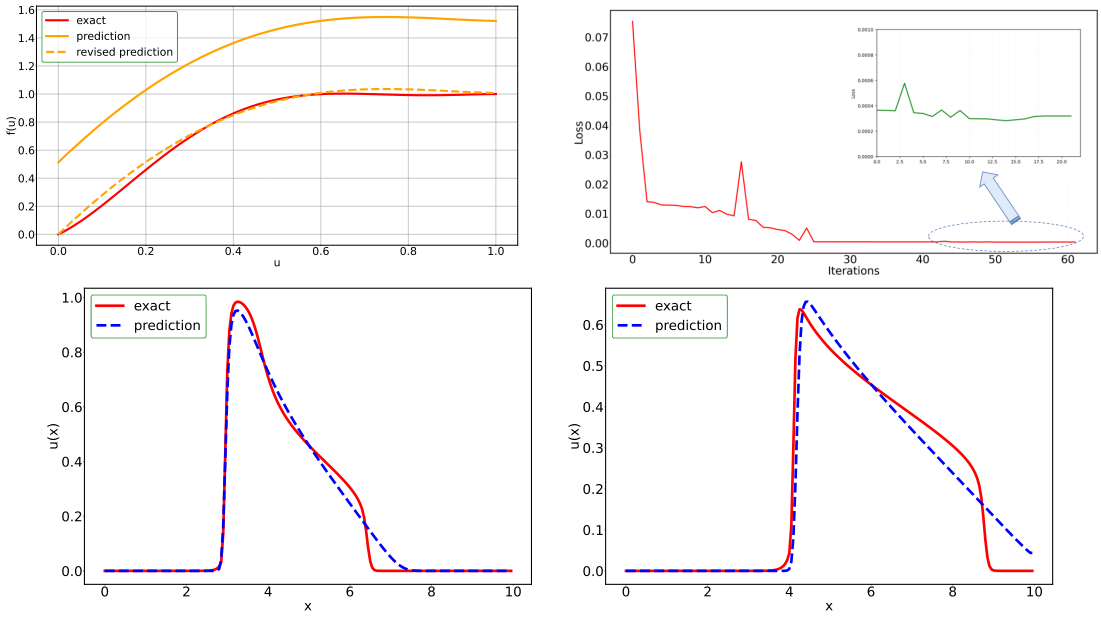


Figure 9. The distribution of clean observations generated from combining $f(u;\beta = 120)$ with initial state Eq (3.4). Left: The distribution of all observation data. Right: The distribution between 0 and 200.

(b) Training and testing

In Figure 10 (top, left), the function $f_{\theta^*}(u)$ obtained from application of ConsLaw-Net and its translated version $f_{\theta^*}(u) - f_{\theta^*}(0)$ is shown and compared to the true $f(u;\beta = 120)$. Interestingly, we see that $f_{\theta^*}(u)$ is essentially a good approximation of the *upper concave envelope* of $f(u;\beta = 120)$, which is the function involved in the construction of the entropy solution associated with the initial discontinuity of Eq (3.4) located at $x = 4$ [15, 30]. The initial jump at $x = 2$, on the other hand, relies on the lower convex envelope which amounts to the straight line which connects $(0, 0)$ and $(1, 1)$ and reveals no information about $f(u)$ for $u \in (0, 1)$.



(a) $t = 1.0$ **(b)** $t = 2.0$

Figure 10. Top. Left. Noise-free observations are generated from $f(u;\beta = 120)$ (red solid line) combined with the initial Eq (3.4). The learned flux $f_{\theta^*}(u)$ generated from ConsLaw-Net (orange solid line) and the revised function (orange dashed line). Right. Loss function behavior shows that error goes to zero. **Bottom**. Comparison of prediced behavior based on, respectively, true $f(u;\beta = 120)$ and learned $f_{\theta^*}(u)$, at times $t = 1.0$ (a), $t = 2.0$ (b).

In Figure 10 (top, right), we show that the loss function tends to zero, which reflects that the (wrongly) identified flux function $f_{\theta^*}(u)$ is largely consistent with the observation data. This brings to the surface a main challenge with learning the unknown flux function, namely, the lack of one-to-one correspondence between observation data and nonlinear flux function, as expressed by the entropy condition Eq (2.4). The consequence of this poor approximation to the true $f(u;\beta = 120)$ is illustrated in Figure 10 (bottom), which presents a comparison of the exact analytical solution and the solution predicted by $f_{\theta^*}(u)$ at times $t = 1.0$, $t = 2.0$ based on the initial state Eq (3.4). From Figure 9 we see

that the observation data is sparse for $u \in (0.0, 0.2)$ and for u centered around 0.8. So we may try to collect more data from these intervals by adding another type of initidal data.

3.2.2. Can learning of $f(u;\beta = 120)$ be improved by using 2 initial states?

(a) Simulated observation data

We use Algorithm 2 to generate new observations based on the two initial states given in Table 4.

Table 4. Two initial states for case with $f(u;\beta = 120)$.

$u^0_{\beta=120} = \begin{cases} 1.0, & \text{if } x \in [2,4] \\ 0, & \text{otherwise} \end{cases}$	$u^1_{\beta=120} = \begin{cases} 0.9, & \text{if } x \in [3.5, 6.5] \\ 0, & \text{otherwise} \end{cases}$

The distribution of the resulting observation data is shown in Figure 11 (top). Clearly, the part of the interval $(0, 1)$ which was poorly represented with only one initial state, is now present to a larger extent.

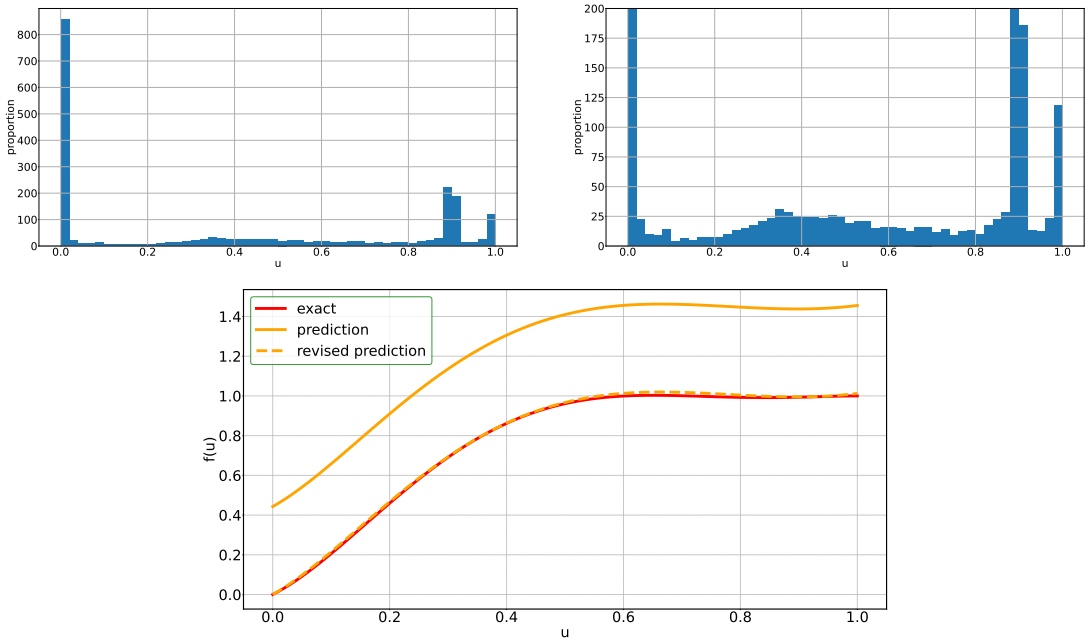


Figure 11. Top. The distribution of noise-free observations generated from combining $f(u;\beta = 120)$ with initial state given in Table 4. Left: The distribution of all observation data. Right: The distribution between 0 and 200. **Bottom**. The flux function $f(u;\beta = 120)$ where noise-free observations have been generated from initial data given in Table 4. The exact flux function $f(u;\beta = 120)$ (red solid line), $f_{\theta^*}(u)$ generated from ConsLaw-Net (orange solid line), and $f_{\theta^*}(u) - f_{\theta^*}(0)$ (orange dashed line).

(b) Training and testing

The analytical expression of the trained flux function $f_{\theta^*}(u)$ is given in Table 5. From the visualization in Figure 11 (bottom), we see that the learned $f_{\theta^*}(u)$ is largely consistent with the exact $f(u;\beta = 120)$ with only a small inaccuracy seen in the interval $u \in [0.6, 0.8]$.

Table 5. The identification of $f(u;\beta = 120)$ based on noise-free data generated by initial data in Table 4.

Correct $f(u)$	$f(u) = \frac{1}{2}u(3-u^2) + \frac{120}{12}u^2\left(\frac{3}{4} - 2u + \frac{3}{2}u^2 - \frac{1}{4}u^4\right)$
$f_{\theta^*}(u)$ generated by ConsLaw-Net	$f(u) = (21.7963)u^4 + (-21.1055)u^3 + (-10.3012)u^5$ $+ (6.7875)u^2 + (2.4417)u^6 + (1.6638)u + (0.4427)$ $+ (-0.2827)u^7 + (0.0127)u^8$

Finally, we also want to test the effect of noisy data for this case. We add 5% noise on data generated by the initial state in Table 4. Table 6 shows the analytic expression of $f_{\theta^*}(u)$ obtained by ConsLaw-Net. The corresponding visualization in Figure 12 reflects that the noise hides for the shape of the true flux function, similarly as for the case with less observation data seen in Figure 10.

Table 6. Identification of $f(u;\beta = 120)$ based on noisy data generated from initial state in Table 4.

Correct $f(u)$	$f(u) = \frac{1}{2}u(3-u^2) + \frac{120}{12}u^2\left(\frac{3}{4} - 2u + \frac{3}{2}u^2 - \frac{1}{4}u^4\right)$
$f(u)$ generated by ConsLaw-Net	$f(u) = (-4.9213)u^3 + (2.3761)u + (1.7820)u^4 + (1.0587)u^2$ $+ (0.6508)u^5 + (0.5101) + (0.0653)u^6 + (0.0027)u^7$ $+ (3.8506e-05)u^8$

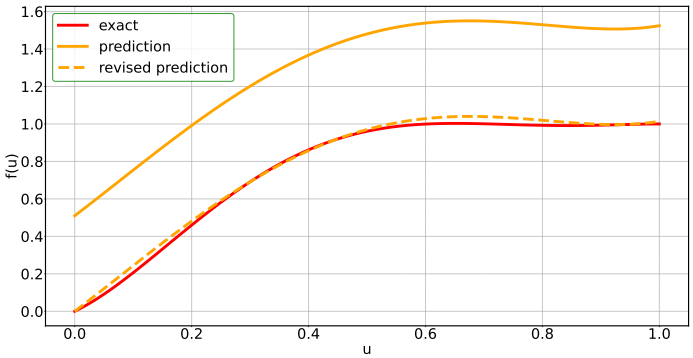


Figure 12. The flux function $f(u;\beta = 120)$ where noisy observations have been generated from initial data given in Table 4. The exact flux function $f(u;\beta = 120)$ (red solid line), $f_{\theta^*}(u)$ generated from ConsLaw-Net (orange solid line), and $f_{\theta^*}(u) - f_{\theta^*}(0)$ (orange dashed line).

A natural remedy is to collect more observations by adding a wider spectrum of initial states, as shown in Table 7.

Table 7. Initial states for the case with $f(u;\beta = 120)$ and noisy data

$u^0_{\beta=120} = \begin{cases} 1.0, & \text{if } x \in [2,4] \\ 0, & \text{otherwise} \end{cases}$	$u^1_{\beta=120} = \begin{cases} 0.9, & \text{if } x \in [3.5,6.5] \\ 0, & \text{otherwise} \end{cases}$
$u^2_{\beta=120} = \begin{cases} 0.8, & \text{if } x \in [3,6] \\ 0, & \text{otherwise} \end{cases}$	$u^3_{\beta=120} = \begin{cases} 0.7, & \text{if } x \in [1,4] \\ 0, & \text{otherwise} \end{cases}$

The corresponding histogram showing the distribution of different values of $u \in (0,1)$ is found in Figure 13 (top). Clearly, the additional initial states has increased the involvement of u values in the whole interval $(0,1)$, suggesting that a better learning can be expected.

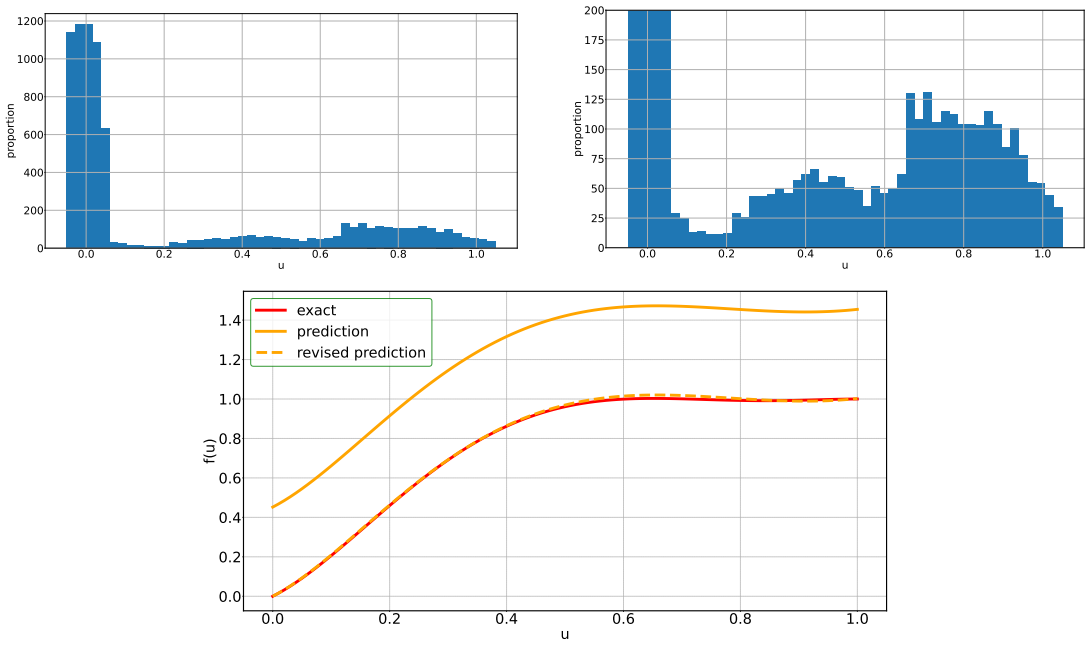


Figure 13. Top. The distribution of noisy observations generated by the four initial states in Table 7 by using $f(u;\beta = 120)$. **Bottom**. The flux function $f(u;\beta = 120)$ where noisy observations are generated based on initial data given in Table 7. The exact flux function $f(u;\beta = 120)$ (red solid line), $f_{\theta^*}(u)$ generated from ConsLaw-Net (orange solid line), and $f_{\theta^*}(u) - f_{\theta^*}(0)$ (orange dashed line).

Table 8. The identification of $f(u;\beta = 120)$ based on noisy data and set of initial states given in Table 7.

Correct $f(u)$	$f(u) = \frac{1}{2}u(3-u^2) + \frac{120}{12}u^2\left(\frac{3}{4} - 2u + \frac{3}{2}u^2 - \frac{1}{4}u^4\right)$
$f(u)$ generated by ConsLaw-Net	$f = (24.6292)u^4 + (-23.4251)u^3 + (-12.1104)u^5$ $+ (7.7098)u^2 + (3.0216)u^6 + (1.5297)u + (0.4520)$ $+ (-0.3702)u^7 + (0.0177)u^8$

Table 8 shows the analytic expression of $f_{\theta*}(u)$ obtained from ConsLaw-Net. Figure 13 (bottom) confirms that the learning of the true $f(u;\beta = 120)$ is quite effective now despite noisy data.

3.3. Example with $f(u;\beta = 200)$ and $f(u;\beta = 300)$ defined by Eq (3.1)

We consider the situation when synthetic data is generated from Eq (3.1) with $\beta = 200$. From Figure 4 it is clear that the shape involves two inflection points. First a convex region for small u, followed by a concave for intermediate u, and then a convex region again for large u. We use the same S-Net-M as for the previous example to represent $f_\theta(u)$, i.e., three hidden layers and 23 trainable parameters. Also the times for observation data is given by Eq (3.2).

3.3.1. Noise-free observations and two initial states

(a) Simulated observation data, training and testing

We use Algorithm 2 to generate the observations based on two initial states as specified in Table 9. The corresponding histogram is shown in Figure 14 (top) and suggests a fair chance to achieve good learning result. The result of the learning is illustrated in Figure 14 (bottom). We see that to a large extent the learned $f_{\theta^*}(u)$ fits well with the true $f(u;\theta = 200)$ with room for improvements in the intervals $(0.4, 0.6)$ and $(0.8, 1.0)$.

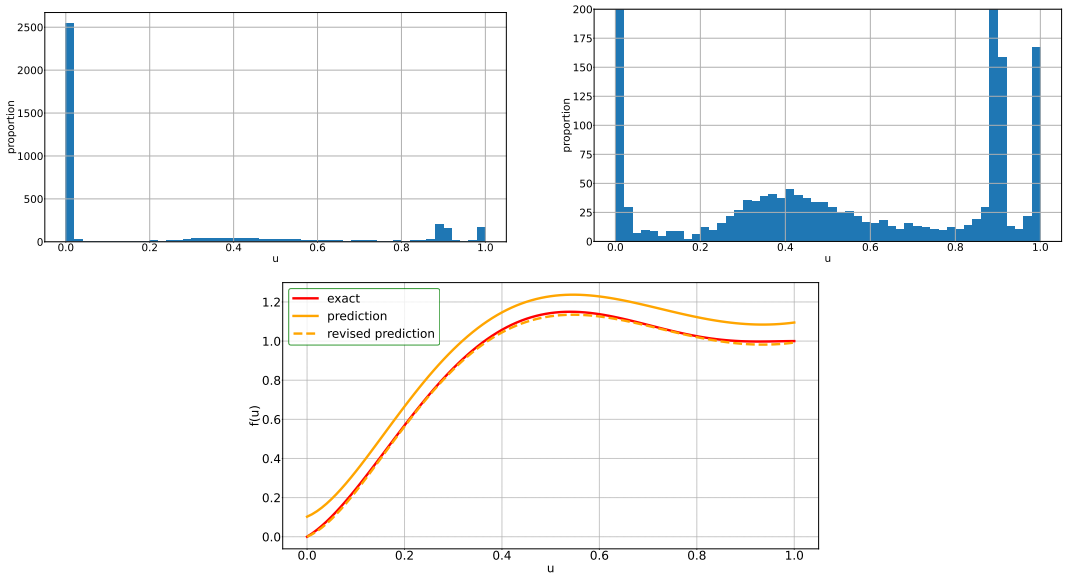


Figure 14. Top. The distribution of noise-free observations generated for the case with $f(u;\beta = 200)$ and initial data as in Table 9. Left. Distribution of all data. Right. Distribution of the quantity between 0 and 200. **Bottom**. The flux function $f(u;\beta = 200)$ where noise-free observations are generated based on initial data given in Table 9. The exact flux function $f(u;\beta = 200)$ (red solid line), $f_{\theta^*}(u)$ generated from ConsLaw-Net (orange solid line), and $f_{\theta^*}(u) - f_{\theta^*}(0)$ (orange dashed line).

Table 9. Two initial states used for the case with $f(u;\beta = 200)$.

$u^0_{\beta=200} = \begin{cases} 1.0, & \text{if } x \in [2,4] \\ 0, & \text{otherwise} \end{cases}$	$u^0_{\beta=200} = \begin{cases} 0.9, & \text{if } x \in [3.5,6.5] \\ 0, & \text{otherwise} \end{cases}$

Now we focus on the case with true flux function $f(u;\beta = 300)$. As seen from Figure 4, the shape bears clear similarity to the case with $f(u;\beta = 200)$. However, the convex and concave regions are more pronounced. Hence, in light of the result for $f(u;\beta = 200)$ we may expect that more observation data is required. Therefore, we consider six initial data as given in Table 10.

Table 10. Six initial states with $\beta = 300$.

$u^0_{\beta=300} = \begin{cases} 1.0, & \text{if } x \in [0,3] \\ 0, & \text{otherwise} \end{cases}$	$u^1_{\beta=300} = \begin{cases} 0.8, & \text{if } x \in [3,6] \\ 0, & \text{otherwise} \end{cases}$
$u^2_{\beta=300} = \begin{cases} 0.6, & \text{if } x \in [4,7] \\ 0, & \text{otherwise} \end{cases}$	$u^3_{\beta=300} = \begin{cases} 0.4, & \text{if } x \in [0,3] \\ 0, & \text{otherwise} \end{cases}$
$u^4_{\beta=300} = \begin{cases} 0.3, & \text{if } x \in [3,6] \\ 0, & \text{otherwise} \end{cases}$	$u^5_{\beta=300} = \begin{cases} 0.7, & \text{if } x \in [1,4] \\ 0, & \text{otherwise} \end{cases}$

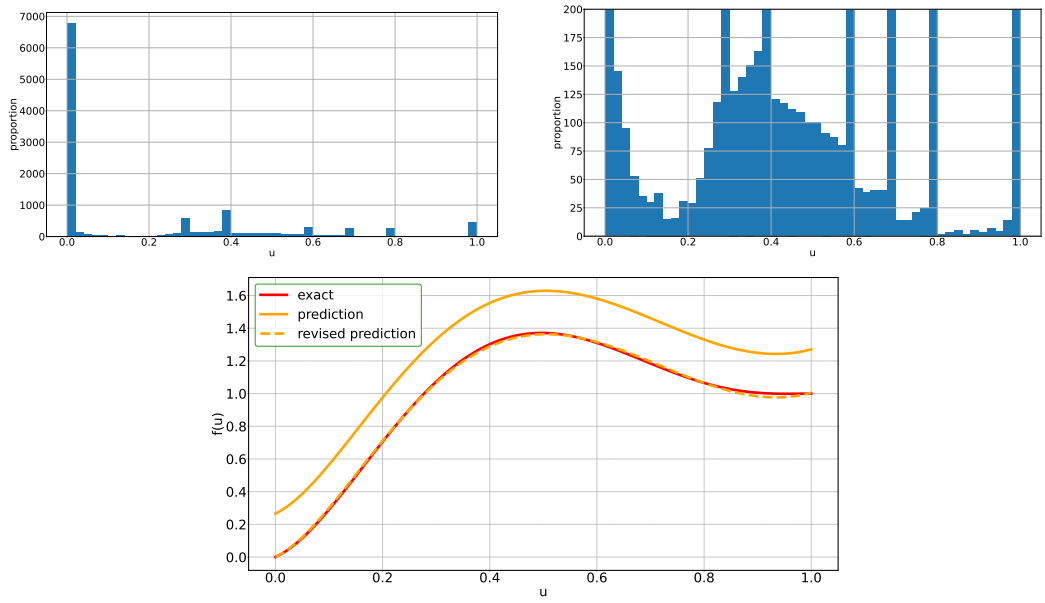


Figure 15. Top. The distribution of noise-free observations generated for the case with $f(u;\beta = 300)$ and initial data as in Table 10. Left. Distribution of all data. Right. Distribution of the quantity between 0 and 200. **Bottom**. The flux function $f(u;\beta = 300)$ where noise-free observations are generated based on initial data given in Table 10. The exact flux function $f(u;\beta = 300)$ (red solid line), $f_{\theta^*}(u)$ generated from ConsLaw-Net (orange solid line), and $f_{\theta^*}(u) - f_{\theta^*}(0)$ (orange dashed line).

The distribution of observation data is shown in Figure 15 (top). It reflects a good distribution apart from a somewhat low representation for $u \in (0.8, 1.0)$. The analytical expression of $f_{\theta^*}(u)$ generated by ConsLaw-Net is given in Table 11. The visualization in Figure 15 (bottom) shows that the learned $f_{\theta^*}(u)$ largely captures the variations of the true flux function $f(u;\beta = 300)$, with a small loss of accuracy in the interval with sparse observation data ($u \in (0.8, 1.0)$).

Table 11. Identification of $f(u;\beta = 300)$ based on clean data generated by 6 initial states (Table 10).

Correct $f(u)$	$f(u) = \frac{1}{2}u(3-u^2) + \frac{300}{12}u^2\left(\frac{3}{4} - 2u + \frac{3}{2}u^2 - \frac{1}{4}u^4\right)$
$f(u)$ generated by ConsLaw-Net	$f(u) = (64.5698)u^4 + (-59.0164)u^3 + (-32.7687)u^5$ $+ (19.2141)u^2 + (8.4140)u^6 + (1.6037)u + (-1.0640)u^7$ $+ (0.2655) + (0.05272)u^8$

3.4. Example with $\beta = -200$ in Eq (3.1)

In this example we explore how ConsLaw-Net can learn the flux function $f(u;\beta = -200)$. As seen from Figure 4 this flux function also has two inflection points. The order of the convex and concave regions is interchanged, as compared to the case with $f(u;\beta = 300)$, where a convex region for intermediate u-values now is surrounded by a concave region for small u and large u. As before, we use S-Net-M to represent $f_\theta(u)$, however, we use four hidden layers which amounts to 34 trainable parameters.

3.4.1. Noise-free observation data and two initial states

(a) Simulated observation data

We use Algorithm 2 to generate the observations based on the two initial states given in Table 12. We generate observation data at times given by Eq (3.2), in addition to the times 1.0 and 1.1. This gives rise to the distribution of observation data as shown in Figure 16 (top). Clearly, the observation data covers the whole interval $(0, 1)$ well.

Table 12. Two initial states for the case with $f(u;\beta = -200)$.

$u^0_{\beta=-200} = \begin{cases} 1.0, & \text{if } x \in [2,4] \\ 0, & \text{otherwise} \end{cases}$	$u^1_{\beta=-200} = \begin{cases} 0.9, & \text{if } x \in [3.5,6.5] \\ 0, & \text{otherwise} \end{cases}$

(b) Training and testing

From Figure 16 (bottom), we see that the learned flux $f_{\theta^*}(u)$ essentially captures the lower convex envelope of the true $f(u;\beta = -200)$. In particular, there is a lack of information about the true flux for $u \in [0.0, 0.3]$ and $u \in [0.6, 1.0]$. This can be understood in light of the fact that the decreasing initial discontinuity located at $x = 4$ and $x = 6.5$, respectively, depends on the upper concave evelope, which

essentially is the straight line which connects $(0, 0)$ and $(0.9, f(0.9))$. Hence, precise information about the shape of $f(u;\beta = -200)$ is difficult to reveal. As a result, the initial increasing jumps at $x = 2$ and $x = 3.5$, respectively, then imply that ConsLaw-Net generates a function $f_{\theta^*}(u)$ which coincides with the lower convex envelope of $f(u;\beta = -200)$, as shown in Figure 16 (bottom).

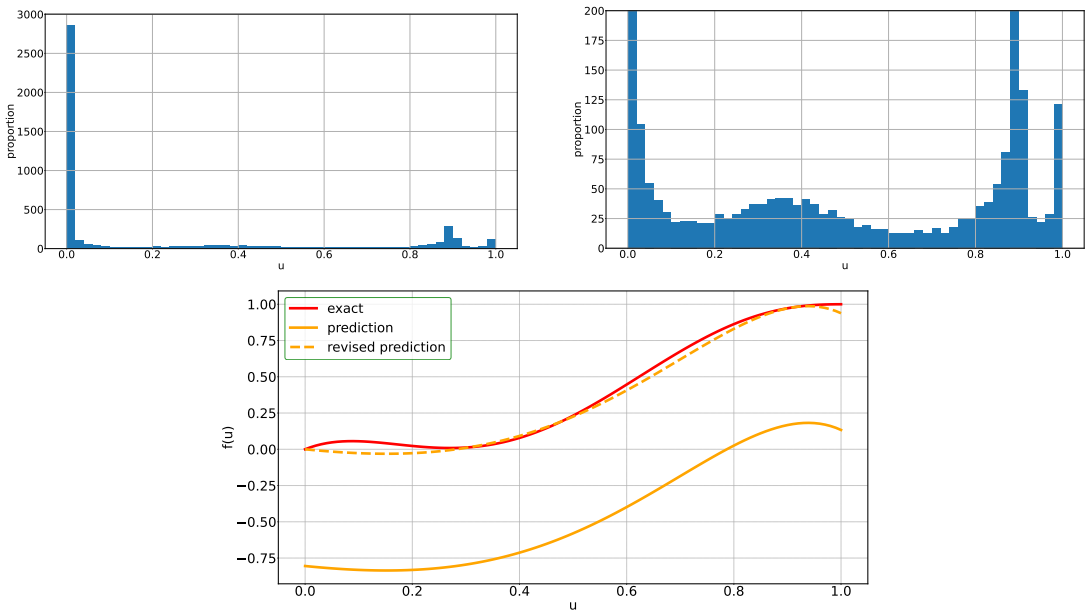


Figure 16. Top. The distribution of noise-free observations generated for the case with $f(u;\beta = -200)$ and initial data as in Table 12. Left. Distribution of all data. Right. Distribution of the quantity between 0 and 200. **Bottom**. The flux function $f(u;\beta = -200)$ where noise-free observations are generated based on initial data given in Table 12. The exact flux function $f(u;\beta = -200)$ (red solid line), $f_{\theta^*}(u)$ generated from ConsLaw-Net (orange solid line), and $f_{\theta^*}(u) - f_{\theta^*}(0)$ (orange dashed line).

3.4.2. Improving the learning of $f(u;\beta = -200)$ by using six initial states

(a) Simulated observation data

We increase the number of initial data from two to six, as shown in Table 13. We use Algorithm 2 to generate the observations based on initial data in Table 13. The corresponding histogram of observation data is shown in Figure 17 (top).

(b) Training and testing

We first illustrate in Table 14 the analytical expression of the learned flux function $f_{\theta^*}(u)$ obtained through ConsLaw-Net. From Figure 17 (bottom) we see that $f_{\theta^*}(u)$ is consistent with the exact $f(u;\beta = -200)$ with respect to u in most intervals, except for the interval $u \in [0.8, 1.0]$. Combined with the

Table 13. Six initial states used in combination with $f(u;\beta = -200)$.

$u^0_{\beta=-200} = \begin{cases} 1.0, & \text{if } x \in [0,3] \\ 0, & \text{otherwise} \end{cases}$	$u^1_{\beta=-200} = \begin{cases} 0.8, & \text{if } x \in [3,6] \\ 0, & \text{otherwise} \end{cases}$
$u^2_{\beta=-200} = \begin{cases} 0.6, & \text{if } x \in [4,7] \\ 0, & \text{otherwise} \end{cases}$	$u^3_{\beta=-200} = \begin{cases} 0.4, & \text{if } x \in [0,3] \\ 0, & \text{otherwise} \end{cases}$
$u^4_{\beta=-200} = \begin{cases} 0.3, & \text{if } x \in [3,6] \\ 0, & \text{otherwise} \end{cases}$	$u^5_{\beta=-200} = \begin{cases} 0.7, & \text{if } x \in [1,4] \\ 0, & \text{otherwise} \end{cases}$

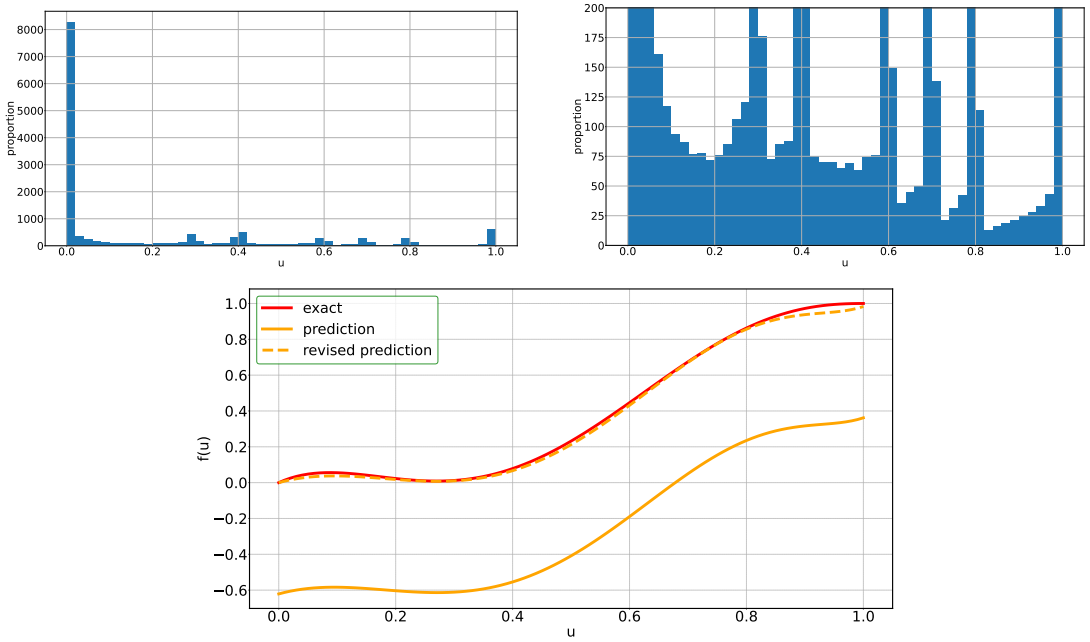


Figure 17. Top. The distribution of noise-free observations generated for the case with $f(u;\beta = -200)$ and initial data as in Table 13. Left. Distribution of all data. Right. Distribution of the quantity between 0 and 200. **Bottom**. The flux function $f(u;\beta = -200)$ where noise-free observations are generated based on initial data given in Table 13. The exact flux function $f(u;\beta = -200)$ (red solid line), $f_{\theta^*}(u)$ generated from ConsLaw-Net (orange solid line), and $f_{\theta^*}(u) - f_{\theta^*}(0)$ (orange dashed line).

observation distribution in Figure 17 (top), we see that there are relatively few observations in this interval, which most likely is the reason for this loss in accurate learning.

4. Learning of fractional flux functions

In this section, we consider a flux function in the following fractional form Eq (4.1)

$$f(u) = \frac{u^2}{u^2 + \beta(1-u)^2}. \tag{4.1}$$

Table 14. Identification of $f(u;\beta = -200)$ based on clean data generated by initial states in Table 13.

Correct $f(u)$	$f(u) = \frac{1}{2}u(3-u^2) + \frac{-200}{12}u^2\left(\frac{3}{4} - 2u + \frac{3}{2}u^2 - \frac{1}{4}u^4\right)$
$f(u)$ generated by ConsLaw-Net	$f(u) = (10.8706)u^3 - 10.7580u^5 - 6.3577u^2 + (5.9855)u^4$ $- 4.9812u^6 + (4.0332)u^7 + (0.8962)u + (0.8686)u^8$ $- 0.6212 + (0.5814)u^{10} - 0.4223u^9 + (0.3570)u^{11}$ $- 0.0937u^{12} - 0.0094u^{15} + (0.0082)u^{13} + (0.0067)u^{14}$ $- 0.0021u^{16}$

This nonlinear flux function appears in the context of creeping two-phase porous media flow [15] and accounts for a large range of nonlinear two-phase behavior. We consider the same spatial domain as before ($L = 10$) and explore solution behavior in the time period $t \in [0, T]$ with $T = 2$. We set $\beta = 0.5$ in Eq (4.1) when we generate synthetic data for further testing of ConsLaw-Net for this class of flux functions. We use a grid of $N_x = 400$ cells and consider observation data Eq (2.9) at times Eq (3.2).

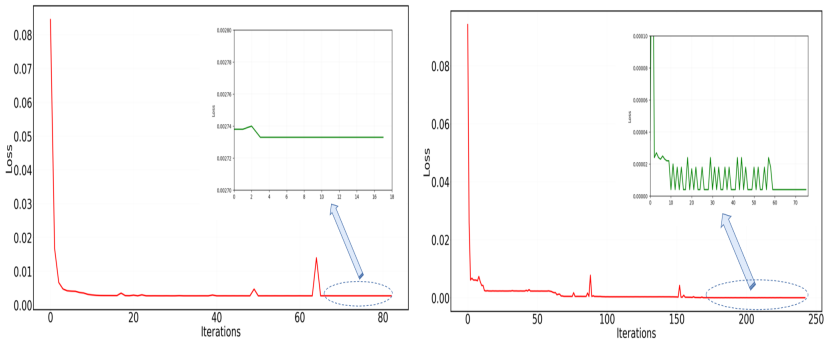


Figure 18. Left. Loss function based on S-Net-M. Right. Loss function based on S-Net-D.

4.1. Example with $\beta = 1/2$ in Eq (4.1)

We first try to use S-Net-M with three hidden layers and 23 trainable parameters, based on previous experience from Section 3. However, we find that the loss function does not converge to zero, see Figure 18 (left). Therefore, we replace it by S-Net-D (2.20) with two hidden layers and 18 trainable parameters which gives significantly better behavior in terms of convergence behavior of the loss function, see Figure 18 (right).

4.1.1. What is the effect of noisy observations?

(a) Simulation of observation data, training, and testing

We use Algorithm 2 to generate the observations based on initial state in Table 15. Then, 3% noise is added to the observations. The distribution of the resulting observation data is shown in Figure 19

(top). Specifically, Figure 20 shows clean and noisy data corresponding to the first initial data in Table 15 at three time points: 0.3, 0.6 and 0.9.

Table 15. Two initial states with $\beta = 0.5$ in the flux function Eq (4.1).

$u^0_{\beta=0.5} = \begin{cases} 1.0, & \text{if } x \in [4,6] \\ 0, & \text{otherwise} \end{cases}$	$u^1_{\beta=0.5} = \begin{cases} 0.8, & \text{if } x \in [4,6] \\ 0, & \text{otherwise} \end{cases}$

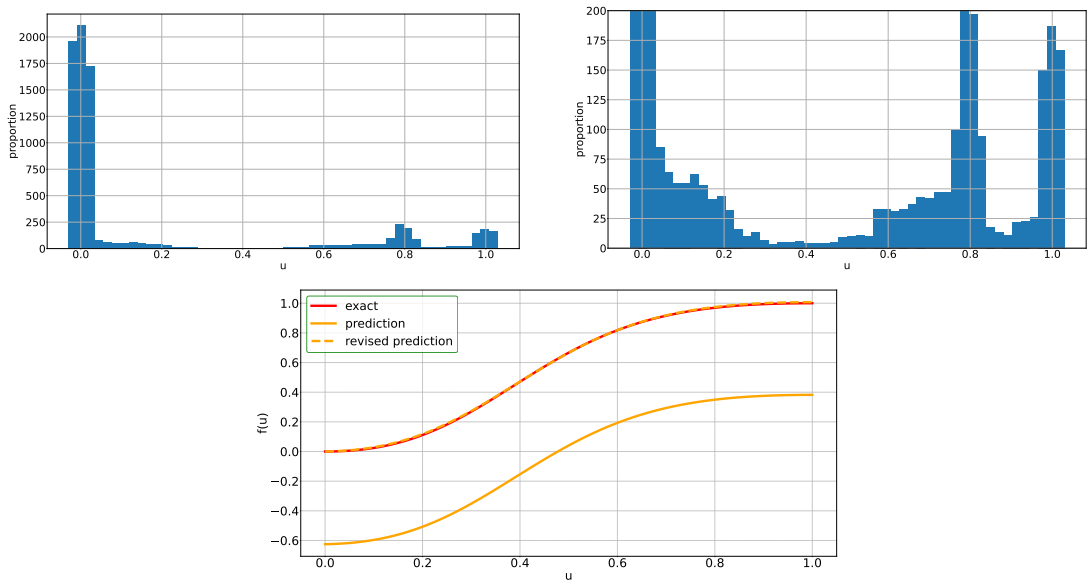


Figure 19. Top. The distribution of noisy observations generated for the flux function given by Eq (4.1) with $\beta = 1/2$ and initial data as in Table 15. Left. Distribution of all data. Right. Distribution of the quantity between 0 and 200. **Bottom**. The true flux function Eq (4.1) with $\beta = 1/2$ where noisy observations are generated based on initial data given in Table 15. The exact flux function (red solid line), $f_{\theta^*}(u)$ generated from ConsLaw-Net (orange solid line), and $f_{\theta^*}(u) - f_{\theta^*}(0)$ (orange dashed line).

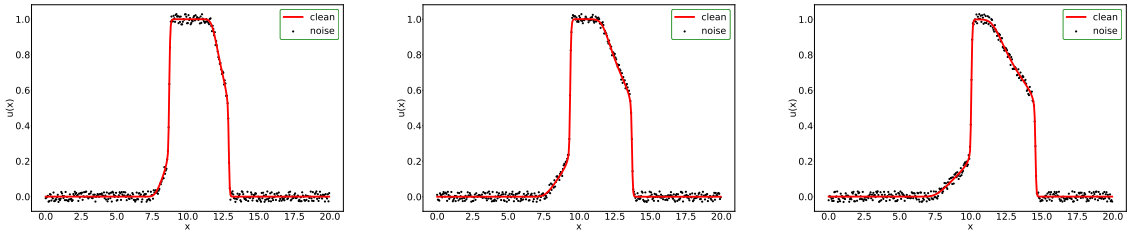


Figure 20. Clean and noisy data generated by the first initial data in Table 15 with $N_x = 400$ at three time points: 0.3 (left), 0.6 (middle) and 0.9 (right).

Table 16 shows the analytic expression of the identified $f_{\theta^*}(u)$. In Figure 19 (bottom), we see a comparison of the learned flux function under noisy data and the true flux function. Clearly, the

learning has been effective for this flux function under noisy data based on ConsLaw-Net combined with the neural network S-Net-D to represent the unknown flux function.

Table 16. The identification of $f(u)$ with $\beta = 1/2$ based on noisy data generated by initial data as given in Table 15.

Correct $f(u)$	$f(u) = \frac{u^2}{u^2+0.5(1-u)^2}$
$f(u)$ learned from ConsLaw-Net	$f(u) = \frac{-1.224u + 0.639 - 0.233u^2 - 0.013u^3 - 6.0e-05u^4}{-2.696u^2 + 1.870u - 1.022 - 0.323u^3 - 0.002u^4}$

5. Conclusion

Compared with advanced methods that have been used to learn PDE problems from data [11,13,14], our method can deal with scalar conservation laws and identification of the unknown nonlinear flux function. In this paper, we designed a framework denoted ConsLaw-Net that combines a deep feed-forward network and an entropy satisfying discrete scheme to learn the unknown flux function. We have found that by including observation data from a sufficient number of initial states, the correct nonlinear flux function can be recovered. This is true both for data with and without noise. Using symbolic multilayer neural networks (S-Net-M or S-Net-D) to represent the unknown flux function $f_\theta(u)$ in an entropy satsifying scheme, represents the key components. It has been demonstrated that the additional regularity imposed on the flux function through $f_\theta(u)$ helps to identify the correct form of it. Moreover, the identified flux function $f_{\theta^*}(u)$ has the ability to discover the unknown nonlinear conservation law model from a relatively sparse amount of observation data, e.g., typically sampled at 10 different times. Interesting findings are:

(1) Depending on the complexity of the hidden flux function $f(u)$ we seek to identify, i.e., the nonlinear shape, we may need observation data corresponding to a set of different initial states $u_0(x)$. This is necessary in order to collect information that can reflect the nonlinear form of $f(u)$ for all values of u in the interval for which we seek to learn the flux function. We also explore the impact of noise in the observation data and find that the correct flux function to a large extent can be identified from noisy data by including 4–6 different initial states.

(2) We find that the method can learn the relevant flux function for a whole family of different flux functions ranging from pure convex/concave to strongly non-convex functions. The role of observation data as a result of different sets of initial states, is highlighted.

(3) In this work we apply a variant of the Rusanov scheme [15] which relies on an estimate of the maximum value of $f'(u)$. The simplicity of the numerical scheme is exploited in the learning process. Other numerical schemes may require suitable modifications in the definition of ConsLaw-Net.

The current version of ConsLaw-Net is restricted to a scalar conservation law in one dimension. Possible further extensions can be: (i) What is the role of the specific discrete scheme that approximates the entropy solution? Does the method work for any entropy-satisfying scheme? (ii) Is it possible to also learn the role played by parameters like β in Eq (3.1) and Eq (4.1) from the observation data? In orther words, identify the functional form of f with respect to both u and β. (iii) Explore the proposed

method in a setting where experimental data is available. This may lead to considering other type of observation data than we have used in this work.

Acknowledgments

The authors acknowledge the University of Stavanger to support this research with funds coming from the project Computations of PDEs systems and use of machine learning techniques (IN-12570).

Conflict of interest

The authors declare there is no conflict of interest.

References

1. J. Bongard, H. Lipson, Automated reverse engineering of nonlinear dynamical systems. *Proc. Natl. Acad. Sci.* , **104** (2007), 9943–9948. https://doi.org/10.1073/pnas.0609476104
2. M. Schmidt, H. Lipson, Distilling free-form natural laws from experimental data, *Science*, 324 (2009), 81–85 . https://doi.org/10.1126/science.1165893
3. H. Owhadi, Bayesian numerical homogenization, *Multiscale. Model. Sim.*, **13** (2015), 812–828. https://doi.org/10.1137/140974596
4. M. Raissi, P. Perdikaris,, G.E. Karniadakis, Inferring solutions of differential equations using noisy multi-fidelity data, *J. Comput. Phys.*, 335 (2017), 736–746. https://doi.org/10.1016/j.jcp.2017.07.050
5. M. Raissi, P. Perdikaris, G.E. Karniadakis, Machine learning of linear differential equations using Gaussian processes, *J. Comput. Phys.*, 348 (2017), 683–693. https://doi.org/10.1016/j.jcp.2017.07.050
6. C.E. Rasmussen, C.K. Williams, *Gaussian processes for machine learning*, Cambridge: MIT press, 2006.
7. S.L. Brunton, J.L. Proctor, J.N. Kutz, Discovering governing equations from data by sparse identification of nonlinear dynamical systems, *Proc. Natl. Acad. Sci.*, **113** (2016), 3932–3937. https://doi.org/10.1073/pnas.1517384113
8. H Schaeffer, Learning partial differential equations via data discovery and sparse optimization, *Proc. Math. Phys. Eng. Sci*, **473** (2017), 20160446. https://doi.org/10.1098/rspa.2016.0446
9. S.H. Rudy, S.L. Brunton, J.L. Proctor, J.N. Kutz, Data-driven discovery of partial differential equations, *Sci. Adv.*, **3** (2017), e1602614. https://doi.org/10.1126/sciadv.1602614
10. Z. Wu, R. Zhang, Learning physics by data for the motion of a sphere falling in a non-Newtonian fluid, *Commun Nonlinear Sci Numer Simul*, **67** (2019), 577–593. https://doi.org/10.1016/j.cnsns.2018.05.007
11. M. Raissi, P. Perdikaris, 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), 686–707. https://doi.org/10.1016/j.jcp.2018.10.045

12. O Fuks, H.A. Tchelepi, Limitations of physics informed machine learning for nonlinear two-phase transport in porous media, *J. Mach. Learn. Model. Comput.*, **1** (2020), 19–37. https://doi.org/10.1615/JMachLearnModelComput.2020033905
13. Z. Long, Y. Lu, X. Ma, B. Dong, PDE-net: Learning PDEs from data, *Proceedings of the 35th International Conference on Machine Learning*, **80** (2018), 3208–3216.
14. Z. Long, Y. Lu, B. Dong, PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network, *J. Comput. Phys.*, **399** (2019), 108925. https://doi.org/10.1016/j.jcp.2019.108925
15. R.J. LeVeque, *Finite volume methods for hyperbolic problems*, Cambridge: Cambridge Texts in Applied Mathematics, 2007.
16. H. Holden, N.H. Risebro, *Front tracking for hyperbolic conservation laws*, Berlin: Springer, 2011.
17. G. Martius, C.H. Lampert, *Extrapolation and learning equations*, arXiv: 1610.02995, [Preprint], (2016) [cited 2022 Oct 18]. Available form: https://arxiv.53yu.com/abs/1610.02995
18. S. Sahoo, C. Lampert, G. Martius, Learning equations for extrapolation and control, *Proceedings of the 35th International Conference on Machine Learning*, **80** (2018), 4442–4450.
19. F. James, M. Sepúlveda, Convergence results for the flux identification in a scalar conservation law, *SIAM J. Control. Optim.*, **37** (1999), 869–891. https://doi.org/10.1137/S0363012996272722
20. H. Holden, F.S. Priuli, N.H. Risebro, On an inverse problem for scalar conservation laws, *Inverse Probl*, **30** (2014), 035015. https://doi.org/10.1088/0266-5611/30/3/035015
21. M. C. Bustos, F. Concha, R. Bürger, E.M. Tory, *Sedimentation and thickening–Phenomenological Foundation and Mathematical Theory*. Dordrecht: Kluwer Academic Publishers, 1999.
22. S. Diehl, Estimation of the batch-settling flux function for an ideal suspension from only two experiments, *Chem. Eng. Sci.*, **62** (2007), 4589–4601. https://doi.org/10.1016/j.ces.2007.05.025
23. R. Bürger, S. Diehl, Convexity-preserving flux identification for scalar conservation laws modelling sedimentation, *Inverse Probl*, **29** (2013), 045008. https://doi.org/10.1088/0266-5611/29/4/045008
24. R. Bürger, J. Careaga, S. Diehl, Flux identification of scalar conservation laws from sedimentation in a cone, *IMA J Appl Math*, **83** (2018), 526–552. https://doi.org/10.1093/imamat/hxy018
25. S. Diehl, Numerical identification of constitutive functions in scalar nonlinear convection–diffusion equations with application to batch sedimentation, *Appl Numer Math*, **95** (2015), 154–172. https://doi.org/10.1016/j.apnum.2014.04.002
26. M. Mishra, Machine learning framework for data driven acceleration of computations of differential equations, *Math. eng.*, **1** (2018), 118–146. https://doi.org/10.3934/Mine.2018.1.118
27. J.W. Thomas, Numerical partial differential equations–Conservation laws and elliptic equations, *Texts in Applied Mathematics*, New York: Springer, 1999.
28. J.S. Hesthaven, *Numerical methods for conservation laws. from analysis to algorithms*, Philadelphia: Society for Industrial and Applied Mathematics, 2017.
29. D. Kröener, *Numerical schemes for conservation laws*, New York: John Wiley & Sons, 1997.
30. M. Mishra, U.S. Fjordholm, R. Abgrall, Numerical methods for conservation laws and related equations. *Lecture notes for Numerical Methods for Partial Differential Equations* **57** (2019), 58.

31. Valerii Iakovlev, Markus Heinonen, Harri Lähdesmäki, *Learning continuous-time pdes from sparse data with graph neural networks*, arXiv: 2006.08956, [Preprint], (2020) [cited 2022 Oct 18]. Available form: https://arxiv.53yu.com/abs/2006.08956

32. C. Zhu, R.H. Byrd, P. Lu, J. Nocedal, Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization, *Acm T math software*, **23** (1997), 550–560. https://doi.org/10.1145/279232.279236

33. Sebastian R, *An overview of gradient descent optimization algorithms*, arXiv: 1609.04747, [Preprint], (2016) [cited 2022 Oct 18]. Available form: https://arxiv.org/abs/1609.04747

34. H.J. Skadsem, S. Kragset, A numerical study of density-unstable reverse circulation displacement for primary cementing, *J. Energy. Resour. Technol.*, **144** (2022), 123008. https://doi.org/10.1115/1.4054367

AIMS Press

Paper IV:
Identification of the Flux Function of Nonlinear Conservation Laws with Variable Parameters

Qing Li, Jiahui Geng, Steinar Evje

Physica D: Nonlinear Phenomena 451 (2023): 133773
doi = https://doi.org/10.1016/j.physd.2023.133773

Physica D 451 (2023) 133773

Contents lists available at ScienceDirect

Physica D

journal homepage: www.elsevier.com/locate/physd

Identification of the flux function of nonlinear conservation laws with variable parameters

Qing Li [a], Jiahui Geng [b], Steinar Evje [a,*]

[a] University of Stavanger, Department of Energy and Petroleum, Group of Computational Engineering, Norway
[b] University of Stavanger, Department of Electrical Engineering and Computer Science, Stavanger, Norway

ARTICLE INFO

Article history:
Received 8 November 2022
Received in revised form 21 March 2023
Accepted 28 April 2023
Available online 9 May 2023
Communicated by R. Kuske

Keywords:
Nonlinear conservation law
Variable parameter flux function
Machine learning method
Entropy condition
Linear Regression Neural Network
Symbolic neural network

ABSTRACT

Machine learning methods have in various ways emerged as a useful tool for modeling the dynamics of physical systems in the context of partial differential equations (PDEs). Nonlinear conservation laws (NCLs) of the form $u_t + f(u)_x = 0$ play a vital role within the family of PDEs. A main challenge with NCLs is that solutions contain discontinuities. That is, one or several jumps of the form $(u_L(t), u_R(t))$ with $u_L \neq u_R$ may move in space and time such that information about $f(u)$ in the interval associated with this jump is not present in the observation data. Moreover, the lack of regularity in the solution $u(x, t)$ prevents use of physics informed neural network (PINN) and similar methods. The purpose of this work is to propose a method to identify the nonlinear flux function $f(u, \beta)$ with variable parameters of an unknown scalar conservation law

$$u_t + f(u, \beta)_x = 0 \qquad (*)$$

with u as the dependent variable and β as the parameter. In a recent work we introduced a framework coined ConsLaw-Net that combines a symbolic multi-layer neural network and an entropy-satisfying discrete scheme to learn the nonlinear, unknown flux function $f(u; \beta)$ for various fixed β. Learning the flux function with variable parameters, marked as $f(u, \beta)$, is more challenging since it requires an understanding of the relationship between the variable u and parameter β as well. In this work we demonstrate how to couple ConsLaw-Net to the Linear Regression Neural Network (LRNN) to learn the functional form of the two variable function $f(u, \beta)$. In addition, ConsLaw-Net is here further developed and made more generic by using a refined discrete scheme combined with a more general symbolic neural network (S-Net). We experimentally demonstrate that the upgraded ConsLaw-Net integrated with LRNN is well-suited for learning tasks, achieving more accurate identification than existing learning approaches when applied to problems that involve general classes of flux functions $f(u, \beta)$. The investigations of this work are restricted to the class of polynomial, rational functions.

1. Introduction

1.1. Background

Machine learning methods for solving partial differential equation problems have been widely used in science and engineering, for example, electromagnetism [1], aerodynamics [2,3], weather prediction [4], and geophysics [5]. Raissi et al. [6] introduced the physics informed neural networks (PINNs) for solving partial differential equations (PDEs) and learning parameters in PDEs. Long et al. [7,8] proposed PDE-Net that combines numerical approximation of differential operators and symbolic multi-layer neural networks. The authors reported that PDE-Net has the most flexible and expressive power by learning both differential operators and the nonlinear response function of the underlying PDE model. In this work, we focus on the problem of learning the unknown nonlinear flux function $f(u, \beta)$ that is involved in the general scalar nonlinear conservation law. Here we are restricted to the one-dimensional case, given by

$$\begin{aligned} u_t + f(u, \beta)_x &= 0, \qquad x \in (0, L) \\ u|_{t=0} &= u_0(x) \\ u_x|_{x=0} &= u_x|_{x=L} = 0 \end{aligned} \qquad (1)$$

where $u = u(x, t)$ is the main variable and β is a parameter, typically, related to different forces that drive the process under consideration. Note that this extension to include dependence on a parameter β is highly relevant for many situations where different physical parameters naturally vary over certain intervals.

* Corresponding author.
E-mail addresses: quing.li@uis.no (Q. Li), jiahui.geng@uis.no (J. Geng), steinar.evje@uis.no (S. Evje).

https://doi.org/10.1016/j.physd.2023.133773

Displacement of several fluids is one classical example where quantities like fluid viscosity and density may strongly influences the displacement behavior and their roles are expressed in a β-like parameter [9,10]. In the context of learning $f(u;\beta)$ for a fixed β, we identify $f(u;\beta)$ with u as the only variable based on a series of observations $\{u(x_j,t_i)\}$ at specific time points $\{t_i\}$ described on a spatial grid $\{x_j\}$. However, when learning $f(u,\beta)$, we seek the analytical expression of $f(u,\beta)$ that includes both the variable u and the parameter β, given a set of observations $\{u(x_j,t_i)\}$ at various time points t_i and at different values of β over an interval of interest.

There are special challenges in identifying $f(u,\beta)$ in (1) as compared to previous PDE models as studied in, e.g., [6–8]. Firstly, in our scenario, the term $f(u,\beta)_x$ in (1) cannot be expressed by $f_u(u,\beta)u_x$ since u generally is not a differentiable function. Consequently, we cannot approximate differential operators using convolutions, as done in [7,8]. In our earlier work [11], we demonstrated that the proposed ConsLaw-Net for solving $f(u;\beta)$ for a fixed β can handle this problem. Secondly, variable u and parameter β are completely distinct. How to learn the functional relation between u and β? A natural idea is to try to discover their relationship by treating both as variables. I.e., feed u and β into the ConsLaw-Net at the same time. However, it seems difficult to control and minimize the error associated with both u and β simultaneously. The discrete scheme we use to represent the entropy solution of (1) would produce a cumulative error in the ConsLaw-Net, especially when the number of iterations is large. Incorporating learning of an unknown parameter into the ConsLaw-Net significantly increases the complexity of the calculations involved in minimizing the loss function. To overcome this, we introduce a Linear Regression Neural Network(LRNN) to distill the structure of $f(u,\beta)$ learned by relying on the ConsLaw-Net for a few selected values of β.

At this stage, it seems appropriate to highlight some essential features of the entropy solution associated with (1). It is well known that conservation laws of the form $(1)_{1,2}$ (Cauchy problem) do not in general possess classical solutions. Instead, one must consider weak solutions in the sense that the following integral equality holds [12–14]

$$\int_{\Omega_x}\int_0^T \Big[u\phi_t + f(u,\beta)\phi_x\Big]\,dx\,dt + \int_{\Omega_x} u_0(x)\phi(x,t=0)\,dx = 0 \quad (2)$$

for all $\phi \in C^1$ such that $\phi(x,t) : \Omega_x \times (0,T) \to \mathbb{R}$ and which is compactly supported, i.e., ϕ vanishes at $x \to \Omega_x$ and $t \to T$. It follows that if a discontinuity occurs in the solution, i.e., a left state u_L and a right state u_R, then the speed s satisfies the Rankine–Hugoniot jump condition [12,14], i.e.,

$$s = \frac{f(u_L,\beta) - f(u_R,\beta)}{u_L - u_R}. \quad (3)$$

However, direct calculations show that there are several weak solutions for one and the same initial data [13]. To overcome this issue of non-uniqueness of weak solutions, we need criteria to determine whether a proposed weak solution is admissible or not. This has led to the class of *entropy* solutions, which amounts to introducing an additional constraint which ensures that the unique physically relevant one is found among all the possible weak solutions. There are different ways to express the entropy condition for scalar nonlinear conservation laws. One variant is by introducing an entropy pair (η,q) where $\eta : \mathbb{R} \to \mathbb{R}$ is any strictly convex function and $q : \mathbb{R} \to \mathbb{R}$ is constructed as [14,15] (where $f'(s)$ represents derivative with respect to s since β is a passive parameter)

$$q(v,\beta) = \int_0^v f'(s,\beta)\eta'(s)\,ds \quad (4)$$

for any v. This implies that $q' = f'\eta'$. Then, u is an entropy solution of $(1)_{1,2}$ if (i) u is a weak solution in the sense of (2); (ii) u satisfies in a weak sense $\eta(u)_t + q(u,\beta)_x \le 0$ for any pair (η,q). This condition can also be formulated as the following characterization of a discontinuity (u_L,u_R) [14,16]: For all numbers v between u_L and u_R,

$$\frac{f(v,\beta)-f(u_L,\beta)}{v-u_L} \ge s \ge \frac{f(v,\beta)-f(u_R,\beta)}{v-u_R} \quad (5)$$

where s is given by (3). In particular, it gives a tool for constructing unique solutions of nonlinear scalar conservations laws as given by (1).

The ConsLaw-Net framework introduced in [11] relies on different symbolic neural networks, S-Net-M for multiplicative function and S-Net-D for division function to represent the flux function, and are combined with an entropy consistent discrete numerical scheme (ECDNS) to identify $f(u;\beta)$ for a fixed β. However, this ConsLaw-Net has two shortcomings: Firstly, the entropy consistent numerical scheme relies on a global estimate of $|f_u(u;\beta)|$ through $M = \max_u |f_u(u;\beta)|$. This facilitates the symbolic neural network to find the optimal solution quickly. However, it is also known to corrupt data accuracy. In this work we explore whether the flux function can be correctly learned when the symbolic neural network is coupled with a numerical scheme that relies on a local estimate of the form $M_{j+1/2} = \max\{|f_u(u_j^n;\beta)|, |f_u(u_{j+1}^n;\beta)|\}$ where u_j^n and u_{j+1}^n refer to u at spatial grid points x_j and x_{j+1}, to improve the overall accuracy of the method. Secondly, there are two different symbolic neural networks involved, S-Net-M and S-Net-D for multiplicative and division functions, respectively. We must decide which type of symbolic neural network to use by comparing the final losses of S-Net-M and S-Net-D, which obviously increases the computational burden.

1.2. Purpose of this work

In particular, our main contribution by this work includes:

- Provide a method for learning of the functional form of the nonlinear two-variable function $f(u,\beta)$ where u is the main variable and β is a parameter. To our knowledge, this represents the first approach to learn the unknown nonlinear flux function involved in a scalar conservation law, with respect to both its dependent variable u as well as the parameter β.
- We update the ConsLaw-Net proposed in [11] regarding the following two aspects:
  - We use a local rather than global estimate of the wave speed for the Rusanov scheme. Specifically, we use a local estimate $M_{j+1/2} = \max\{|f_u(u_j^n;\beta)|, |f_u(u_{j+1}^n;\beta)|\}$ instead of a global estimate of $M = \max_u |f_u(u;\beta)|$. This updated ECDNS provides a more accurate description of observation data versus the underlying flux function, but also increases the computational time for model optimization (i.e., identification of the unknown flux function). The success of applying the improved numerical scheme demonstrates that the methodology can be extended to a wider variety, i.e., the ConsLaw-Net model has some robustness;
  - We employ a unified Symbolic Multi-layer Neural Network (S-Net) to learn $f(u;\beta)$, regardless of whether it is a division or a multiplication function that is involved in the unknown underlying conservation law (1).

Hence, the updated ConsLaw-Net is a refinement and a generalization as compared to its first version presented

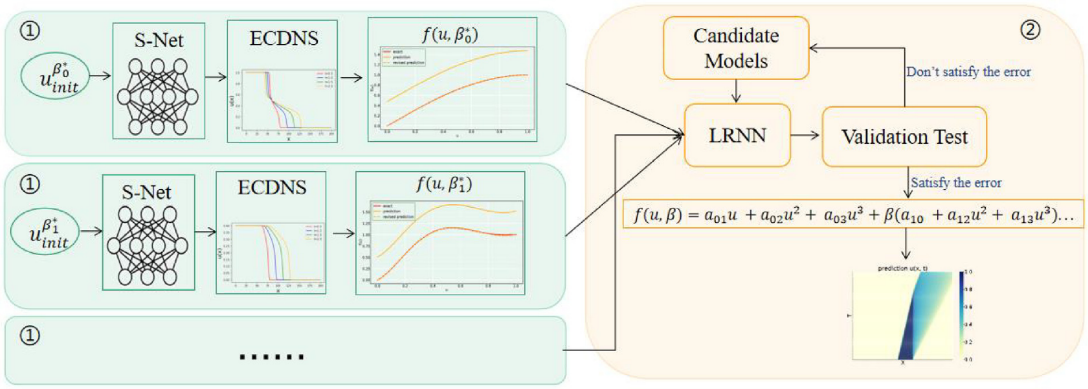


Fig. 1. Pipeline for our approach. There are two steps: ① using the improved ConsLaw-Net to learn $f(u;\beta)$ where $\beta \in \{\beta_0^*, \beta_1^*, \ldots, \beta_{N_\beta}^*\}$. We use the symbol β^* to distinguish the values of β used in the first step. The learned expression $f(u;\beta_i^*)$ in each subproblem i is referring to an equation with u as variable but not β_i^*. ② Given the results of the first step, we first construct a collection of candidate models of the two variable function $f(u,\beta)$, and then use LRNN to obtain the coefficients of the selected candidate model. If this candidate model can satisfy the error criterion, we will apply it to compute the solutions of (1). Otherwise, we reselect the candidate model and repeat the process in the second step.

in [11]. However, this comes with an additional challenge related to the optimization of (1). To handle this, we design a new loss function.

- Learned graph neural networks (GNNs) have recently been established as fast and accurate methods when applied to PDEs with regular solutions, such as the Wave equation, Poisson, or Navier–Stokes equations [17–19]. We compare our method with the GNNs method proposed in [17]. Experiments show that our method generates relative accurate predictions whereas this GNN does not seem to have a built-in capacity to handle the problem (1) well.

The proposed approach is summarized in Fig. 1. This method consists of two steps. The first step is to use the optimized ConsLaw-Net to learn $f(u;\beta)$ for a few values of β, whereas the second step is to use LRNN to learn $f(u,\beta)$ relying on the results of the first step. The challenge with lack of uniqueness, i.e., the fact that many different flux functions can give rise to the same observation data, is dealt with by using a combination of two different components: (i) we collect observation data corresponding to a set of different initial states; (ii) regularity information pertaining to the class of flux functions $f(u,\beta)$ we seek, is incorporated through the use of symbolic neural networks, as mentioned above.

1.3. Related work

Learning the flux function of a nonlinear conservation law from sparse data has important applications in industry. There are several examples of non-machine approaches to deal with this problem. Holden et al. used the front-tracking algorithm to reconstruct the flux function from observed solutions with suitable initial data [20] for a traffic flow relevant model. Recent studies on the reconstruction of the flux function for sedimentation problems involved the separation of a flocculated suspension into a clear fluid and a concentrated sediment [21,22]. In particular, Bürger and Diehl revealed that the inverse problem of identifying the batch flux density function has a unique solution, and derived an explicit formula for the flux function [23]. This method was recently extended to construct almost the entire flux function [24] using a cone-shaped separator. Diehl explored a direct inversion method using linear combinations of finite element hat functions to represent the unknown nonlinear function [25].

With the development of deep AI, more and more neural network methods are used to solve forward and inverse problems of PDEs and ODEs. These problems frequently involve recovering solutions to PDEs [6] or physical quantities, such as the density, viscosity, or other material parameters [26–28], from a sparse set of measurements. We refer to [29,30] for examples of methods based on assuming that the governing PDE is linear combination of a few differential terms in a prescribed dictionary, and the objective is to find the correct coefficients. Raissi et al. [6] introduced the physics informed neural networks (PINNs) for solving two types of problems: data-driven solutions and data-driven discovery of partial differential equations. Except for a few scalar learnable parameters, the explicit form of the PDEs is assumed to be known in the second problem. Therefore, the structure of the equation is used as part of the loss function to guide the optimization. However, PINNs seem not able to learn the nonlinear hyperbolic PDE that governs two-phase transport in porous media [10]. It was suggested that this shortcoming of PINNs for hyperbolic PDEs is due to the lack of regularity in the solution. Mesh-based simulations have recently made significant progress [17,18], enabling faster runtimes than principled solvers, and higher adaptivity to the simulation domain compared to the grid-based convolutional neural network (CNNs) [31,32]. Neural networks have been used also in combination with discrete schemes to accelerate computations of PDEs [33].

The rest of this paper is organized as follows: In Section 2 the updated version of the ConsLaw-Net proposed in [11] is described. In particular, it is described how LRNN is used to learn the two variable flux function $f(u,\beta)$. In Sections 3 and 4 we present different experimental results on identifying $f(u,\beta)$ using our method by exploring performance, respectively, for two different families of flux functions. In Section 5, we close with a discussion and outlook.

2. Method

2.1. The first step: using an improved ConsLaw-net to learn the fixed parameter problem $f(u;\beta)$, with $\beta \in \{\beta_0^*, \beta_1^*, \ldots, \beta_{N_\beta}^*\}$.

2.1.1. Entropy consistent discrete numerical scheme (ECDNS)

We investigate a discretization of the spatial domain $[0, L]$ in terms of $\{x_i\}_{i=0}^{N_x-1}$ where $x_i = (1/2+i)\Delta x$ for $i = 0, \ldots, N_x-1$ with $\Delta x = L/N_x$. Furthermore, we consider time lines $\{t^n\}_{n=0}^{N_t}$ with $N_t \Delta t = T$. Our discrete version of (1) is based on the Rusanov scheme [12] which takes the form

$$u_j^{n+1} = u_j^n - \lambda(F_{j+1/2}^n - F_{j-1/2}^n), \qquad \lambda = \frac{\Delta t}{\Delta x},$$

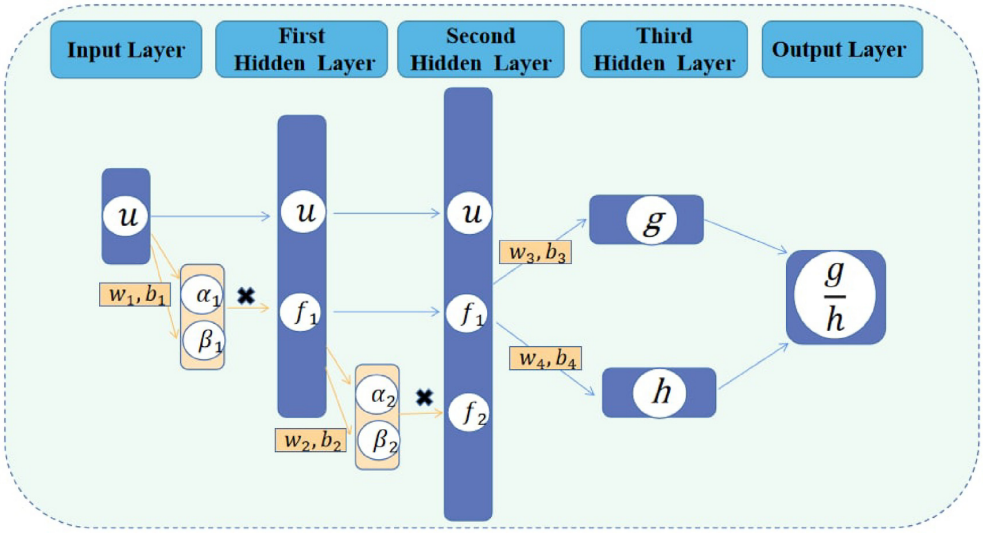


Fig. 2. The structure of S-Net with three hidden layers.

$$u_1^{n+1} = u_2^{n+1}, \qquad u_{N_x}^{n+1} = u_{N_x-1}^{n+1} \tag{6}$$

with $j = 2, \ldots, N_x - 1$ and where the Rusanov flux takes the form

$$F_{j+1/2}^n = \frac{f(u_j^n;\beta) + f(u_{j+1}^n;\beta)}{2} - \frac{M_{j+1/2}^n}{2}(u_{j+1}^n - u_j^n).$$

We adopt a local estimation by using $M_{j+1/2}^n = \max\{|f_u(u_j^n;\beta)|, |f_u(u_{j+1}^n;\beta)|\}$ instead of a global approximation through $M_{j+1/2}^n = \max_u |f_u(u;\beta)|$, as used in [11]. The local estimation is beneficial to simulation accuracy but harmful to the optimization of the model. The CFL condition determines the magnitude of Δt for a given Δx through the constraint [12]

$$\text{CFL} := \frac{\Delta t}{\Delta x} M \le 1, \qquad M = \max_u |f_u(u;\beta)|.$$

We invoke the CFL condition in Algorithm 1. Algorithm 2 presents how to learn the solution $U^n = \{u(x_j, t^n)\}_{j=1}^{N_x}$ of the discrete conservation law (6). The local estimation $M_{j+1/2}^n$ is used to define the interface flux $F_{j+\frac{1}{2}}$ in Algorithm 2. Note that we for simplicity ignore the explicit dependence on β in Algorithm 1 and Algorithm 2 since they are applied for $f(u;\beta)$ with a fixed choice of β.

Algorithm 1: CFL

Input: L: length of the spatial domain; N_x: the number of spatial grid cells; $f(u)$: the nonlinear flux function; T: computational time period;
Output: Δt: local time interval

$\Delta x = L/N_x$
$M = \max_u |f'(u)|$
$dt = (\frac{3}{4}\Delta x)/(M + 0.0001)$
$n\_time = \lfloor T/dt \rfloor$
$\Delta t = T/n\_time$

2.1.2. S-net

Inspired by [34,35], we use the S-Net to represent the unknown function $f(u;\beta)$ because S-Net can learn the analytical expression of the flux function $f(u;\beta)$. In this work we no longer distinguish between S-Net-M and S-Net-D, but instead use S-Net-D as S-Net to learn all functions. Fig. 2 depicts the building of

Algorithm 2: DataGenerator

Input: T: computational time period; N_x: the number of spatial grid cells; L: length of the spatial domain; $u_0 = \{u_0(x_j)\}_{j=1}^{N_x}$: initial state set of dimension N_x; $f(u)$: the flux function;
Output: $U = \{u_j^n\}$: the solution based on initial state u_0;

Δt = CFL(L, N_x, $f(u)$, T)
$\Delta x = L/N_x$
$U[0] = u_0$
$\tilde{u} = u_0$
for n = 1,...,$T/\Delta t$ **do**
  for j = 1,...,N_x - 1 **do**
    $F_{j+1/2} = \frac{1}{2}\left(f(\tilde{u}_j) + f(\tilde{u}_{j+1})\right) - \frac{\max\{|f'(\tilde{u}_j)|, |f'(\tilde{u}_{j+1})|\}}{2}\left(\tilde{u}_{j+1} - \tilde{u}_j\right)$
  end
  for j = 2,...,N_x - 1 **do**
    $u_j = \tilde{u}_j - \frac{\Delta t}{\Delta x}\left(F_{j+1/2} - F_{j-1/2}\right)$
  end
  $u_1 = u_2$
  $u_{N_x} = u_{N_x-1}$
  $\tilde{u} = u$
  $U[i] = u$
end

S-Net in the form of $f = g/h$ with three hidden layers. As shown in Fig. 2, the identity directly maps u from input layer to the first hidden layer. The linear combination map uses parameters $\mathbf{w_1}$ and $\mathbf{b_1}$ to choose two elements from u and are denoted by α_1 and β_1.

$$(\alpha_1, \beta_1)^T = \mathbf{w_1} \cdot (u) + \mathbf{b_1}, \mathbf{w_1} \in \mathbb{R}^{2\times 1}, \mathbf{b_1} \in \mathbb{R}^{2\times 1} \tag{7}$$

These two elements of α_1 and β_1 are multiplied which give

$$f_1 = \alpha_1 \beta_1 \tag{8}$$

Apart from u gotten by the identity map, f_1 also is input to the second hidden layer

$$(\alpha_2, \beta_2)^T = \mathbf{w_2} \cdot (u, f_1)^T + \mathbf{b_2}, \mathbf{w_2} \in \mathbb{R}^{2\times 2}, \mathbf{b_2} \in \mathbb{R}^{2\times 1}. \tag{9}$$

Similarly with the first hidden layer, we get another combination f_2

$$f_2 = \alpha_2\beta_2 \tag{10}$$

Then we obtain the numerator part g and the denominator part h of the flux function $f(u)$ based on $\mathbf{w_3}$, $\mathbf{b_3}$ and $\mathbf{w_4}$, $\mathbf{b_4}$, respectively.

$$g = \mathbf{w_3}\cdot(u, f_1, f_2)^T + \mathbf{b_3}, \mathbf{w_3} \in \mathbb{R}^{1\times 3}, \mathbf{b_3} \in \mathbb{R} \tag{11}$$

$$h = \mathbf{w_4}\cdot(u, f_1, f_2)^T + \mathbf{b_4}, \mathbf{w_4} \in \mathbb{R}^{1\times 3}, \mathbf{b_4} \in \mathbb{R} \tag{12}$$

The analytic expression of the flux function f is the combination of g and h, i.e.,

$$f = \frac{g}{h}. \tag{13}$$

The parameters involved in the network described above is denoted by $\boldsymbol{\theta}$ and the resulting function according to (13) is denoted by $f_{\boldsymbol{\theta}}(u)$. More information about S-Net can be found in [11].

2.1.3. The ConsLaw-net

The ConsLaw-Net combines ECDNS and S-Net to learn $f(u;\beta)$ with $\beta \in \{\beta_0^*, \beta_1^*, \ldots, \beta_{N_\beta}^*\}$. We consider observation data in terms of x-dependent data at fixed times $\{t_i^*\}_{i=1}^{N_{\text{obs}}}$ extracted from the solution U as follows:

$$U_{\text{sub}} = \left\{u(x_j, t_1^*), u(x_j, t_2^*), \ldots, u(x_j, t_{N_{\text{obs}}}^*)\right\}, \quad j = 1, \ldots, N_x. \tag{14}$$

Eq. (14) is utilized to select synthetic observation data denoted by U_{sub} as well as predictions based on the learned $f_{\boldsymbol{\theta}}(u;\beta)$ written as $\hat{U}_{\text{sub}}$. We specify times for collecting the time dependent data

$$T_{\text{obs}} = \{t_i^* = i\Delta t^{\text{obs}} : i = 1, \ldots, N_{\text{obs}}\}. \tag{15}$$

Typically, we set $N_{\text{obs}} = 9$ with $\Delta t^{\text{obs}} = 0.1$. For clarity, we use $f_{\boldsymbol{\theta}}(u;\beta)$ to denote the function learned by S-Net. During S-Net training, $f_{\boldsymbol{\theta}}(u;\beta)$ is fed into Algorithm 2 to obtain the predicted solution $\hat{U}$. Similar with U_{sub}, choose predicted solutions $\hat{U}_{sub}$ according to (15). The difference between U_{sub} and $\hat{U}_{sub}$, denoted as L^{data}, is involved in the optimization of the model. In addition to L^{data}, we also introduce $L_\gamma^{denominator}$ and $L_\gamma^{numerator}$ in the loss function to limit the search space of the flux function. Detailed information on the loss functions is given in Section 2.3. Herein, we use the second-order quasi-Newton method, L-BFGS-B [36,37], to update the parameter vector $\boldsymbol{\theta}$. Finally, we get the best flux function $f_{\boldsymbol{\theta}^*}(u;\beta)$ and use it to represent the learned conservation law for the selected β. The learning process is depicted in Algorithm 3.

Overall, learning the fixed parameter problem $f(u;\beta)$ can be formalized as a nonlinear optimization problem as follows:

$$\begin{aligned} &\min_{f(u)} Loss(U_{sub}, \hat{U}_{sub}), \\ &\text{s.t.} \quad f(u) \in R(u) \end{aligned} \tag{16}$$

where $f(u)$ is the flux function to be learned and $R(u)$ is the set of rational functions. U_{sub} and $\hat{U}_{sub}$ are extracted from the solution of Eq. (1) with fixed parameter β based on the true flux function and learned flux function, respectively. *Loss* is the operator that measures the difference between U_{sub} and $\hat{U}_{sub}$ and $Loss(U_{sub}, \hat{U}_{sub})$ is the objective function.

2.1.4. Why use S-net instead of the piecewise affine functions method or the finite polynomial expansion method?

A wide range of methods exists to solve nonlinear optimization problems, including traditional approaches like gradient-based and Quasi-Newton methods, as well as contemporary methods such as machine learning techniques. James and Sepúlveda [38,39] minimized $Loss(U_{sub}, \hat{U}_{sub})$ by means of gradient methods. Since $Loss(U_{sub}, \hat{U}_{sub})$ may not be differentiable

Algorithm 3: ConsLaw-Net

Input: T: computational time period; N_x: the number of spatial grid cells; L: length of the spatial domain; u_0: initial state vector of dimension N_x; $f(u;\beta)$: true flux function with fixed parameters β; $\boldsymbol{\theta}_0$: initial parameters of S-Net; $\boldsymbol{\theta}$: parameters of S-Net; $f_{\boldsymbol{\theta}}(u;\beta)$: flux function generated by S-Net; *T_obs*: observation time points Eq. (15); *Loss*: loss function; *epoch*: the number of epoch; *DataGenerator*: Algorithm 2

Output: $f_{\boldsymbol{\theta}^*}(u;\beta)$: the best flux function generated by the S-Net based on parameter vector $\boldsymbol{\theta}^*$;

```
U = DataGenerator(T, N_x, L, u_0, f(u; β))
U_sub = {u ∈ U | t ∈ T_obs}
θ = θ_0
for i = 1,...,epoch do
    Û = DataGenerator(T, N_x, L, u_0, f_θ(u; β))
    Û_sub = {u ∈ Û | t ∈ T_obs}
    loss = Loss(U_sub, Û_sub)
    Updating θ and loss by optimizer L-BFGS-B;
end
θ* = θ
```

with respect to the parameters of $f(u;\beta)$ and the dependence of objective function on these parameters occurs via the PDE solution, additional assumptions have to be made, and the formal gradient is obtained by means of an adjoint equation. Already for a hyperbolic scalar conservation law in one dimension, the problem of identifying a nonlinear flux $f(u;\beta)$ is generally ill-posed in the sense that $f(u;\beta)$ is not uniquely determined by observed data. For example, the same discontinuity in a solution may arise from different flux functions. Recently, Diehl in [25] formulated a general flux identification method based on expanding the flux function in terms of piecewise affine functions with unknown weights and obtained the least squares solution to the integrated form of the conservation equation which could also contain a diffusion term.

To solve the nonlinear optimization problem (16), we have opted for machine learning methods, which offer greater robustness and flexibility. In particular, we are using S-Net, as employed in [7,8,34,35], to learn the unknown function. Besides S-Net, other classical methods explicitly represent functions, such as the piecewise affine functions method described in [25] and the finite polynomial expansion method. In the piecewise affine functions method, divide the u-axis into n equidistant intervals: let u^{max} be the largest value of the data and set $u_k = ku^{max}/n$ for $k = -1, \ldots, n+1$. Assume that

$$f(u) = \sum_{k=0}^{n} f_k \psi_k(u) \tag{17}$$

where f_k is $n+1$ parameters to be determined, and the affine functions are defined by

$$\psi_k(u) = \begin{cases} \frac{u-u_{k-1}}{u_k-u_{k-1}}, & u_{k-1} < u \le u_k \\ \frac{u_{k+1}-u}{u_{k+1}-u_k}, & u_k < u \le u_{k+1} \\ 0, & \text{Otherwise} \end{cases} \tag{18}$$

where $k = 0, \ldots, n$. The polynomial expression with respect to u is given by:

$$f(u;\beta) = \frac{c_0 + c_1 u + c_2 u^2 + c_3 u^3 + \cdots + c_n u^{n_1}}{d_0 + d_1 u + d_2 u^2 + d_3 u^3 + \cdots + d_n u^{n_2}} \tag{19}$$

We acquire the coefficients $\{f_k\}_{k=0}^n$ of (17) and $\{c_i\}_{i=0}^{n_1}$ and $\{d_i\}_{i=0}^{n_2}$ of Eq. (19), respectively, by using the observation data set. From

the various methods available, we have decided to implement the S-Net approach for the following reasons:

- Experimental performance: In Section 3.1.2 we do a comparison of S-Net, the piecewise affine function representation, and the finite polynomial expansion method. It is observed that the performance of S-Net is clearly better than these two other methods for our test cases.
- Robustness: S-Net demonstrates a high capacity to handle noisy and incomplete data [11]. In our problem, the number of local time steps (of length Δt) we need to compute numerical solutions through ECDNS is higher than the number of observation data, i.e., $\Delta t \ll \Delta t^{\text{obs}}$. Since Δt is dictated by the CFL condition for the given choice of the flux function f (which will vary during the training), we do not know that $\Delta t K_i = t_i^*$ for $i = 1, \ldots, N_x$ for some integer K_i. In that case, we choose the one closest to t_i^*. Given that the observations are derived from approximate data, we have opted for S-Net due to its robustness.
- Flexibility: S-Net is flexible and powerful. Expanding functions with more variables is easy with S-Net, as it can automatically detect nonlinear relationships between input variables. The finite polynomial expansion method requires the manual listing of all possible combinations, which becomes an exhaustive task as the number of variables increases, potentially leading to an exponential growth in the number of combinations. Furthermore, incorporating nonlinearities such as sine and cosine into S-Net is straightforward, enabling it to learn combinations of variables that act on sine or cosine.

2.2. The second step: using LRNN to learn $f(u, \beta)$

From the first step, we have learned the fixed parameter problems $f_{\theta^*}(u; \beta)$, $\beta \in \{\beta_0^*, \beta_1^*, \ldots, \beta_{N_\beta}^*\}$. Based on the above results, we learn $f(u, \beta)$ using LRNN in the following section.

2.2.1. Generate observation data **Y**

We have learned the analytical expressions of $f_{\theta^*}(u; \beta)$ with some fixed β in the first step, denoted as

$$\mathbf{Y}' = \left\{ f_{\theta^*}(u; \beta) \,\middle|\, u \in [\min(u_0), \max(u_0)],\ \beta \in \{\beta_0^*, \beta_1^*, \ldots, \beta_{N_\beta}^*\} \right\}.$$

Since the solution of (1) is TVD (total variation diminishing) [15, 40], we know that the solution $u(x, t)$ at any time $t > 0$ does not contain any new maxima or minima as compared to the initial data $u_0(x)$, i.e., $\min u_0(x) \le u(x, t) \le \max u_0(x)$. In the following we tactically assume that $0 \le u_0(x) \le 1$, therefore $0 \le u(x, t) \le 1, \forall t \in [0, T]$. Consider a discretization of $u \in [0, 1]$, denoted as

$$\mathscr{U} = \left\{ u \,\middle|\, u_i = i\Delta u, i = 0, 1, 2, \ldots, N_u, \Delta u = 1/N_u \right\}. \tag{20}$$

Therefore, a discrete version of Y' is obtained by

$$\mathbf{Y} = \left\{ f_{\theta^*}(u_i; \beta) \,\middle|\, u_i \in \mathscr{U},\ \beta \in \{\beta_0^*, \beta_1^*, \ldots, \beta_{N_\beta}^*\} \right\}. \tag{21}$$

Typically, $N_u = 400$. Y will be used as the observation data to train LRNN.

2.2.2. Construct the candidate models for $f(u, \beta)$ based on Linear Regression Neural Network (LRNN)

Next, we create the candidate models of $f(u, \beta)$ that contain multiplication function form (22) and division function form (23):

$$f(u, \beta) = \sum_{i=0}^{K} \sum_{j=0}^{J_i} \left(a_{ij} \beta^i u^j \right) \tag{22}$$

$$f(u, \beta) = \frac{\sum_{i=0}^{K} \sum_{j=0}^{J_i} \left(a_{ij} \beta^i u^j \right)}{\sum_{i=0}^{P} \sum_{j=0}^{Q_i} \left(b_{ij} \beta^i u^j \right)} \tag{23}$$

where $K, J_i, P, Q_i \in \mathbb{N}$, and $\mathscr{P} = \left\{ a_{ij} | i = 0, \ldots, K; j = 0 \cdots J_i \right\} \cup \left\{ b_{ij} | i = 0, \ldots, P; j = 1, \ldots, Q_i \right\} \subset \mathbb{R}$, β^i denotes the i-th power of β, u^j represents the j-th power of u. The largest power of u, i.e., J_i, Q_i, can in principle be any positive integer. The use of (22) and (23) implies that the flux function $f(u)$ we seek to identify is restricted to be a polynomial, rational function or one that can be well approximated within this class. We can appropriately limit the candidate models by using the learned expressions of $f(u; \beta^*)$ obtained from the first step. Generally speaking, the biggest power of u in the true function will not be greater than the largest power in the learned expressions. $\mathscr{P}$ is the vector of the parameters we want to learn in the second phase. According to the Occam's razor principle, the preferred formula is singled out by being the simplest one that still predicts well. Therefore, we test the candidate model based on the principle from simplicity to complexity. That is, from multiplication to division models, and from lower to higher powers. If the simple model can meet the error requirements in the validation set, we use this simple model instead of the complex model.

Specifically, to begin with a straightforward approach, we set $K = 1$ and $J_0 = J_1 = 1$ in (22), which yields a simple model,

$$f(u, \beta) = a_{00} + a_{01} u^1 + a_{10} \beta + a_{11} \beta u^1. \tag{24}$$

There are four unknown parameters, namely $a_{00}, a_{01}, a_{10}, a_{11}$, in (24). These parameters can be optimized based on the results obtained in the first step described in Section 2.1, denoted as $a_{00}^*, a_{01}^*, a_{10}^*, a_{11}^*$. The learning process is explained in detail in Sections 2.2.3 and 2.2.4. The resulting equation can be denoted as

$$f(u, \beta) = a_{00}^* + a_{01}^* u^1 + a_{10}^* \beta + a_{11}^* \beta u^1. \tag{25}$$

To evaluate the ability of (25) in representing the true flux function, we must validate it using a separate dataset. This validation dataset, denoted as U_{sub}^{valid}, comprises the solutions extracted from Eq. (1) for specific time points (15), generated using β that was not used in the first step. We can then solve for Eq. (1) with the learned flux function (25) using the same β as the validation dataset, resulting in the estimated solutions $\hat{U}_{sub}^{valid}$. If the errors between U_{sub}^{valid} and $\hat{U}_{sub}^{valid}$ satisfy the specific error requirement Δ, we consider the learned equations to be a suitable representation of the true flux function. In this case, we do not conduct further validation of the other candidate models. However, if the errors do not meet the requirements, we move to a more complex model. For instance, we may set $K = 1$ and $J_0 = J_1 = 2$ in (22). We gradually increase the complexity of the candidate models until we find one that satisfies the error requirements.

2.2.3. Establish the input **X**

After choosing the possible expression of $f(u, \beta)$, we have to establish the features of the expression in order to learn the coefficients a_{ij}, b_{ij} in (22) or (23). For each item $a_{ij}(b_{ij})\beta^i u^j$ in (22) and (23), we can build features according to $\beta^i u^j$. Take the possible expression (26) as an example.

$$f(u, \beta) = a_{00} + a_{01} u^1 + a_{02} u^2 + \beta(a_{10} + a_{11} u^1 + a_{12} u^2). \tag{26}$$

For every $\beta \in \{\beta_0^*, \beta_1^*, \ldots, \beta_{N_\beta}^*\}$ and $u_i \in \mathscr{U}$, we can create a set of features

$$\mathbf{X}_{i,\beta} = \{1, u_i^1, u_i^2, \beta, \beta u_i^1, \beta u_i^2\}.$$

Denote the features of the potential formulation of $f(u, \beta)$ as

$$\mathbf{X} = \left\{ \mathbf{X}_{i,\beta} \,\middle|\, u_i \in \mathscr{U},\ \beta \in \{\beta_0^*, \beta_1^*, \ldots, \beta_{N_\beta}^*\} \right\}. \tag{27}$$

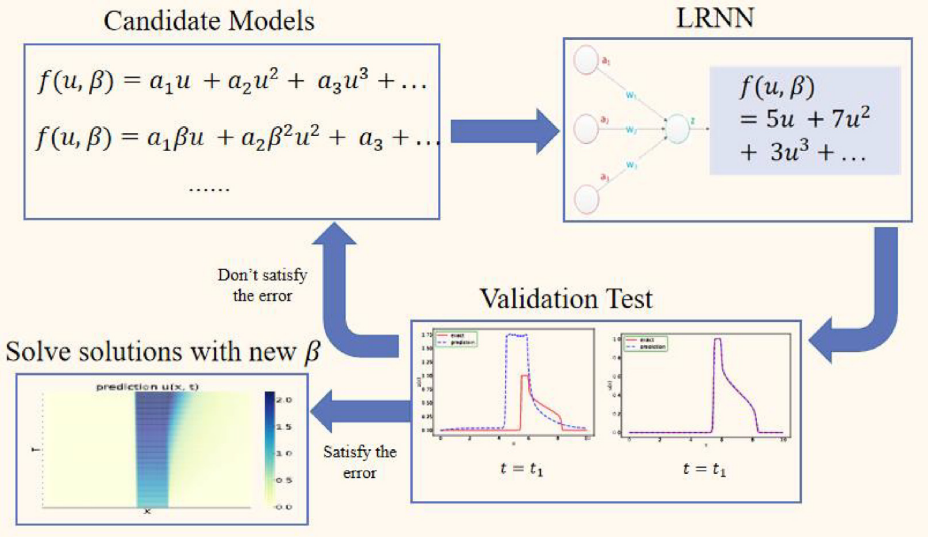


Fig. 3. Pipeline for the second step of our approach: Choose one possible expression $f(u,\beta)$ from candidate models. Build the observation data Y (21) and features of $f(u,\beta)$ (27), and get the coefficients $a_{ij}(b_{ij})$ by LRNN. Test the learned $f(u,\beta)$ on the validation data. If the error criteria is met, stop; otherwise, continue with the operation.

2.2.4. Train the model

Assume the parameter vector of LRNN is **W**, so

$$\mathbf{Y} = \mathbf{W}^T\mathbf{X}. \tag{28}$$

Using the neural network, we can easily get **W** which is the vector of coefficients of the possible expression of $f(u,\beta)$.

2.2.5. Test on the validation dataset

After obtaining the trained model, it is tested on the validation set. If the error condition is satisfied, the model behavior can be predicted using the new parameter β and initial state u_0. Fig. 3 depicts the pipeline for the second step of our approach.

2.3. Loss functions

2.3.1. Loss function used in the first step

Optimizations of the ConsLaw-Net, including using a local estimate $M_{j+1/2} = \max\{|f_u(u_j^n,\beta)|, |f_u(u_{j+1}^n,\beta)|\}$ instead of a global estimate $M = \max_u |f_u(u,\beta)|$ and employing a unified S-Net to learn $f(u;\beta)$, place high demands on our loss functions. To obtain the solutions of (1), we need to estimate the quantity M, that decides the local time interval Δt used in Algorithm 2. If M is too large, it will cause Δt too small according to $\Delta t = \frac{\frac{3}{4}\Delta x}{M+0.0001}$ in Algorithm 1. In this case, the numerical scheme needs to be iterated tens of thousands of times to obtain the solutions of (1), which takes an inordinate amount of time. As a result, we estimate an upper bound on M, denoted as α. In our experiments, $\alpha = 100$. When $M < \alpha$, we compare the difference between U_{sub} and $\hat{U}_{\text{sub}}$ to guide optimization, written as L^{data} and defined in (30). S-Net has two sub-neural networks, $g(u)$ and $h(u)$, to learn the numerator and the denominator, respectively.

There are two cases where the M value is too large caused by $g(u)$ and $h(u)$. One is when the denominator $h(u)$ is very close to 0, and the other is when the value of the denominator is reasonable, but the numerator $g(u)$ is vast. The above situations inevitably occur during the optimization of the model. To deal with these two cases, we introduce $L_\gamma^{denominator}$ and $L_\gamma^{numerator}$ to optimize the model. Therefore, the loss function used in Algorithm 3 can be expressed as

$$Loss = \begin{cases} L^{data} & M < \alpha \\ L_\gamma^{denominator} & M \geq \alpha,\ L_\gamma^{denominator} \neq 0 \\ L_\gamma^{numerator} & M \geq \alpha,\ L_\gamma^{denominator} = 0 \end{cases} \tag{29}$$

where L^{data}, $L_\gamma^{denominator}$, and $L_\gamma^{numerator}$ are defined as follows:

$\boldsymbol{L^{data}}$: Assume we have K_{ini} different initial states used for the training process, and solutions are described on a grid of N_x grid cells and at N_{obs} different times. The true observation data set denoted by $\{U_{\text{sub},k}(x_j,t_i^*) : 1 \leq k \leq K_{ini};\ 1 \leq j \leq N_x;\ 1 \leq i \leq N_{obs}\}$ is obtained using Algorithm 2. The predicted data is created by an iterative loop in Algorithm 3, and is denoted by $\{\hat{U}_{\text{sub},k}(x_j,t_i^*) : 1 \leq k \leq K_{ini};\ 1 \leq j \leq N_x;\ 1 \leq i \leq N_{obs}\}$. So we define the data approximation term L^{data} as:

$$L^{data} = \frac{1}{K_{ini}N_xN_{obs}} \sum_{k=1}^{K_{ini}} \sum_{j=1}^{N_x} \sum_{i=1}^{N_{obs}} \left(U_{\text{sub},k}(x_j,t_i^*) - \hat{U}_{\text{sub},k}(x_j,t_i^*)\right)^2. \tag{30}$$

$\boldsymbol{L_\gamma^{denominator}}$: To steer the network away from small values of the denominator, we add a penalty term to our objective (29), denoted by $L_\gamma^{denominator}$. Compute the values of the subneural network $h(u)$ on $\mathscr{U}$ (20), written as

$$H = \{h(u_i) | i = 0, 1, 2, \ldots, N_u\}.$$

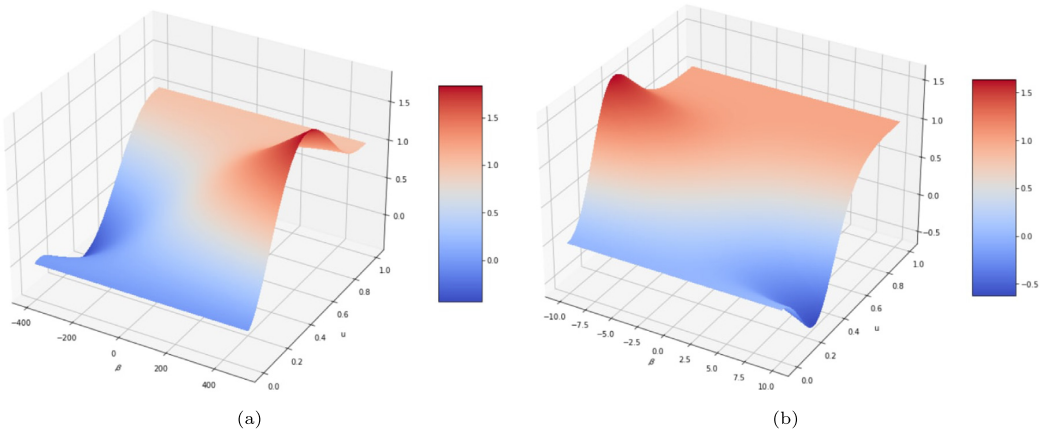
If there are elements in H smaller than γ, it can cause M to be too large (i.e., $M \geq \alpha$). In our experiment, we set $\gamma = 0.001$ and let $L_\gamma^{denominator}$ be associated with the net effect of these small values of H

$$L_\gamma^{denominator} = \sum_0^{N_u} |h(u_i)|\, I(h(u_i) < \gamma),$$

$$I(h(u_i) < \gamma) = \begin{cases} 1, & \text{if } h(u_i) < \gamma \\ 0, & \text{otherwise} \end{cases} \tag{31}$$

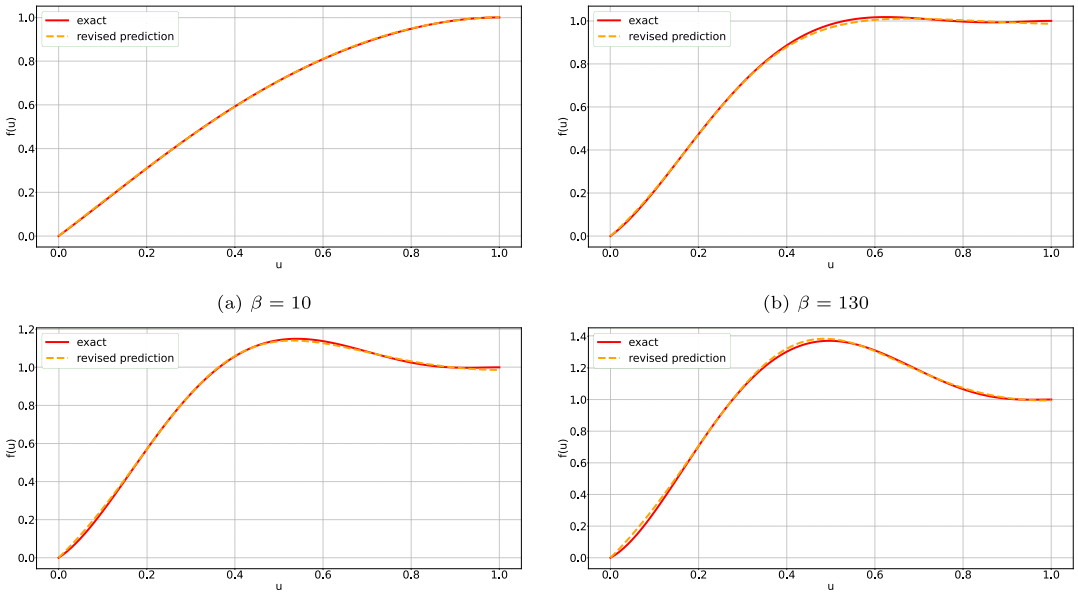
Note that $0 < L_\gamma^{denominator} < (N_u + 1)\gamma$.

$\boldsymbol{L_\gamma^{numerator}}$: When the value of the denominator h is greater than γ, i.e., $h(u_i) \geq \gamma$ for all $i \in \mathscr{U}$ implying that $L_\gamma^{denominator} = 0$, and the value of the numerator g is too large, M can also be too



(a) (b)

Fig. 4. (a) Flux function (34); (b) Flux function (39).



(a) $\beta = 10$ (b) $\beta = 130$

(c) $\beta = 200$ (d) $\beta = 300$

Fig. 5. The graphical results of identification of flux function $f_{\theta^*}(u;\beta)$ on $\beta = 10, 130, 200, 300$. In each subplot, the solid red line is the true function, and the orange dashed line is the learned function.

big (i.e., $M \geq \alpha$). In this situation, we compute the approximate derivative of $f(u)$ with respect to $u \in [0, 1]$. For that purpose we introduce

$$\mathscr{U}' = \mathscr{U} \cup \{1 + \Delta u\} = \{u_i | i = 0, 1, 2, \ldots, N_{u+1}\}.$$

We compute the discrete derivative of $f(u)$ on $\mathscr{U}'$, denoted by

$$Df(u) = \left\{ \frac{f(u_{i+1}) - f(u_i)}{\Delta u} \middle| i = 0, 1, 2, \ldots, N_u \right\}$$

where $L_\gamma^{numerator}$ is set to be

$$L_\gamma^{numerator} = \max(|Df(u)|) - \alpha. \tag{32}$$

The purpose of $L_\gamma^{denominator}$ and $L_\gamma^{numerator}$ is to steer the loss function (29) in a direction such that $Loss = L^{data}$ is the dominating term, which allows to identify the unknown flux function. As u is bounded by $[0, 1]$, L^{data} in the case of $M < \alpha$ is much smaller than $L_\gamma^{denominator}$ or $L_\gamma^{numerator}$ which are at work in the case of $M \geq \alpha$. During the optimization process of the neural network, the parameters are updated in the direction that minimizes the loss function (29), i.e., the situation where loss is governed by $L_\gamma^{denominator}$ or $L_\gamma^{numerator}$ will be avoided. In this sense, the presence of $L_\gamma^{denominator}$ and $L_\gamma^{numerator}$ steers the loss function (29) in a direction such that $Loss = L^{data}$ is the governing term, which allows to identify the unknown flux function.

2.3.2. Loss function used in the second step

We optimize models using the Adam optimizer [41] for 5000 epochs and minimize MSE between observation data (21) and predicted values $\overline{\mathbf{Y}}_W = \left\{ \bar{f}_W(u_i, \beta) \middle| u_i \in \mathscr{U}, \beta \in \{\beta_0^*, \beta_1^*, \ldots, \beta_{N_\beta}^*\} \right\}$ generated by LRNN where W represents the parameter vector, see (28). One of the primary reasons for selecting Adam is its ability to automatically adjust the learning rate, which ensures

the accuracy of learning. For clarity, we denote the trained LRNN function as $\bar{f}_{W^*}(u,\beta)$. The loss function is written as

$$Loss = \frac{1}{(N_u+1)(N_\beta+1)}\sum_{i=0}^{N_u}\sum_{j=0}^{N_\beta}\left(f_{\theta^*}(u_i;\beta_j^*)-\bar{f}_W(u_i,\beta_j^*)\right)^2 \quad (33)$$

3. Learning a general class of nonlinear flux functions $f(u,\beta)$

In this section, we consider a class of nonlinear conservation laws that naturally arise from studying displacement of one fluid by another fluid in a vertical domain. The resulting displacement process involves a balance between buoyancy and viscous forces. Depending on the properties of the fluids used, there is room for various types of displacement behavior. This is expressed by the fact that one can derive a family of flux functions $f(u,\beta)$ which takes the following form as illustrated in [9]

$$f(u,\beta)=\frac{1}{2}u(3-u^2)+\frac{\beta}{12}u^2\left(\frac{3}{4}-2u+\frac{3}{2}u^2-\frac{1}{4}u^4\right). \quad (34)$$

The parameter β represents the balance between gravity (buoyancy) and viscous forces. In this work, $\beta\in[-400,500]$. Different values of β result in different types of flux functions. Fig. 4(a) shows the shape of $f(u,\beta)$.

In the following, we generate synthetic data with a few different values of β and a class of initial data $u_0(x)$. As mentioned in the introduction, observation data of the entropy solution leaves room for many possible flux functions to be candidates to explain the observations. However, as demonstrated in [11], the combination of using a few initial data combined with additional regularity by representing the unknown flux function through symbolic multilayer neural networks, enables identification of the relevant flux function quite effectively. In other words, as more initial data is added (with corresponding observation data), more details of the physically relevant flux function are revealed.

We consider a spatial domain $L=10$ such that $x\in[0,10]$ and a time interval $[0,T]$ with $T=2$. We collect observation data in the form (14) with $N_{obs}=9$. The aim is to identify the unknown flux function $f(u,\beta)$ for $u\in[0,1]$.

3.1. The first step: using the improved ConsLaw-net to learn $f(u;\beta)$ where $\beta=\{10,130,200,300\}$

3.1.1. The results of using S-net to represent $f(u;\beta)$

In order to learn $f(u;\beta)$ for $u\in[0,1]$, we consider a set of initial data $\{u_0^k\}_{k=1}^K$ such that $0\le u_0^k(x)\le 1$. We choose box-like states that give rise to Riemann problems, one at each initial discontinuity. In the following, we apply a numerical grid composed of $N_x=400$ grid cells and choose observation data (14) with time points

$$\{t_i^*\}=\{0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9\}. \quad (35)$$

S-Net is used to represent the unknown flux function $f(u;\beta)$ where $\beta\in\{10,130,200,300\}$. Set three hidden layers in S-Net, and the total number of trainable parameters in every subproblem is 28. Choose the observations from the solutions U generated by Algorithm 2 based on initial states shown in Table 7 in Appendix A.1. Finally, we obtain after training the S-Net denoted by $f_{\theta^*}(u;\beta)$ by applying Algorithm 3. The resulting analytical expressions are given in Table 1. $f_{\theta^*}(u;\beta)$ differs from the true one. However, we recall that what matters is the derivative $f'_{\theta^*}(u;\beta)$. To visualize the learning effect, we plot the translated function $f_{\theta^*}(u;\beta)-f(u=0;\beta)$ in Fig. 5. Clearly, the improved ConsLaw-Net has the ability to identify the true flux function with quite good accuracy for the flux $f_{\theta^*}(u;\beta)$ with $\beta\in\{10,130,200,300\}$.

3.1.2. Comparison of S-net versus the piecewise affine functions and the finite polynomial expansion methods.

In addition to utilizing S-Net to represent the flux function, we also investigated the piecewise affine functions method (17) and (18) and the finite polynomial expansion method (19). Figs. 6(a) and 6(b) demonstrate the learned flux function $f(u;\beta)$ for $\beta=200$ and $\beta=300$, respectively, using the piecewise affine functions method with $n=10$ in (17). Lower values of n gave better results with less oscillatory behavior. Figs. 6(c) and 6(d) demonstrate the learned flux function $f(u;\beta)$ for $\beta=200$ and $\beta=300$, respectively, using the finite polynomial expansion method (19) with $n_1=8$ ($\beta=200$) and $n_1=7$ ($\beta=300$). Upon comparing these figures with Figs. 5(c) and 5(d), it is evident that the flux function learned by S-Net is clearly better than the results obtained by these two methods.

3.2. The second step: using LRNN to learn the flux function (34)

In this section, we distill the analytical expression corresponding to (34) from the results of the four subproblems where $\beta=10,130,200,300$. We build candidate models in the form of (22) and (23) with different values of K, J_i, P, Q_i. For example, setting $K=1$ and $J_i=6$ in (22), we obtain

$$\begin{aligned} &f_W(u,\beta)\\ &=a_{00}+a_{01}u+a_{02}u^2+a_{03}u^3+a_{04}u^4+a_{05}u^5+a_{06}u^6\\ &\quad+\beta(a_{10}+a_{11}u+a_{12}u^2+a_{13}u^3\\ &\quad+a_{14}u^4+a_{15}u^5+a_{16}u^6). \end{aligned} \quad (36)$$

The learning details using LLNN on (36) are shown in the following. Herein, $\{a_{ij}|i=0,1;j=0,1,2,\dots,6\}$ is the set of parameters that we will learn in the second step. Discretize $u\in[0,1]$ into 401 points: $\{u_i\}_{i=0}^{i=400}$, and use the following 14 features at each point $u_i\in\{u_i\}_{i=0}^{i=400}$:

$$\mathbf{X}_{i,\beta}=[1,u_i,u_i^2,u_i^3,u_i^4,u_i^5,u_i^6,\beta,\beta u_i,\beta u_i^2,\beta u_i^3,\beta u_i^4,\beta u_i^5,\beta u_i^6].$$

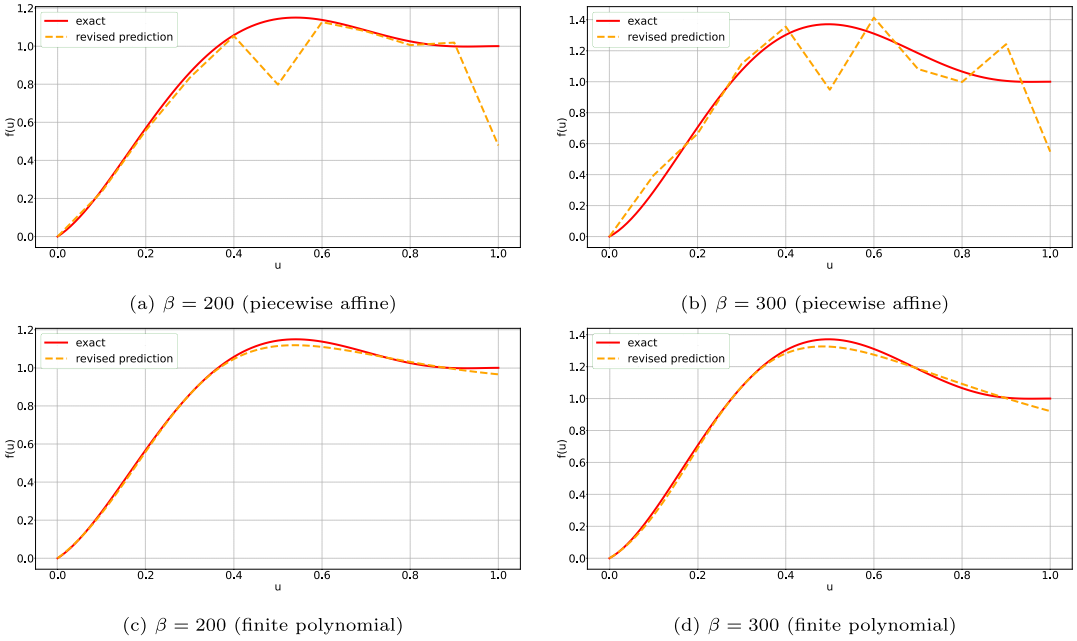
Feed features $\mathbf{X}=\{\mathbf{X}_{i,\beta}|i=0,1,2\dots400;\beta=10,130,200,300\}$ into LRNN, and finally we acquire the learned parameters of $\{a_{ij}|i=0,1;j=0,1,2,\dots,6\}$ by using Adam optimizer to train the LRNN. The learned analytical expression of (36) is denoted by

$$\begin{aligned} &f_{W^*}(u,\beta)\\ &=0.15651u^6-0.018213u^5-0.39308u^4-0.33295u^3\\ &\quad+0.19598u^2+1.3823u+0.008492\\ &\quad+\beta(-0.037291u^6+0.051649u^5+0.059704u^4\\ &\quad-0.12439u^3+0.048243u^2+0.0020499u\\ &\quad-1.7243e-5). \end{aligned} \quad (37)$$

Is the identified two variable function $f_{W^*}(u,\beta)$ in (37) capable to represent (34) for different β in the whole interval $[-400,500]$? We test on the validation set where $\beta=-50$ and $\beta=350$ (which is well outside the β values used for training). Table 2 shows the test results of (37) where $K=1$ and $J_i=6$. Obviously, for both cases $\beta=-50$ and $\beta=350$, the error is relatively small, and meets the criteria we set, $\Delta=0.0001$. We also test other candidate models on the validation set. The results are shown in Table 3. These models may perform better in one case, but worse in another, such as $K=1$ and $J_i=4$. Or even worse in both cases, such as $K=1$ and $J_i=2$. It is obvious that when $K=1$ and $J_i=6$, the candidate model has the best performance. Therefore, we choose (37) as the preferred learned model and use it to test for new initial states and parameters.

Table 1
The expression results of identification of flux function $f_{\theta^*}(u;\beta)$ with specific β.

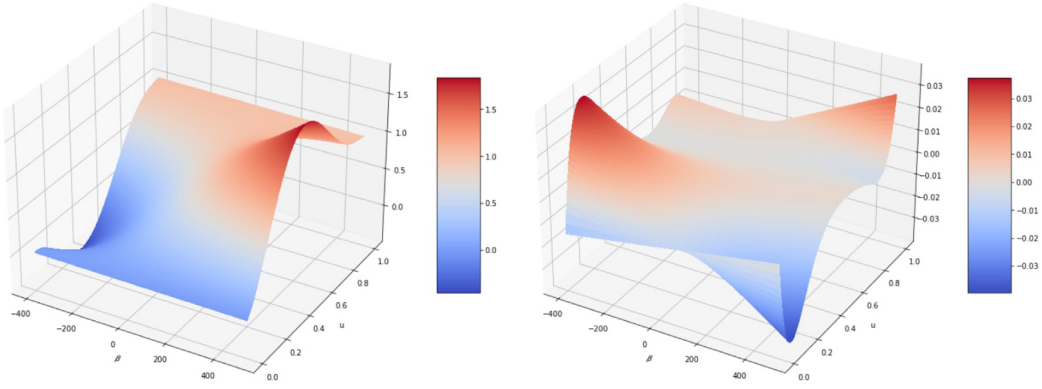
β	The identification of flux function $f_{\theta^*}(u;\beta)$
$\beta_1 = 10$	$f_{\theta^*}(u;\beta_1) = \frac{-0.0005u^8+0.0029u^7-0.0036u^6-0.0030u^5+0.3214u^4-0.9641u^3-0.0198u^2+1.7007u-1.6453}{0.0002u^8-0.0014u^7+0.0018u^6+0.0015u^5-0.0013u^4+0.0025u^3+0.0431u^2-0.0663u+1.0205}$
$\beta_2 = 130$	$f_{\theta^*}(u;\beta_2) = \frac{0.0171u^8-0.0451u^7-0.0553u^6+0.1688u^5+0.6588u^4-0.9677u^3-1.2969u^2+1.2216u-0.5410}{0.0058u^8-0.0153u^7-0.0188u^6+0.0573u^5-0.4797u^4+0.5975u^3+0.9857u^2-0.3791u+0.3038}$
$\beta_3 = 200$	$f_{\theta^*}(u;\beta_3) = \frac{-3.004e{-}6u^8+0.0001u^7-0.0005u^6-0.0144u^5-0.0181u^4-0.3600u^3-2.2609u^2+1.7718u-0.6950}{9.4576e{-}6u^8-0.0004u^7+0.0015u^6+0.0453u^5+0.1119u^4+0.0464u^3+0.7181u^2-0.4093u+0.2220}$
$\beta_4 = 300$	$f_{\theta^*}(u;\beta_4) = \frac{2.001e{-}5u^7-0.0071u^6-0.0981u^5-0.2864u^4-0.1568u^3-3.1849u^2+2.6000u-0.9848}{-1.1929e{-}5u^7+0.0042u^6+0.0585u^5+0.1709u^4-0.0887u^3+0.6433u^2-0.4293u+0.2083}$



(a) $\beta = 200$ (piecewise affine) (b) $\beta = 300$ (piecewise affine)

(c) $\beta = 200$ (finite polynomial) (d) $\beta = 300$ (finite polynomial)

Fig. 6. The graphical results of identification of flux function $f_{\theta^*}(u;\beta)$ using the piecewise affine functions method (17) and (18) (upper row) and the finite polynomial expansion method (19) on $\beta = 200, 300$ (lower row). In each subplot, the solid red line is the true function, and the orange dashed line is the learned function.

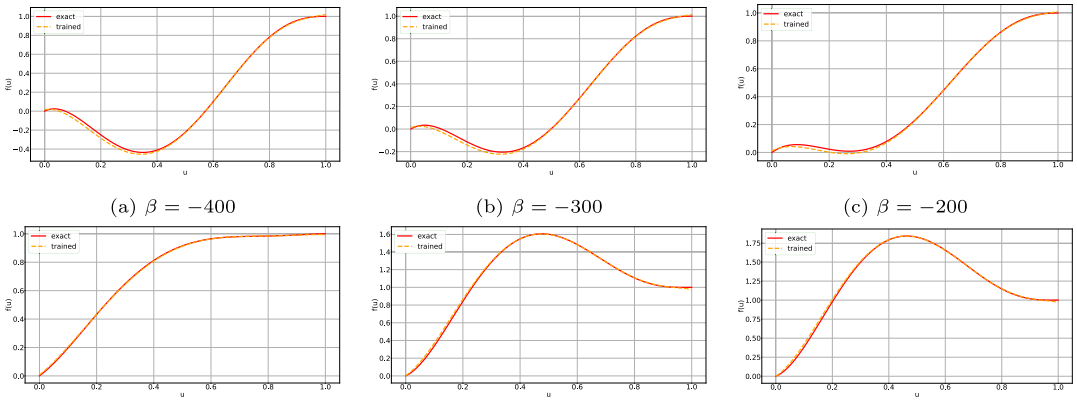


(a) (b)

Fig. 7. (a) Learned flux function (37). (b) The difference between the true flux function (34) and learned (37).

3.3. Results and discussions

In Fig. 7(a) we first visualize the learned flux function $f_{W^*}(u,\beta)$ given by (37) whereas Fig. 7(b) visualizes the error between learned flux function and the true flux function $f(u,\beta)$ given by (34) for $\beta \in [-200, 500]$ and $u \in [0, 1]$. The absolute error of the two functions is less than 0.038 over the entire domain. Specifically, we set $\beta = -400, -300, -200, 100, 400, 500$ separately to



(a) $\beta = -400$ (b) $\beta = -300$ (c) $\beta = -200$

(d) $\beta = 100$ (e) $\beta = 400$ (f) $\beta = 500$

Fig. 8. The generalization ability of (37) tested for $\beta = -400, -300, -200, 100, 400, 500$. In each subplot, the solid red line is the true function, and the orange dashed line is the learned function.

Table 2
The results of (37) applied on the test data $\beta = -50$ and $\beta = 350$. The value of Error is the MSE between true and predicted solutions of (1) at times (35). We also plot the solution at time point $t = 0.3, 0.6, 0.9$ respectively. In each subplot, the solid red line is the true solution, and the blue dashed line is the solution generated by (37) based on initial state $u_0 = \begin{cases} 1.0, & \text{if } x \in [4.5, 6] \\ 0.0, & \text{otherwise} \end{cases}$.

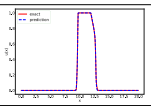
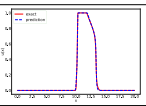
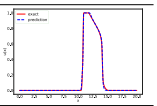
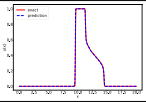
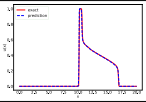
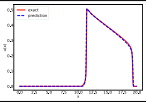
β	Error	t = 0.3	t = 0.6	t = 0.9
$\beta = -50$	7.5443e−05			
$\beta = 350$	0.0001			

Table 3
The results of different candidate models on the validation set. The value is the MSE between true and predicted solutions of (1) at times (35) for four different initial states $u^0_{\beta=-50,350} = \begin{cases} 1.0, & \text{if } x \in [4.5, 6] \\ 0.0, & \text{otherwise} \end{cases}$, $u^1_{\beta=-50,350} = \begin{cases} 0.9, & \text{if } x \in [3.5, 5] \\ 0.0, & \text{otherwise} \end{cases}$, $u^2_{\beta=-50,350} = \begin{cases} 0.8, & \text{if } x \in [5, 6.5] \\ 0.0, & \text{otherwise} \end{cases}$ and $u^3_{\beta=-50,350} = \begin{cases} 0.7, & \text{if } x \in [3, 6] \\ 0.0, & \text{otherwise} \end{cases}$, respectively.

β	$K = 1$, $J_i = 1$	$K = 1$, $J_i = 2$	$K = 1$, $J_i = 3$	$K = 1$, $J_i = 4$	$K = 1$, $J_i = 5$	$K = 1$, $J_i = 7$	$K = 1$, $J_i = 8$
$\beta = -50$	0.0013	0.0011	0.0006	0.0002	0.0002	0.0002	0.0003
$\beta = 350$	0.0515	0.0046	0.0083	0.0019	0.0028	0.0002	0.0012
β	$k = 2$, $J_i = 1$	$K = 2$, $J_i = 2$	$K = 2$, $J_i = 3$	$K = 2$, $J_i = 4$	$K = 1$, $J_i = 3$, $P = 1$, $Q_i = 3$	$K = 1$, $J_i = 4$, $P = 1$, $Q_i = 4$	$K = 1$, $J_i = 5$, $P = 1$, $Q_i = 5$
$\beta = -50$	0.0018	0.0056	0.0177	0.0095	0.0004	0.0002	0.0001
$\beta = 350$	0.0522	0.0049	0.0977	0.0221	0.0004	0.0005	0.0004

test the generalization ability of $f_{W^*}(u, \beta)$ (37). In Fig. 8, we show the true and the learned function with different values of β. It seems clear that (37) has relatively strong generalization ability.

In addition, we compare the modified graph neural networks (GNNs) used in [17] with our approach. The GNNs method offers faster runtimes than other solvers like PINNs and grid-based CNNS [31,32] and better adaptivity to the simulation domain [17–19]. One major difference between our method and the GNNs method is that our method first learns the expression of $f(u, \beta)$ and then uses the learned model to compute solutions of (1) based on new initial states and parameters. In contrast, the GNNs method directly predicts the solution of (1). This type of GNNs is called message passing neural networks (MPNNs) [42], and is given in the form

$$\frac{du(x_i, t)}{dt} = \hat{F}_\theta(x_{\mathcal{N}(i)} - x_i, u_i, u_{\mathcal{N}(i)}, \beta_i) \tag{38}$$

where x_i is the coordinate of the node i, $\mathcal{N}(i)$ is the neighborhood of node i, θ denote parameters of the MPNNs and β_i is the physical parameter in the flux function. We use Rprop optimizer [43] to minimize the loss function with learning rate set to $lr = 10^{-6}$

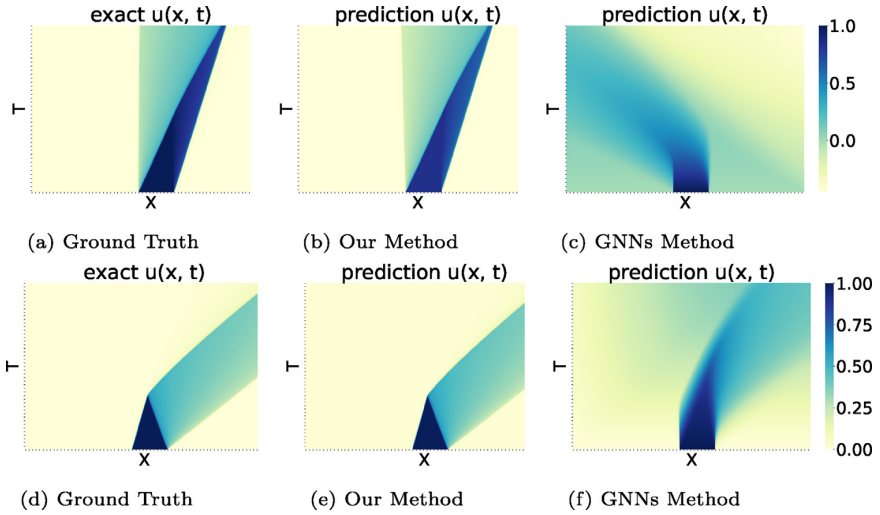


Fig. 9. The compared results of our method and the GNNs method. We compare the two methods at $\beta = -200$ (top row) and $\beta = 400$ (bottom row) based on initial state $u_0 = \begin{cases} 1.0, & \text{if } x \in [4.5, 6] \\ 0.0, & \text{otherwise} \end{cases}$. Left: True solutions of (1). Middle: Solutions generated by our method. Right: Solution generated by the GNNs method.

for 5000 iterations and batch size set to 4. To be fair, the GNNs method and our method use the same observational data. The trained model is used to predict based on new initial states and parameter β.

Fig. 9 shows the solutions of (1) with $\beta = 400$ and $\beta = -200$ using the two different methods. Our method performs better than the GNNs model. The ConsLaw-Net method is based on a discrete scheme which is entropy consistent and therefore can deal with formation of discontinuities in the solutions. The above simulation result for GNN suggests that it has not a built-in capacity to deal with discontinuous solutions.

4. Learning the flux function $f(u, \beta) = \frac{u^2\left(1-\beta(1-u)^4\right)}{u^2+0.5(1-u)^4}$

In this section, we consider a class of nonlinear conservation laws that appears in the context of two-phase flow in porous media [10]. Displacement of two fluids in an inclined reservoir gives rise to a family of flux functions in the form

$$f(u, \beta) = \frac{u^2}{u^2 + 0.5(1-u)^4}\left(1 - \beta(1-u)^4\right). \tag{39}$$

The parameter β represents the gravity effect. Herein, we study $\beta \in [-10, 10]$. Fig. 4(b) shows the shape of $f(u, \beta)$. In the following, we generate synthetic data with different values of β and a class of initial data $u_0(x)$. Consider a spatial domain $L = 10$ such that $x \in [0, 10]$ and a time interval $[0, T]$ with $T = 2$.

4.1. The first step: using the improved ConsLaw-net to learn $f(u; \beta)$ where $\beta \in \{-1, 1, 3, 5, 7\}$

Set a numerical grid composed of $N_x = 400$ grid cells, and consider observation data (14) with

$$\{t_i^*\} = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}. \tag{40}$$

S-Net with three hidden layers is used to represent the unknown flux function $f(u; \beta)$, where $\beta \in \{-1, 1, 3, 5, 7\}$. The total number of trainable parameters in every subproblem is 28. Choose the observations from the solutions U generated by Algorithm 2 based on initial states shown in Table 8 in Appendix A.2. Finally, we obtain after training the S-Net denoted by $f_{\theta^*}(u; \beta)$ by applying Algorithm 3. The resulting analytical expressions are given in Table 4. We plot the translated function $f_{\theta^*}(u; \beta) - f(u = 0; \beta)$ in Fig. 10. Clearly, the improved ConsLaw-Net has the ability to identify the true flux function in most ranges except for some lack of accuracy for $u \in \{0, 0.2\}$.

4.2. The second step: using LRNN to learn Eq. (39)

In this section, we distill the analytical expression that can approximate the true function (39) from the five subproblems where $\beta = -1, 1, 3, 5, 7$. We build candidate models in the form of (22) and (23) with different values of K, J_i, P, Q_i. For example, setting $K = 1, J_i = 5, P = 1$ and $Q_i = 5$ in (23), then we get Eq. (41) (see Box I) where $W = \left\{a_{ij}, b_{ij} | i = 0, 1; j = 0, 1, 2, \ldots, 5\right\}$ is the set of parameters that we will learn in the second step. Discretize $u \in [0, 1]$ into 401 points: $\{u_i\}_{i=0}^{i=400}$, and build the following 12 features at each point $u_i \in \{u_i\}_{i=0}^{i=400}$:

$$\mathbf{X}_{i,\beta} = [1, u_i, u_i^2, u_i^3, u_i^4, u_i^5, \beta, \beta u_i, \beta u_i^2, \beta u_i^3, \beta u_i^4, \beta u_i^5].$$

Put features $\mathbf{X} = \left\{\mathbf{X}_{i,\beta} | i = 0, 1, 2 \ldots 400; \beta = -1, 1, 3, 5, 7\right\}$ into LRNN, and finally we acquire the learned parameters of $\left\{a_{ij}, b_{ij} | i = 0, 1; j = 0, 1, 2, \ldots, 5\right\}$ by using Adam optimizer. The learned analytical expression of $f(u, \beta)$, in terms of g and h, then becomes

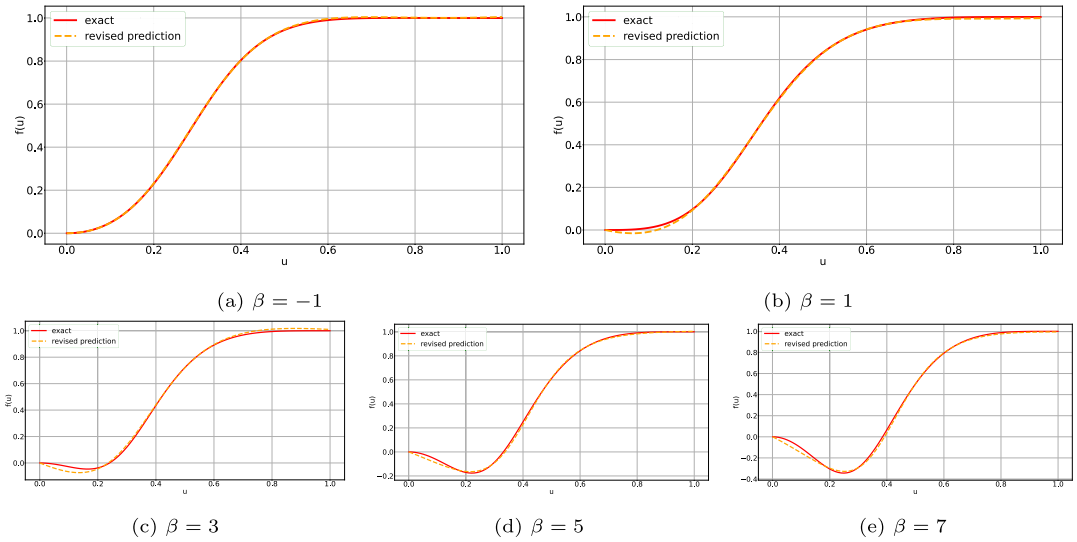
$$\begin{aligned} g_{W^*}(u, \beta) &= \beta(0.099u^5 + 0.078u^4 + 0.261u^3 - 0.184u^2 \\ &\quad - 0.021u - 0.00012) \\ &\quad + 1.920u^5 + 0.380u^4 - 0.113u^3 + 0.852u^2 \\ &\quad - 0.060u + 0.0003 \\ h_{W^*}(u, \beta) &= \beta(0.126u^5 + 0.183u^4 + 0.0645u^3 - 0.186u^2 \\ &\quad + 0.048u - 0.0016) \\ &\quad + 1.710u^5 + 0.400u^4 + 0.509u^3 + 0.487u^2 \\ &\quad - 0.3112u + 0.17557 \end{aligned} \tag{42}$$

Hence, the learned expression of $f(u, \beta)$ is given by

$$f_{W^*}(u, \beta) = \frac{g_{W^*}(u, \beta)}{h_{W^*}(u, \beta)}. \tag{43}$$

Table 4
The results of identification of flux function $f_{\theta^*}(u;\beta)$ with specific β.

β	The identification of flux function $f_{\theta^*}(u;\beta)$
$\beta_1=-1$	$f_{\theta^*}(u;\beta_1)=\frac{-0.0808u^8+0.2555u^7-0.0379u^6-0.4234u^5+3.2486u^4-4.6848u^3-3.2596u^2+3.6620u-1.2970}{0.3590u^8-1.1358u^7+0.1685u^6+1.8822u^5-2.9914u^4+2.7129u^3+2.6686u^2-1.7852u+0.6337}$
$\beta_2=1$	$f_{\theta^*}(u;\beta_2)=\frac{-0.9589u^8+4.7491u^7-6.1089u^6-0.2182u^5+7.73097u^4-13.0489u^3+0.7633u^2+4.3662u-2.3523}{1.4639u^8-7.2501u^7+9.3260u^6+0.3332u^5-7.0604u^4+8.1785u^3+1.9316u^2-2.5813u+1.2348}$
$\beta_3=3$	$f_{\theta^*}(u;\beta_3)=\frac{0.0002u^8+0.0091u^7+0.1226u^6+0.7346u^5+2.5724u^4+6.5190u^3+7.1617u^2-1.9897u-0.0680}{0.0003u^8+0.0121u^7+0.1630u^6+0.9770u^5+3.3709u^4+7.7091u^3+5.9935u^2-5.3643u+2.4714}$
$\beta_4=5$	$f_{\theta^*}(u;\beta_4)=\frac{1.1340u^8+5.2921u^7+8.9497u^6+6.3207u^5+2.1223u^4+1.8374u^3+1.4766u^2-0.3334u-0.5300}{1.7459u^8+8.1478u^7+13.7791u^6+9.7314u^5+3.8413u^4+4.16754u^3+1.1539u^2-2.7925u+1.4449}$
$\beta_5=7$	$f_{\theta^*}(u;\beta_5)=\frac{7.2266e{-}5u^8+0.0031u^7+0.0486u^6+0.3395u^5+0.8249u^4-0.1620u^3+2.3615u^2-0.6630u-0.1760}{7.7078e{-}5u^8+0.0033u^7+0.0518u^6+0.3621u^5+0.8955u^4+0.1576u^3+4.0315u^2-3.2001u+0.9126}$



(a) $\beta=-1$ (b) $\beta=1$

(c) $\beta=3$ (d) $\beta=5$ (e) $\beta=7$

Fig. 10. The graphical results of identification of flux function $f_{\theta^*}(u;\beta)$ for $\beta=-1,1,3,5,7$. In each subplot, the solid red line is the true function, and the orange dashed line is the learned function.

$$f_W(u,\beta)=\frac{a_{00}+a_{01}u+a_{02}u^2+a_{03}u^3+a_{04}u^4+a_{05}u^5+\beta(a_{10}+a_{11}u+a_{12}u^2+a_{13}u^3+a_{14}u^4+a_{15}u^5)}{b_{00}+b_{01}u+b_{02}u^2+b_{03}u^3+b_{04}u^4+b_{05}u^5+\beta(b_{10}+b_{11}u+b_{12}u^2+b_{13}u^3+b_{14}u^4+b_{15}u^5)} \tag{41}$$

Box I.

Next, we verify that (43) is a model that can represent well the true family of flux functions (39). We test (43) on the validation set where $\beta=-3$ and $\beta=9$. Table 5 shows the results. Obviously, whether it is for $\beta=-3$ or for $\beta=9$, the error is relatively small, and meets the criteria we set $\Delta=0.0007$. We also test other candidate models on the validation set. The results are shown in Table 6. It is obvious that when $K=1$, $J_i=5$, $P=1$ and $Q_i=5$ the candidate model has the best performance. Therefore, we choose (42) and (43) as the preferred learned model and use it to test new initial states and parameters.

4.3. Results and discussions

In Fig. 11 we first show the learned flux function (43) (panel a) and the error between true flux function (39) and the learned (43) (panel b) with $\beta\in[-10,10]$ and $u\in[0,1]$. The absolute error of the two functions is less than 0.118 over the entire domain. Specifically, we set $\beta=-10,-7,-5,8,9,10$ separately to test the generalization ability of (43). In Fig. 12, we show the true and the learned function with different values of β. The model performs well for most parameter choice of β in $[-10,10]$, however, conduct worse when $\beta<-8$ although still within our acceptable range $\Delta=0.13$.

Finally, we compare the GNNs method with our approach. We use Rprop optimizer [43] to minimize the loss function with learning rate set to $lr=10^{-6}$ for 5000 iterations and batch size set to 5. Fig. 13 shows the solutions of (1) with $\beta=-7$ and $\beta=9$ using the two different methods. The proposed method enables quite a nice approximation of the true predicted behavior whereas the GNNs fail to capture it, most likely, since it has not been constructed to handle discontinuous behavior.

5. Conclusion

In this paper we have developed a framework that combines the recently proposed ConsLaw-Net with LRNN to learn the functional form of the two variable function $f(u,\beta)$ involved in the

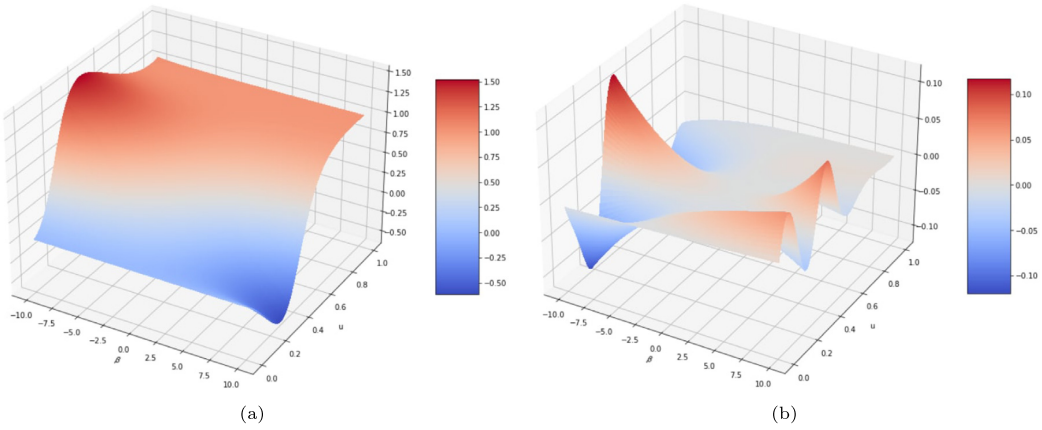


Fig. 11. (a) Flux function (43). (b) The difference between the flux function (39) and (43).

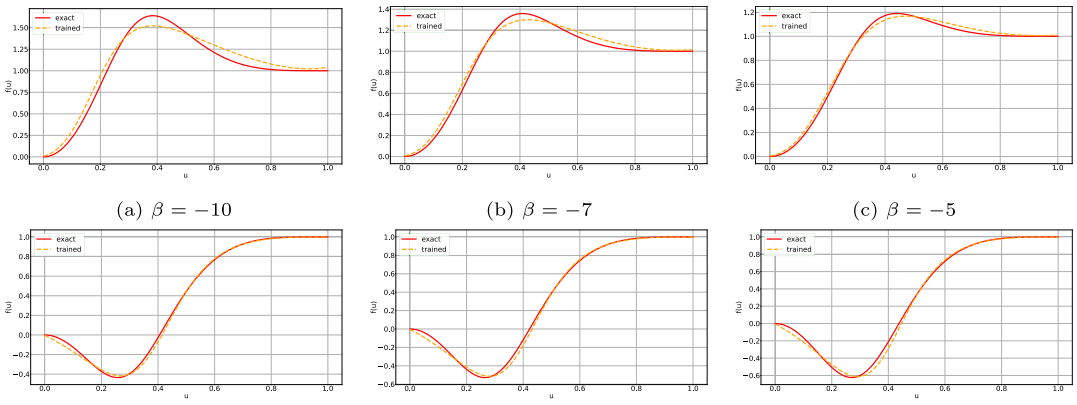


Fig. 12. The generalization ability of the model (43) for $\beta = -10, -7, -5, 8, 9, 10$. In each subplot, the red solid line is the true function, and the orange dashed line is the learned function.

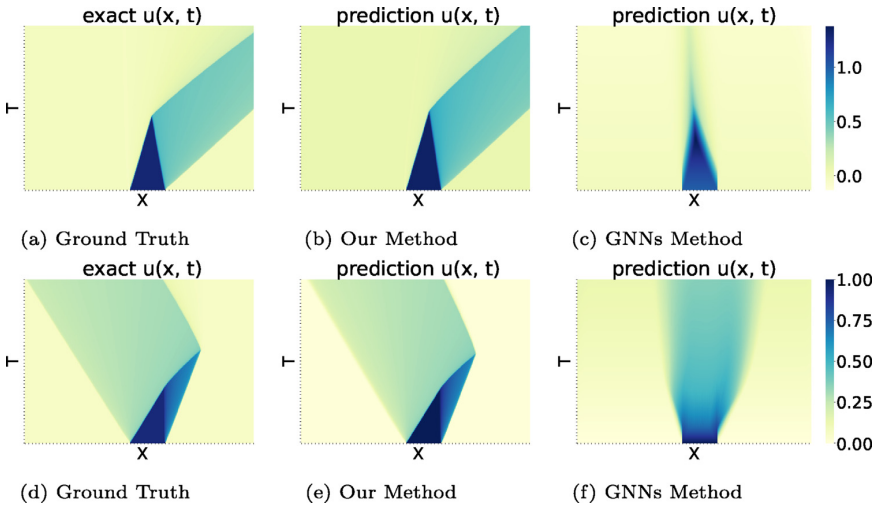


Fig. 13. The compared results of our method and the GNNs method. We compare the two methods at $\beta = -7$(top row) and $\beta = 9$(bottom row) based on initial state $u_0 = \begin{cases} 1.0, & \text{if } x \in [4.5, 6] \\ 0.0, & \text{otherwise} \end{cases}$. Left: True solutions of (1). Middle: Solutions generated by our method. Right: Solutions generated by the GNNs method.

Table 5
The results of (43) on the validation data $\beta = -3$ and $\beta = 9$. The value of Error is the MSE between true and generated solutions of (1) at times (40). We also plot the solution at time point $t = 0.3, 0.6, 0.9$ respectively. In each subplot, the solid red line is the true solution, and the blue dashed line is the solution generated by (43) based on initial state $u_0 = \begin{cases} 1.0, & \text{if } x \in [4.5, 6] \\ 0.0, & \text{otherwise} \end{cases}$.

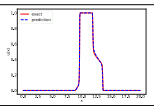
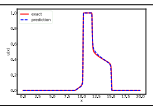
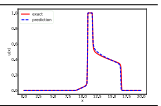
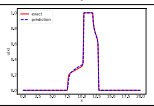
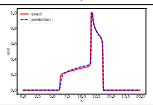
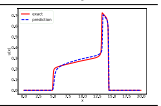
β	Error	t = 0.3	t = 0.6	t = 0.9
$\beta = -3$	0.0002			
$\beta = 9$	0.0006			

Table 6
The results of other candidate models on the validation set. The value is the MSE between true and predicted solutions of (1) at times (40) for four different initial states given by $u^0_{\beta=-3,9} = \begin{cases} 1.0, & \text{if } x \in [4.5, 6] \\ 0.0, & \text{otherwise} \end{cases}$, $u^1_{\beta=-3,9} = \begin{cases} 0.9, & \text{if } x \in [3.5, 5] \\ 0.0, & \text{otherwise} \end{cases}$, $u^2_{\beta=-3,9} = \begin{cases} 0.8, & \text{if } x \in [5, 6.5] \\ 0.0, & \text{otherwise} \end{cases}$ and $u^3_{\beta=-3,9} = \begin{cases} 0.7, & \text{if } x \in [3, 6] \\ 0.0, & \text{otherwise} \end{cases}$, respectively.

β	$K = 1$, $J_i = 6$	$K = 1$, $J_i = 7$	$K = 1$, $J_i = 8$	$K = 1$, $J_i = 3$, $P = 1$, $Q_i = 3$	$K = 1$, $J_i = 4$, $P = 1$, $Q_i = 4$
$\beta = -3$	0.0014	0.0005	0.0005	0.0210	0.0003
$\beta = 9$	0.0038	0.0036	0.0026	0.0146	0.0008

conservation law (1). More precisely, the method is composed of the following two steps: (i) learn $f(u; \beta_i)$ for a few selected β_i in an interval of interest; (ii) Combine the obtained $f(u; \beta_i)$ with LRNN to learn the full two-variable function $f(u, \beta)$. As a by product of this method we have improved the ConsLaw-Net by employing an entropy consistent numerical scheme which relies on local information (relatively the discrete spatial grid) about the unknown flux function $f(u, \beta)$. This makes the resulting ConsLaw-Net more generic and useful for challenging real-world data. The well-known challenge with identification of a nonlinear flux function from data due to appearance of discontinuities and lack of uniqueness (i.e., many different flux functions can fit with the observation data), is dealt with by relying on an entropy consistent numerical scheme in combination with a small set of more or less randomly selected initial data. We experimentally demonstrate the effectiveness of our method when applied to synthetic data generated from two different families of flux functions $f(u, \beta)$ given by (34) and (39), respectively.

Our method is sensitive to the selection of the initial states, as initial states affect the distribution of the observed data. If the data is not sufficiently evenly distributed, it may lead to failure in the first step of the learning process. The optimization of the second step is heavily dependent on the results of the first step, and a small error in the first step may amplify this error in the second step. However, in a setting where the observation data comes from measurements of experimental data our approach gives a systematic way to try to learn an underlying conservation law that can explain the observation data. Before using the learned conservation law for prediction one must be aware that enough variation in initial data has been provided.

Overall, our approach based on ConsLaw-Net and LRNN has shown a good ability learn the unknown flux function with variable parameters $f(u, \beta)$. Possible further extensions can be to explore the method in higher dimensional spaces and consider other types of observation data than we have used in this work. An interesting direction for further investigations is to see if there is room for including in the loss function some explicit characteristics of the entropy solution that will make the identification of the flux function more efficient.

CRediT authorship contribution statement

Qing Li: Methodology, Coding, Investigation, Writing, Visualization. **Jiahui Geng:** Software, Writing – review & editing. **Steinar Evje:** Coding, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Appendix

A.1. The initial states used to train $f(u; \beta)$ in Section 3.1

See Table 7.

A.2. The initial states used to train $f(u; \beta)$ in Section 4.1

See Table 8.

Table 7
The initial states used to train $f(u;\beta)$ in Section 3.1.

$$u^0_{\beta=10}=\begin{cases}1.0, & \text{if } x\in[0,3]\\ 0, & \text{otherwise}\end{cases} \quad u^0_{\beta=130}=\begin{cases}1.0, & \text{if } x\in[4.5,6]\\ 0.3, & \text{otherwise}\end{cases} \quad u^1_{\beta=130}=\begin{cases}0.8, & \text{if } x\in[5,6.5]\\ 0.2, & \text{otherwise}\end{cases} \quad u^2_{\beta=130}=\begin{cases}0.85, & \text{if } x\in[4,7]\\ 0, & \text{otherwise}\end{cases}$$

$$u^3_{\beta=130}=\begin{cases}0.95, & \text{if } x\in[0,3]\\ 0, & \text{otherwise}\end{cases} \quad u^4_{\beta=130}=\begin{cases}0.7, & \text{if } x\in[3,6]\\ 0, & \text{otherwise}\end{cases} \quad u^5_{\beta=130}=\begin{cases}0.9, & \text{if } x\in[3.5,5]\\ 0, & \text{otherwise}\end{cases} \quad u^0_{\beta=200}=\begin{cases}1.0, & \text{if } x\in[4.5,6]\\ 0.3, & \text{otherwise}\end{cases}$$

$$u^1_{\beta=200}=\begin{cases}0.8, & \text{if } x\in[5,6.5]\\ 0.2, & \text{otherwise}\end{cases} \quad u^2_{\beta=200}=\begin{cases}0.85, & \text{if } x\in[4,7]\\ 0, & \text{otherwise}\end{cases} \quad u^3_{\beta=200}=\begin{cases}0.95, & \text{if } x\in[0,3]\\ 0, & \text{otherwise}\end{cases} \quad u^4_{\beta=200}=\begin{cases}0.7, & \text{if } x\in[3,6]\\ 0, & \text{otherwise}\end{cases}$$

$$u^5_{\beta=200}=\begin{cases}0.9, & \text{if } x\in[3.5,5]\\ 0, & \text{otherwise}\end{cases} \quad u^0_{\beta=300}=\begin{cases}1.0, & \text{if } x\in[4.5,6]\\ 0.3, & \text{otherwise}\end{cases} \quad u^1_{\beta=300}=\begin{cases}0.8, & \text{if } x\in[5,6.5]\\ 0.2, & \text{otherwise}\end{cases} \quad u^2_{\beta=300}=\begin{cases}0.85, & \text{if } x\in[4,7]\\ 0, & \text{otherwise}\end{cases}$$

$$u^3_{\beta=300}=\begin{cases}0.95, & \text{if } x\in[0,3]\\ 0, & \text{otherwise}\end{cases} \quad u^4_{\beta=300}=\begin{cases}0.7, & \text{if } x\in[3,6]\\ 0, & \text{otherwise}\end{cases} \quad u^5_{\beta=300}=\begin{cases}0.9, & \text{if } x\in[3.5,5]\\ 0, & \text{otherwise}\end{cases} \quad u^6_{\beta=300}=\begin{cases}0.6, & \text{if } x\in[3.5,5]\\ 0, & \text{otherwise}\end{cases}$$

$$u^7_{\beta=300}=\begin{cases}0.4, & \text{if } x\in[3.5,5]\\ 0, & \text{otherwise}\end{cases} \quad u^8_{\beta=300}=\begin{cases}0.35, & \text{if } x\in[3.5,5]\\ 0, & \text{otherwise}\end{cases}$$

Table 8
The initial states used to train $f(u;\beta)$ in Section 4.1.

$$u^0_{\beta=-1}=\begin{cases}0.8, & \text{if } x\in[4,6]\\ 0, & \text{otherwise}\end{cases} \quad u^1_{\beta=-1}=\begin{cases}1.0, & \text{if } x\in[4,6]\\ 0, & \text{otherwise}\end{cases} \quad u^2_{\beta=-1}=\begin{cases}0.9, & \text{if } x\in[3,5]\\ 0, & \text{otherwise}\end{cases} \quad u^3_{\beta=-1}=\begin{cases}0.95, & \text{if } x\in[2.5,4.5]\\ 0.3, & \text{otherwise}\end{cases}$$

$$u^4_{\beta=-1}=\begin{cases}0.85, & \text{if } x\in[2.5,4.5]\\ 0.3, & \text{otherwise}\end{cases} \quad u^0_{\beta=1}=\begin{cases}0.8, & \text{if } x\in[4,6]\\ 0, & \text{otherwise}\end{cases} \quad u^1_{\beta=1}=\begin{cases}1.0, & \text{if } x\in[4,6]\\ 0, & \text{otherwise}\end{cases} \quad u^2_{\beta=1}=\begin{cases}0.9, & \text{if } x\in[3,5]\\ 0, & \text{otherwise}\end{cases}$$

$$u^3_{\beta=1}=\begin{cases}0.7, & \text{if } x\in[2.5,4.5]\\ 0.2, & \text{otherwise}\end{cases} \quad u^4_{\beta=1}=\begin{cases}0.6, & \text{if } x\in[2.5,4.5]\\ 0.2, & \text{otherwise}\end{cases} \quad u^5_{\beta=1}=\begin{cases}0.5, & \text{if } x\in[2.5,4.5]\\ 0.2, & \text{otherwise}\end{cases} \quad u^0_{\beta=3}=\begin{cases}0.8, & \text{if } x\in[4,6]\\ 0, & \text{otherwise}\end{cases}$$

$$u^1_{\beta=3}=\begin{cases}1.0, & \text{if } x\in[4,6]\\ 0, & \text{otherwise}\end{cases} \quad u^2_{\beta=3}=\begin{cases}0.9, & \text{if } x\in[3,5]\\ 0, & \text{otherwise}\end{cases} \quad u^3_{\beta=3}=\begin{cases}0.95, & \text{if } x\in[2.5,4.5]\\ 0.3, & \text{otherwise}\end{cases} \quad u^4_{\beta=3}=\begin{cases}0.85, & \text{if } x\in[2.5,4.5]\\ 0.3, & \text{otherwise}\end{cases}$$

$$u^5_{\beta=3}=\begin{cases}1.0, & \text{if } x\in[2.5,4.5]\\ 0.2, & \text{otherwise}\end{cases} \quad u^6_{\beta=3}=\begin{cases}0.9, & \text{if } x\in[2.5,4.5]\\ 0.2, & \text{otherwise}\end{cases} \quad u^0_{\beta=5}=\begin{cases}0.8, & \text{if } x\in[4,6]\\ 0, & \text{otherwise}\end{cases} \quad u^1_{\beta=5}=\begin{cases}1.0, & \text{if } x\in[4,6]\\ 0, & \text{otherwise}\end{cases}$$

$$u^2_{\beta=5}=\begin{cases}0.9, & \text{if } x\in[3,5]\\ 0, & \text{otherwise}\end{cases} \quad u^3_{\beta=5}=\begin{cases}0.95, & \text{if } x\in[2.5,4.5]\\ 0.3, & \text{otherwise}\end{cases} \quad u^4_{\beta=5}=\begin{cases}0.85, & \text{if } x\in[2.5,4.5]\\ 0.3, & \text{otherwise}\end{cases} \quad u^0_{\beta=7}=\begin{cases}0.8, & \text{if } x\in[4,6]\\ 0, & \text{otherwise}\end{cases}$$

$$u^1_{\beta=7}=\begin{cases}1.0, & \text{if } x\in[4,6]\\ 0, & \text{otherwise}\end{cases} \quad u^2_{\beta=7}=\begin{cases}0.9, & \text{if } x\in[3,5]\\ 0, & \text{otherwise}\end{cases} \quad u^3_{\beta=7}=\begin{cases}0.95, & \text{if } x\in[2.5,4.5]\\ 0.3, & \text{otherwise}\end{cases} \quad u^4_{\beta=7}=\begin{cases}0.85, & \text{if } x\in[2.5,4.5]\\ 0.3, & \text{otherwise}\end{cases}$$

References

[1] D. Pardo, L. Demkowicz, C. Torres-Verdin, M. Paszynski, A self-adaptive goal-oriented hp-finite element method with electromagnetic applications. Part II: Electrodynamics, Comput. Methods Appl. Mech. Engrg. 196 (37–40) (2007) 3585–3597.

[2] Thomas D Economon, Francisco Palacios, Sean R Copeland, Trent W Lukaczyk, Juan J Alonso, SU2: An open-source suite for multiphysics simulation and design, AIAA J. 54 (3) (2016) 828–846.

[3] Ravi Ramamurti, William Sandberg, Simulation of flow about flapping airfoils using finite element incompressible flow solver, AIAA J. 39 (2) (2001) 253–260.

[4] Peter Bauer, Alan Thorpe, Gilbert Brunet, The quiet revolution of numerical weather prediction, Nature 525 (7567) (2015) 47–55.

[5] Christoph Schwarzbach, Ralph-Uwe Börner, Klaus Spitzer, Three-dimensional adaptive higher order finite element simulation for geo-electromagnetics—a marine CSEM example, Geophys. J. Int. 187 (1) (2011) 63–74.

[6] Maziar Raissi, Paris Perdikaris, George 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) 686–707.

[7] Zichao Long, Yiping Lu, Xianzhong Ma, Bin Dong, Pde-net: Learning PDEs from data, in: International Conference on Machine Learning, PMLR, 2018, pp. 3208–3216.

[8] Zichao Long, Yiping Lu, Bin Dong, PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network, J. Comput. Phys. 399 (2019) 108925.

[9] Hans Joakim Skadsem, Steinar Kragset, A numerical study of density-unstable reverse circulation displacement for primary cementing, . Energy Resour. Technol. 144 (12) (2022) 123008.

[10] Olga Fuks, Hamdi A. Tchelepi, Limitations of physics informed machine learning for nonlinear two-phase transport in porous media, J. Mach. Learn. Model. Comput. 1 (1) (2020).

[11] Qing Li, Steinar Evje, Learning the nonlinear flux function of a hidden scalar conservation law from data, Netw. Heterog. Media 18 (2023).

[12] R.J. LeVeque, Finite volume methods for hyperbolic problems, in: Cambridge Texts in Applied Mathematics, 2007.

[13] J.W. Thomas, Numerical partial differential equations. conservation laws and elliptic equations, in: Texts in Applied Mathematics 33, 1999.

[14] J.S. Hesthaven, Numerical methods for conservation laws. From analysis to algorithms, SIAM. Comput. Sci. Eng. (2017).

[15] D. Kröener, Numerical schemes for conservation laws, in: Wiley-Teubner Series Advances in Numerical Mathematics, 1997.

[16] Siddhartha Mishra, U. Fjordholm, R. Abgrall, Numerical methods for conservation laws and related equations, in: Lecture Notes for Numerical Methods for Partial Differential Equations, ETH, vol. 57, 2019, p. 58.

[17] Valerii Iakovlev, Markus Heinonen, Harri Lähdesmäki, Learning continuous-time PDEs from sparse data with graph neural networks, in: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021, OpenReview.net, 2021.

[18] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, Peter W. Battaglia, Learning mesh-based simulation with graph networks, in: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021, OpenReview.net, 2021.

[19] Qingqing Zhao, David B. Lindell, Gordon Wetzstein, Learning to solve PDE-constrained inverse problems with graph networks, in: Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, Sivan Sabato (Eds.), International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA, in: Proceedings of Machine Learning Research, vol. 162, PMLR, 2022, pp. 26895–26910.

[20] Helge Holden, Fabio Simone Priuli, Nils Henrik Risebro, On an inverse problem for scalar conservation laws, Inverse Problems 30 (3) (2014) 035015.
[21] María Cristina Bustos, EM Tory, Raimund Bürger, F Concha, Sedimentation and thickening: Phenomenological foundation and mathematical theory, Vol. 8, Springer Science & Business Media, 1999.
[22] Stefan Diehl, Estimation of the batch-settling flux function for an ideal suspension from only two experiments, Chem. Eng. Sci. 62 (17) (2007) 4589–4601.
[23] Raimund Bürger, Stefan Diehl, Convexity-preserving flux identification for scalar conservation laws modelling sedimentation, Inverse Problems 29 (4) (2013) 045008.
[24] Raimund Bürger, Julio Careaga, Stefan Diehl, Flux identification of scalar conservation laws from sedimentation in a cone, IMA J. Appl. Math. 83 (3) (2018) 526–552.
[25] Stefan Diehl, Numerical identification of constitutive functions in scalar nonlinear convection–diffusion equations with application to batch sedimentation, Appl. Numer. Math. 95 (2015) 154–172.
[26] Lukas Mosser, Olivier Dubrule, Martin J. Blunt, Stochastic seismic waveform inversion using generative adversarial networks as a geological prior, Math. Geosci. 52 (1) (2020) 53–79.
[27] Qinglong He, Yanfei Wang, Reparameterized full-waveform inversion using deep neural networks, Geophysics 86 (1) (2021) V1–V13.
[28] Tiffany Fan, Kailai Xu, Jay Pathak, Eric Darve, Solving inverse problems in steady-state navier-stokes equations using deep neural networks, 2020, arXiv preprint arXiv:2008.13074.
[29] Hayden Schaeffer, Learning partial differential equations via data discovery and sparse optimization, Proc. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci. 473 (2197) (2017) 20160446.
[30] Sung Ha Kang, Wenjing Liao, Yingjie Liu, Ident: Identifying differential equations with numerical time evolution, J. Sci. Comput. 87 (2021) 1–27.
[31] Nils Thuerey, Konstantin Weißenow, Lukas Prantl, Xiangyu Hu, Deep learning methods for Reynolds-averaged Navier–Stokes simulations of airfoil flows, AIAA J. 58 (1) (2020) 25–36.
[32] Nils Wandel, Michael Weinmann, Reinhard Klein, Learning incompressible fluid dynamics from scratch - towards fast, differentiable fluid models that generalize, in: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021, OpenReview.net, 2021.
[33] Siddhartha Mishra, A machine learning framework for data driven acceleration of computations of differential equations, Mathematics in Engineering 1 (2019) 118–146.
[34] Georg Martius, Christoph H. Lampert, Extrapolation and learning equations, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings, OpenReview.net, 2017.
[35] Subham Sahoo, Christoph Lampert, Georg Martius, Learning equations for extrapolation and control, in: International Conference on Machine Learning, PMLR, 2018, pp. 4442–4450.
[36] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, Jorge Nocedal, Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization, ACM Trans. Math. Softw. 23 (4) (1997) 550–560.
[37] Sebastian Ruder, An overview of gradient descent optimization algorithms, 2016, arXiv preprint arXiv:1609.04747.
[38] François James, M. Sepúlveda, Parameter identification for a model of chromatographic column, Inverse Problems 10 (6) (1994) 1299.
[39] François James, Mauricio Sepúlveda, Convergence results for the flux identification in a scalar conservation law, SIAM J. Control Optim. 37 (3) (1999) 869–891.
[40] Helge Holden, Nils Henrik Risebro, Front Tracking for Hyperbolic Conservation Laws, Vol. 152, Springer, 2015.
[41] Ilya Loshchilov, Frank Hutter, Decoupled weight decay regularization, in: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, la, USA, May 6-9, 2019, OpenReview.net, 2019.
[42] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, George E Dahl, Neural message passing for quantum chemistry, in: International Conference on Machine Learning, PMLR, 2017, pp. 1263–1272.
[43] Martin Riedmiller, Heinrich Braun, Rprop-a fast adaptive learning algorithm, in: Proc. of ISCIS VII, Universitat, Citeseer, 1992.

Paper V:
An Alternating Flux Learning Method for Multidimensional Nonlinear Conservation Laws

Qing Li, Steinar Evje

Under Review in SIAM Journal on Scientific Computing

This paper is not included in the repository because it's still under review.

Paper VI:
Learning the Flux and Diffusion Function for Degenerate Convection-Diffusion Equations Using Different Types of Observations

Qing Li, Steinar Evje

Under Review in BIT Numerical Mathematics

This paper is not included in the repository because it's still under review.