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 $\lambda_1$ and $\lambda_2$ 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 $\lambda_2$ 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

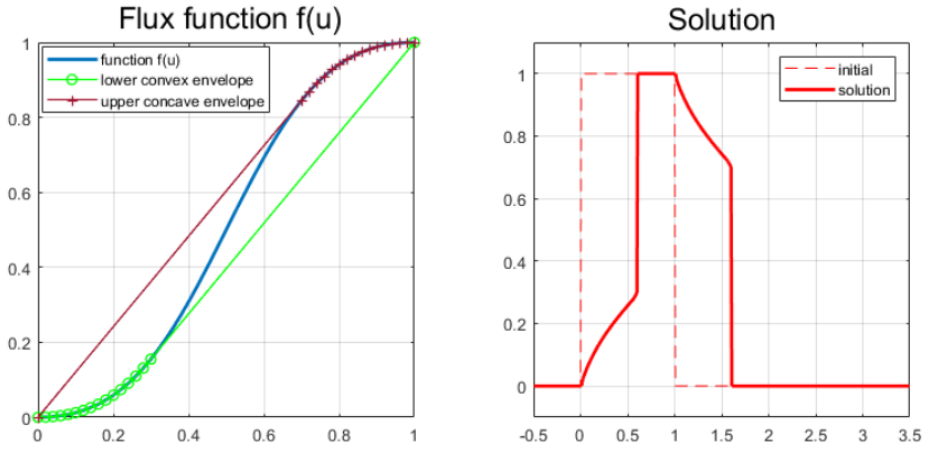



**Figure 1.** Left: Example of nonlinear flux function $f(u) = \frac{u^2}{u^2+(1-u)^2}$ (blue curve). Upper concave envelope (brown curve) and lower convex envelope (green curve) are also included. Right: The solution of Eq (1.4) at time $T = 0.5$ is shown (red solid curve) together with its initial 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., $\phi$ 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 $(\eta, q)$ 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 $(\eta, q)$. 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 $\Delta t$ for a given $\Delta x$ 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 $\Delta t$ 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 $U_{\text{sub}}$) 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 $N_{\text{obs}}$ of collected spatial-dependent data is relatively sparse. Also the number of local time steps (of length $\Delta t$) 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 $\Delta t$ 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 $\Delta t$ 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 $U_{\text{sub}}$ 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**: $\Delta t$: 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.

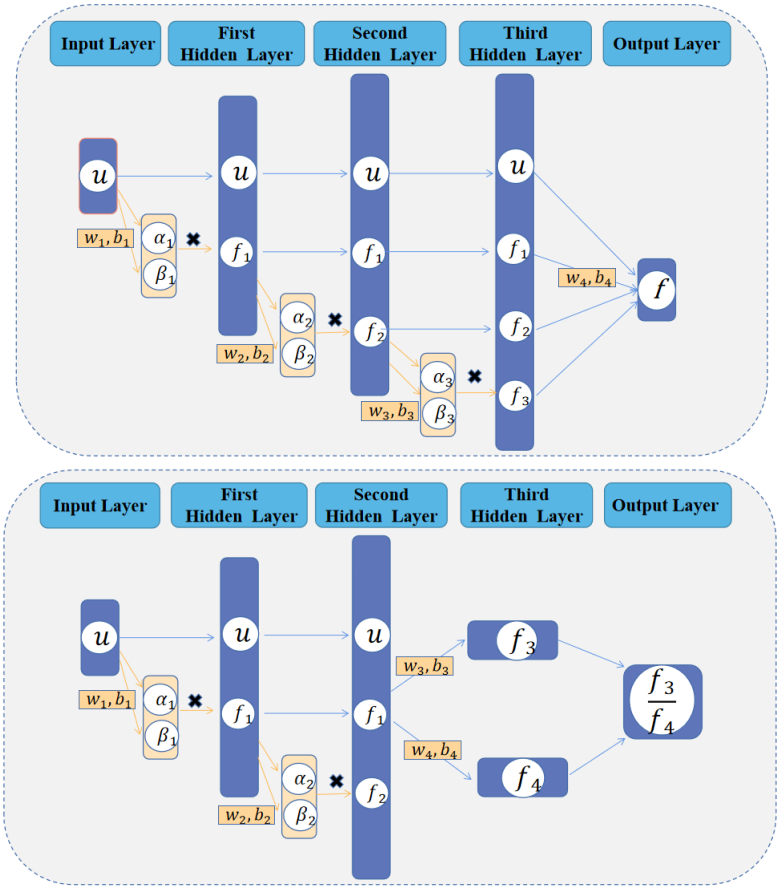



**Figure 2. Top**. The framework of S-Net-M for multiplicative function. **Bottom**. The framework of S-Net-D for division function.

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 $\alpha_1$ and $\beta_1$.

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

These two elements of $\alpha_1$ and $\beta_1$ 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 $\alpha_3$ and $\beta_3$ 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 $\alpha_3$ and $\beta_3$ 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 $\theta$ 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 $\theta$ 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.

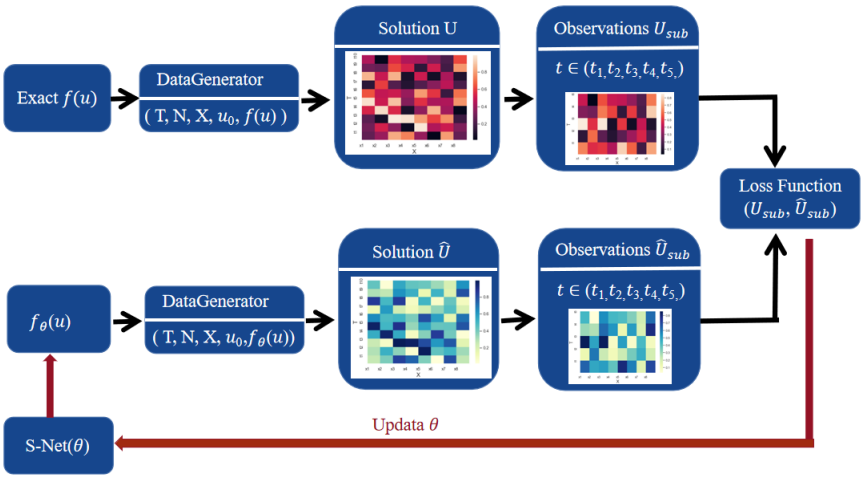



**Figure 3.** Schematic diagram of the framework.

**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; $\theta_0$: initial parameters of S-Net; $\theta$: 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 $\theta^*$;

$U = DataGenerator(T, N_x, L, u_0, f(u))$
$U\_sub = \{u \in U | t \in T\_obs\}$
$\theta = \theta_0$
**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 $\theta$ 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 $\beta$ represents the balance between gravity (bouyancy) and viscous forces and, typically, $\beta \in [-200, 300]$. Different values of $\beta$ 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.

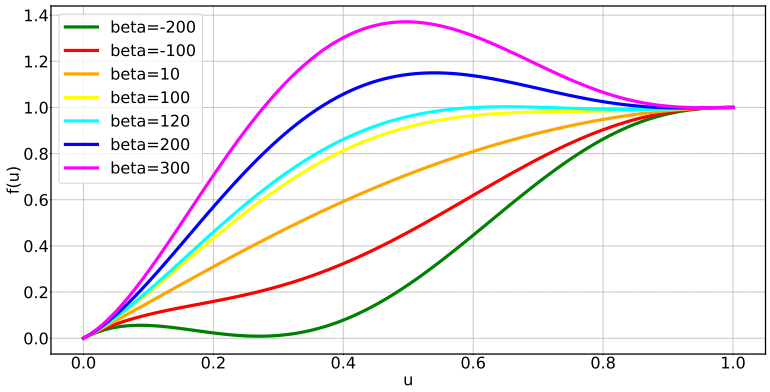



**Figure 4.** $f(u)$ with different values of $\beta$. Green, red, orange, yellow, cyan, blue and magenta line are generated by $\beta = -200$, $\beta = -100$, $\beta = 10$, $\beta = 100$, $\beta = 120$, $\beta = 200$, and $\beta = 300$, respectively.

In the following we generate synthetic data by specifying $\beta$ 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.

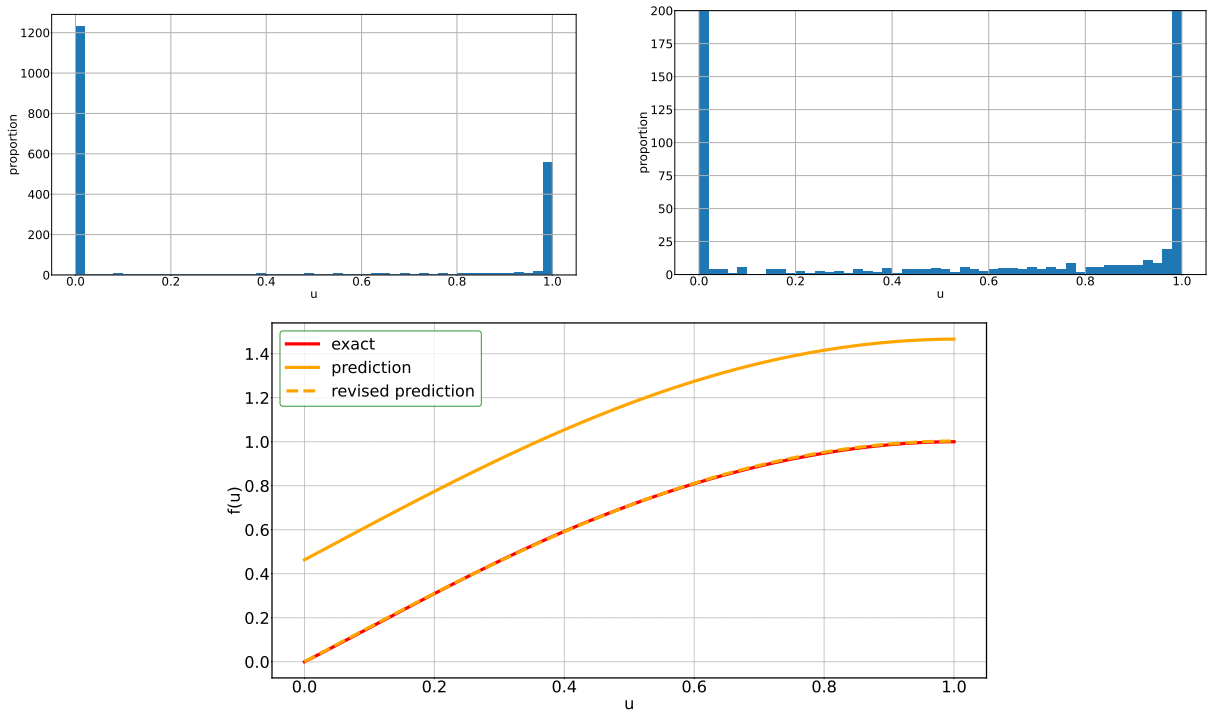



**Figure 5. Top**. The distribution of noise-free observations generated from combining $f(u;\beta = 10)$ with initial state Eq (3.3). Left: The distribution of all observation data. Right: The distribution between 0 and 200 (number of times values occur in the data). **Bottom**. The flux function $f(u;\beta = 10)$ (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 $\epsilon$ 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.

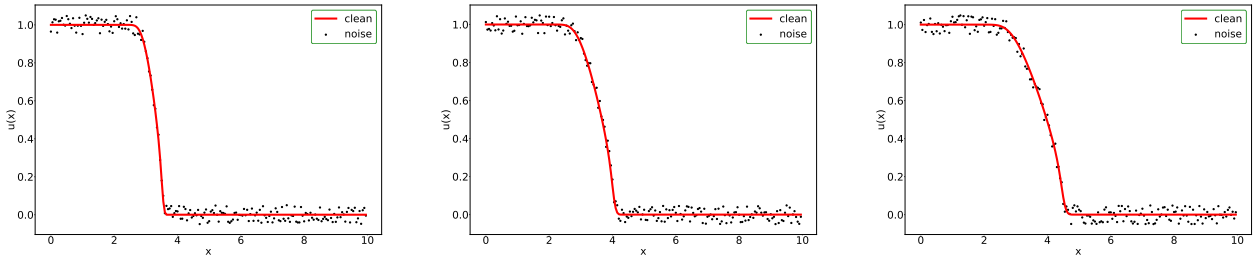
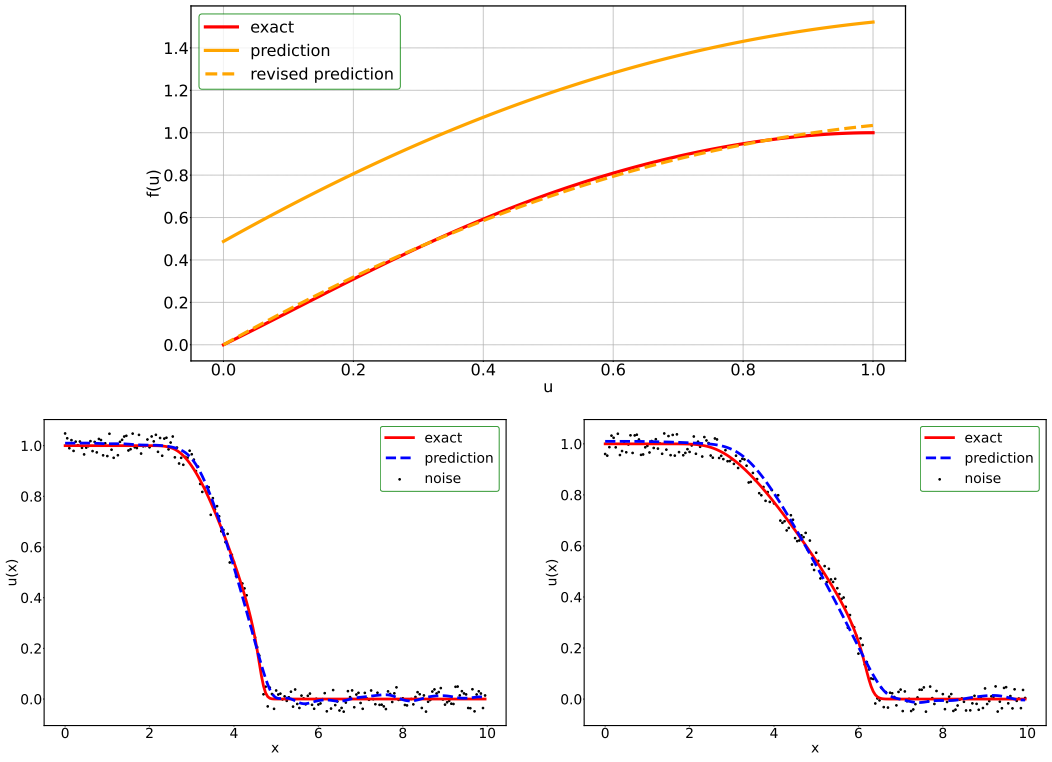



**Figure 6.** Noise-free and noisy data generated by the initial state (3.3) combined with the flux function $f(u;\beta = 10)$ at three time points: 0.3 (left), 0.6 (middle) and 0.9 (right).




**(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.

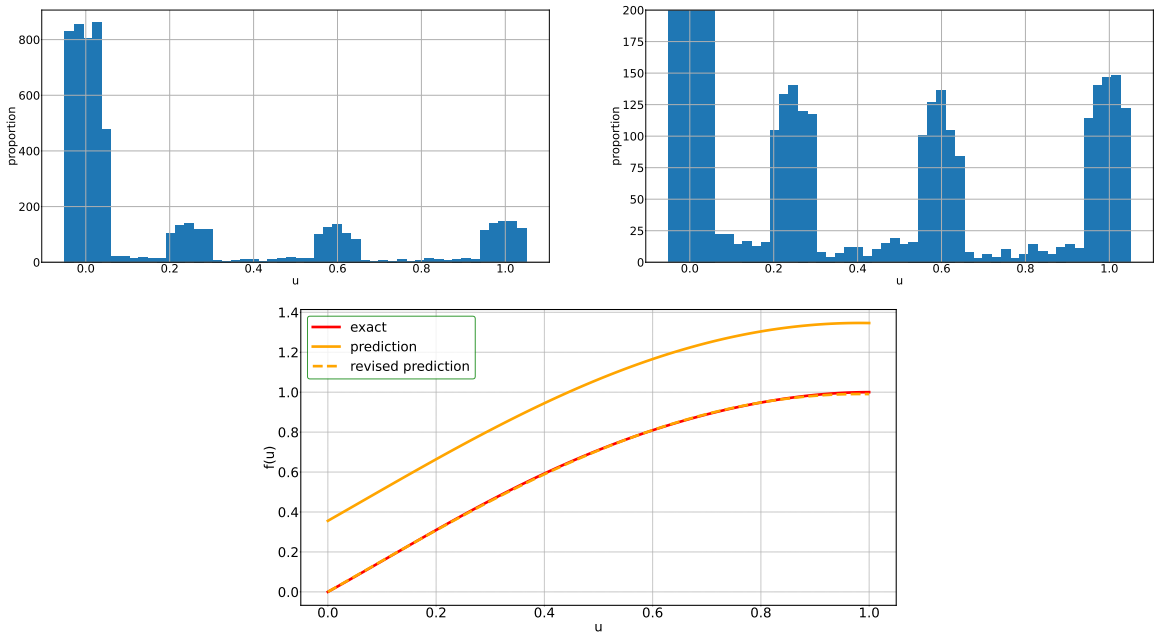



**Figure 8. Top**. The distribution of noisy observations generated by combining $f(u;\beta = 10)$ with initial states given in Table 3. Left: all observation data. Right: The distribution between 0 and 200. **Bottom**. True flux function $f(u;\beta = 10)$ (red solid line), $f_{\theta^*}(u)$ generated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

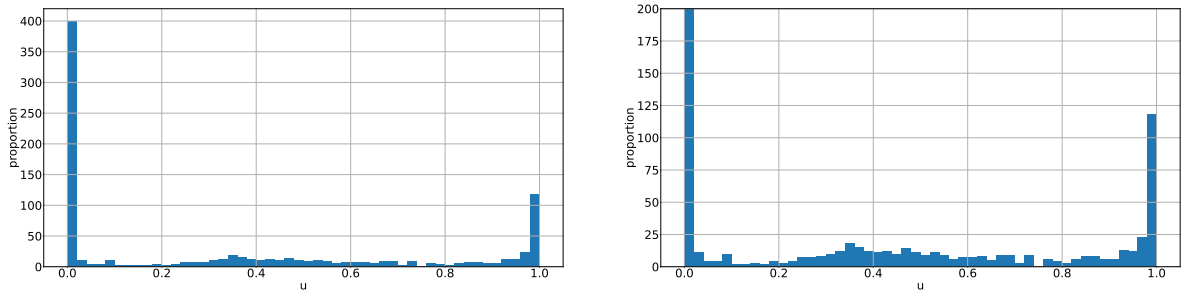
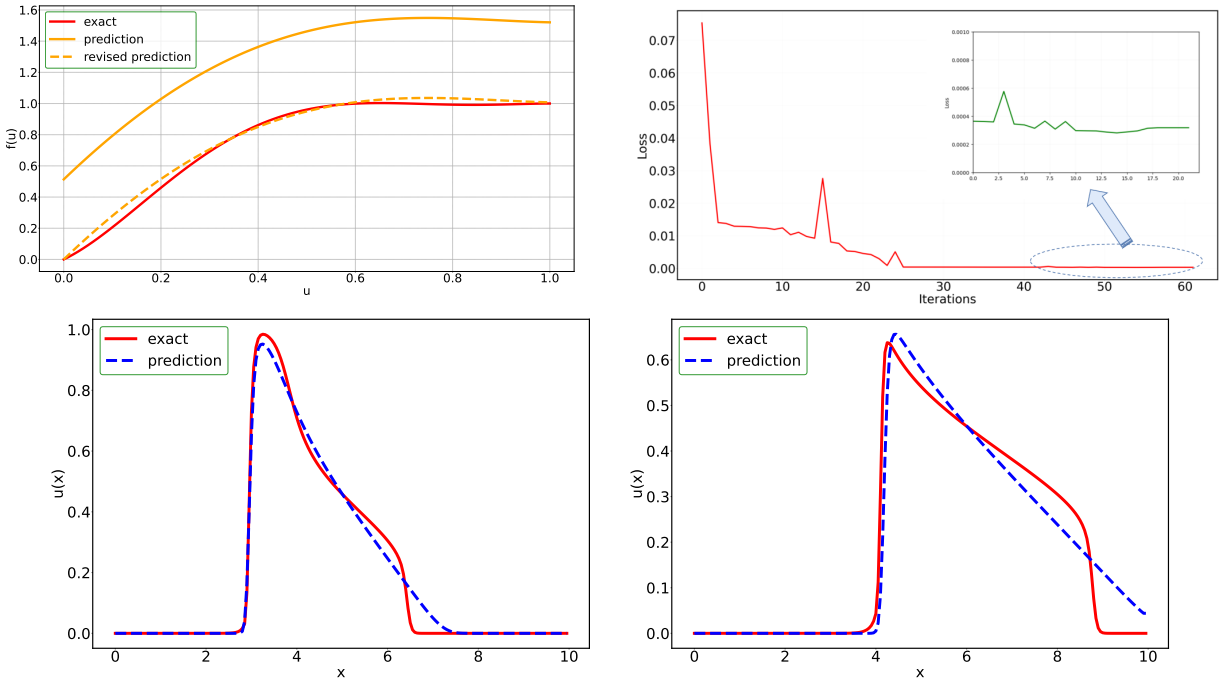



**Figure 9.** The distribution of clean observations generated from combining $f(u;\beta = 120)$ with initial state Eq (3.4). Left: The distribution of all observation data. Right: The distribution between 0 and 200.

### (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.

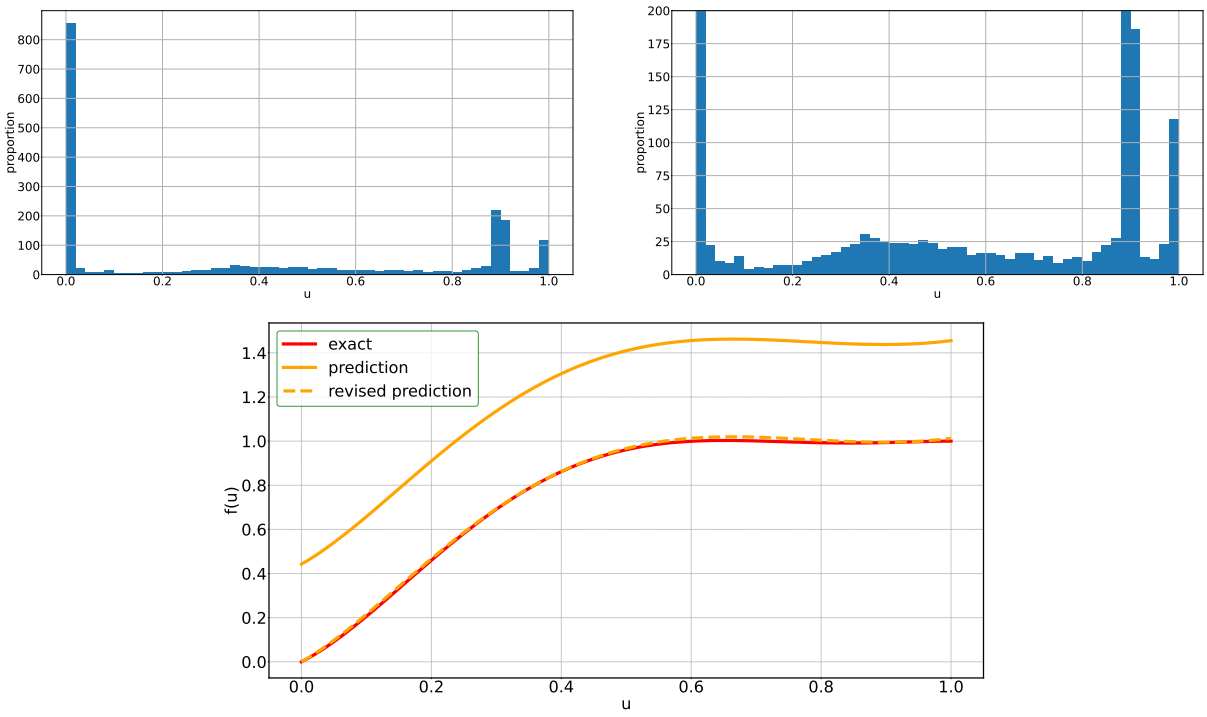



**Figure 11. Top**. The distribution of noise-free observations generated from combining $f(u;\beta = 120)$ with initial state given in Table 4. Left: The distribution of all observation data. Right: The distribution between 0 and 200. **Bottom**. The flux function $f(u;\beta = 120)$ where noise-free 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 | $f(u) = (-4.9213)u^3 + (2.3761)u + (1.7820)u^4 + (1.0587)u^2$ $+ (0.6508)u^5 + (0.5101) + (0.0653)u^6 + (0.0027)u^7$ $+ (3.8506e-05)u^8$ |

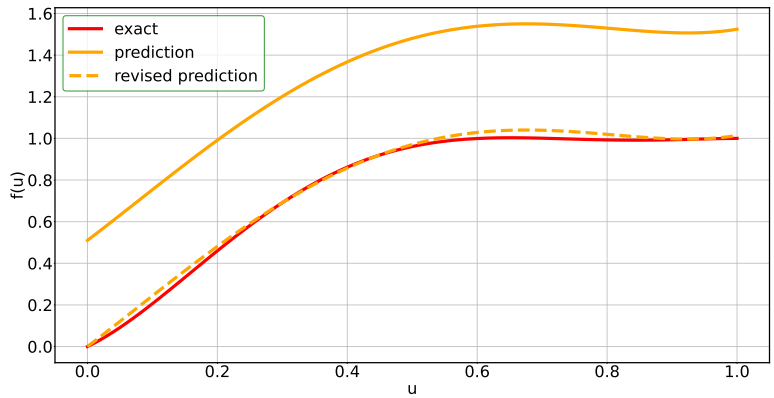



**Figure 12.** The flux function $f(u;\beta = 120)$ where noisy observations have been generated from initial data given in Table 4. The exact flux function $f(u;\beta = 120)$ (red solid line), $f_{\theta^*}(u)$ generated from ConsLaw-Net (orange solid line), and $f_{\theta^*}(u) - f_{\theta^*}(0)$ (orange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

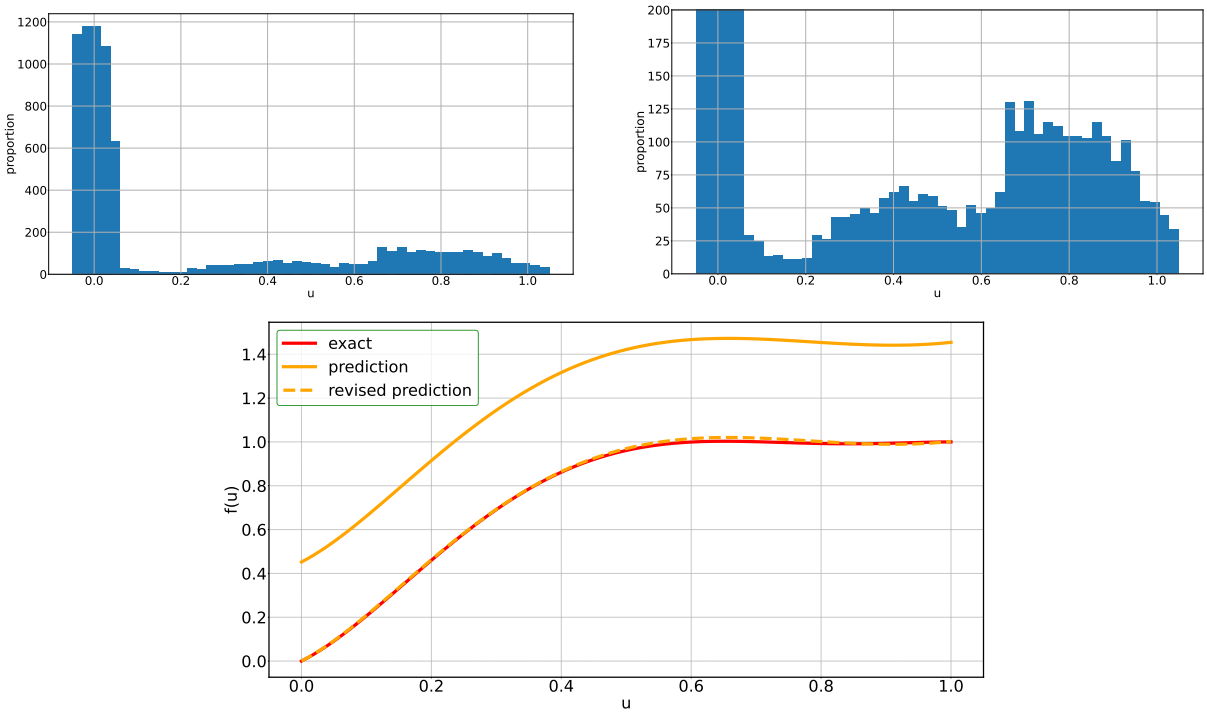



**Figure 13. Top**. The distribution of noisy observations generated by the four initial states in Table 7 by using $f(u;\beta = 120)$. **Bottom**. The flux function $f(u;\beta = 120)$ where noisy observations are generated based on initial data given in Table 7. The exact flux function $f(u;\beta = 120)$ (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 $(0.8, 1.0)$.

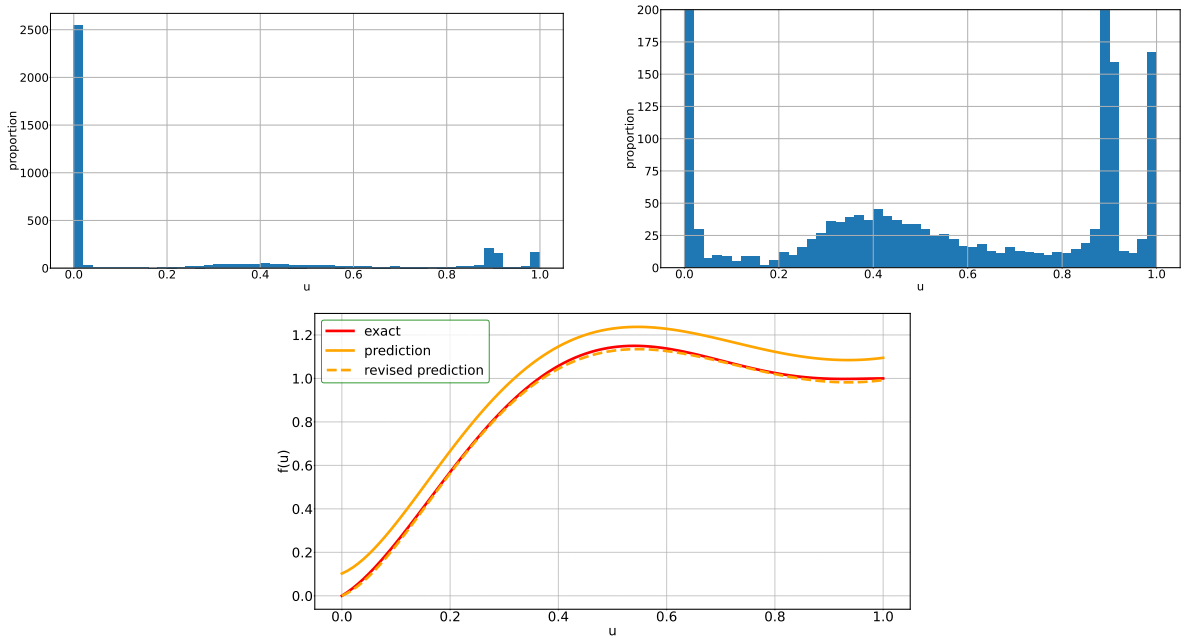



**Figure 14. Top**. The distribution of noise-free observations generated for the case with $f(u;\beta = 200)$ and initial data as in Table 9. Left. Distribution of all data. Right. Distribution of the quantity between 0 and 200. **Bottom**. The flux function $f(u;\beta = 200)$ where noise-free observations 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 $\beta = 300$.

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

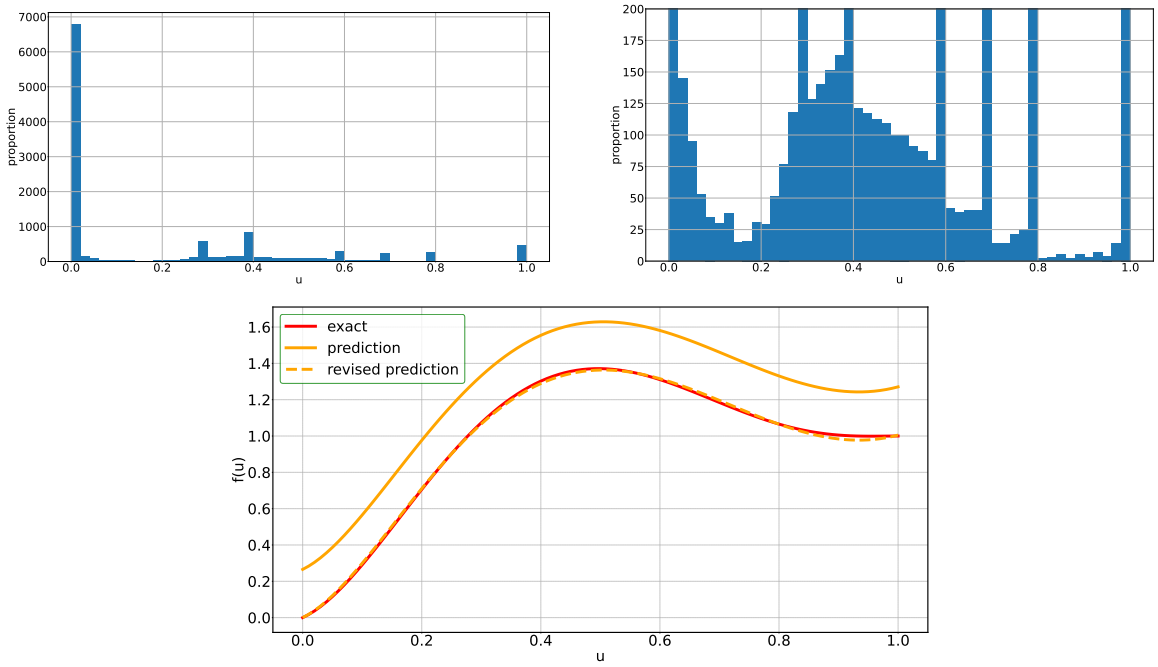



**Figure 15. Top**. The distribution of noise-free observations generated for the case with $f(u;\beta = 300)$ and initial data as in Table 10. Left. Distribution of all data. Right. Distribution of the quantity between 0 and 200. **Bottom**. The flux function $f(u;\beta = 300)$ where noise-free observations 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).

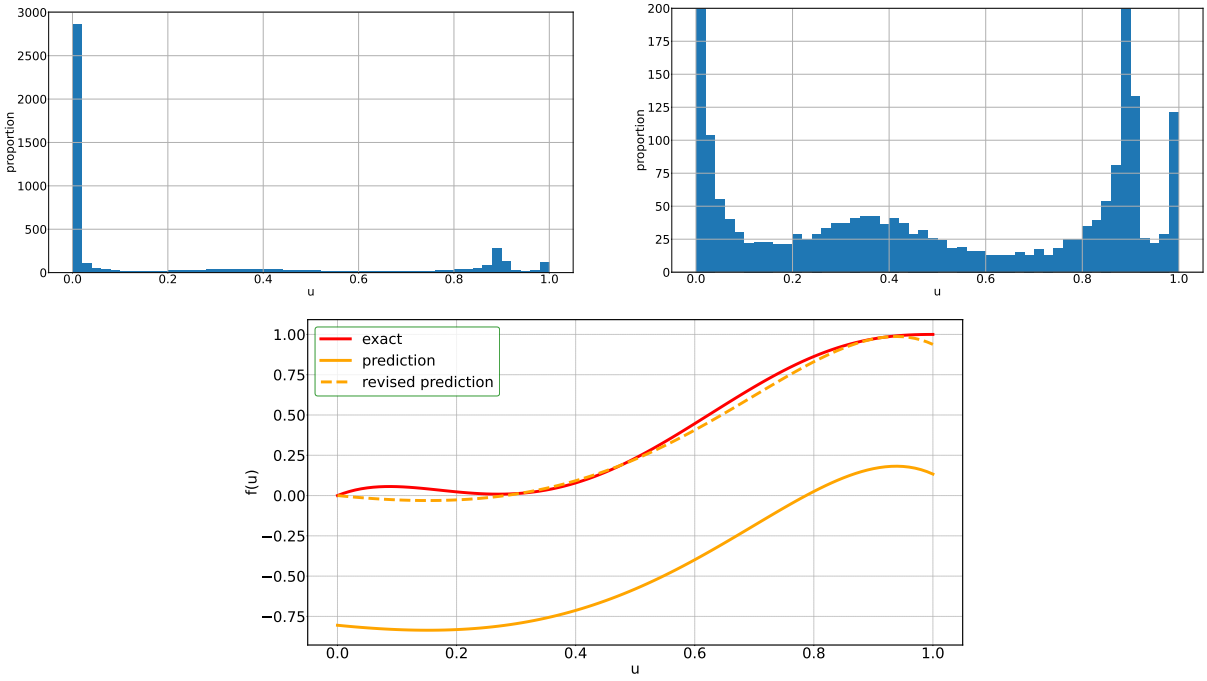



**Figure 16. Top**. The distribution of noise-free observations generated for the case with $f(u;\beta = -200)$ and initial data as in Table 12. Left. Distribution of all data. Right. Distribution of the quantity between 0 and 200. **Bottom**. The flux function $f(u;\beta = -200)$ where noise-free observations 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 $f(u;\beta = -200)$.

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

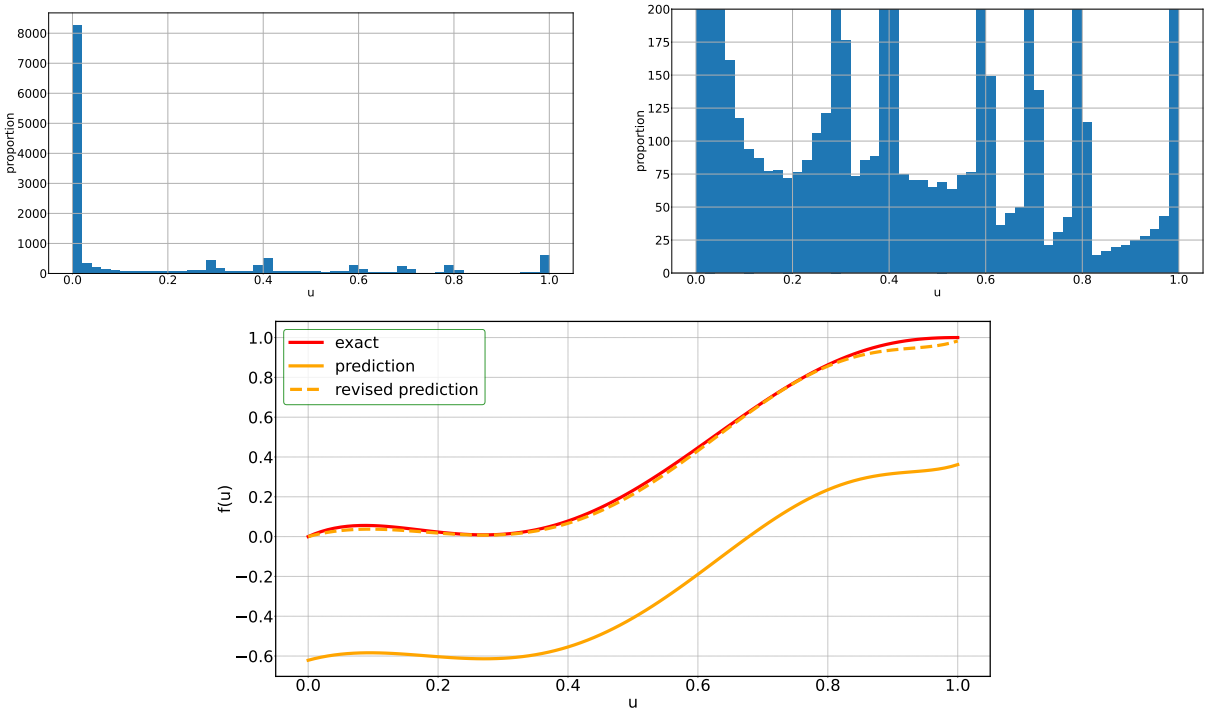



**Figure 17. Top**. The distribution of noise-free observations generated for the case with $f(u;\beta = -200)$ and initial data as in Table 13. Left. Distribution of all data. Right. Distribution of the quantity between 0 and 200. **Bottom**. The flux function $f(u;\beta = -200)$ where noise-free observations 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).

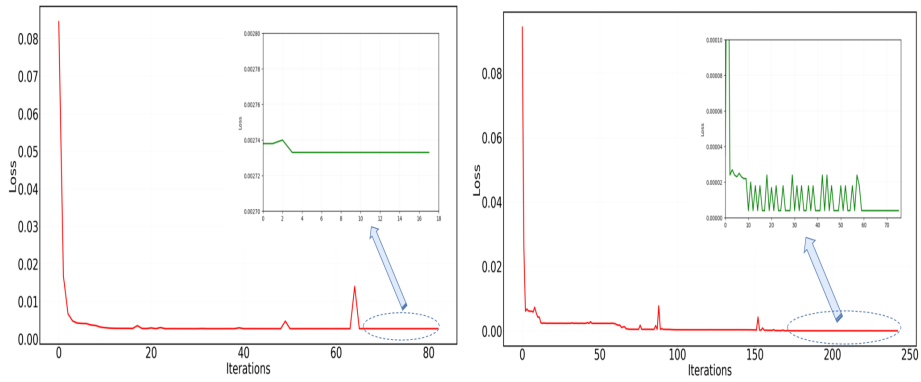



**Figure 18.** Left. Loss function based on S-Net-M. Right. Loss function based on S-Net-D.

### *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).

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

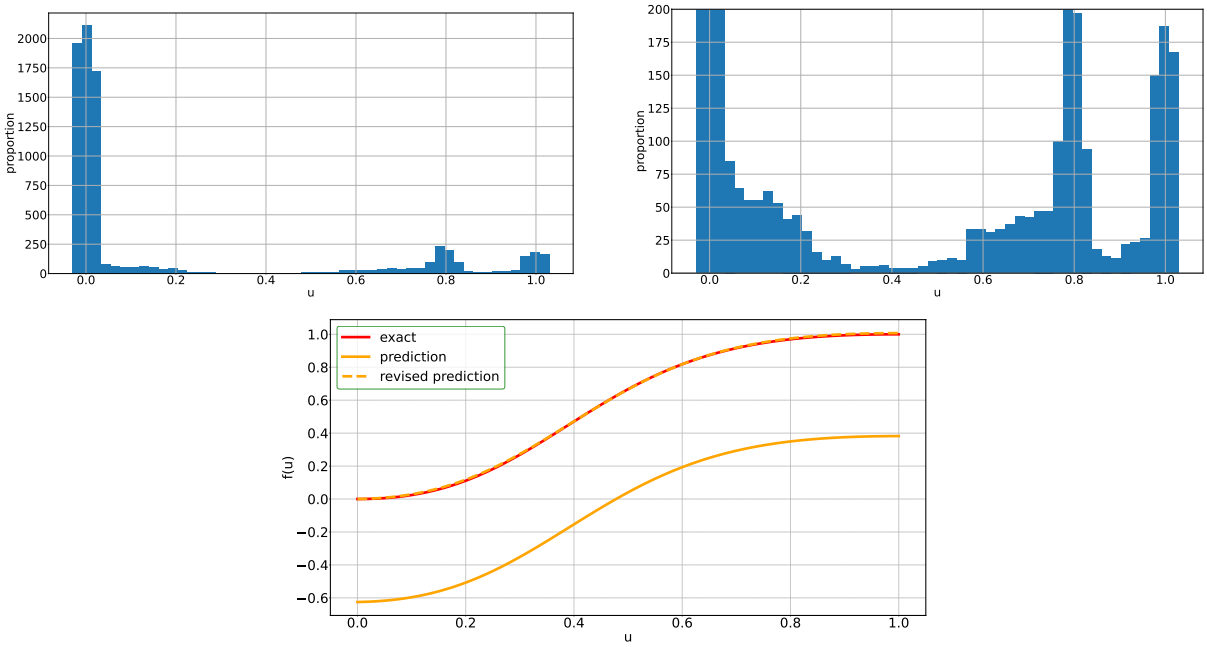



**Figure 19. Top**. The distribution of noisy observations generated for the flux function given by Eq (4.1) with $\beta = 1/2$ and initial data as in Table 15. Left. Distribution of all data. Right. Distribution of the quantity between 0 and 200. **Bottom**. The true flux function Eq (4.1) with $\beta = 1/2$ 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).

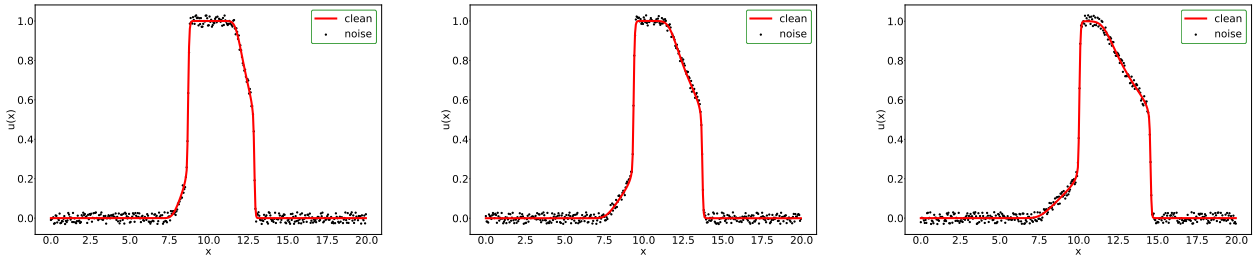



**Figure 20.** Clean and noisy data generated by the first initial data in Table 15 with $N_x = 400$ at three time points: 0.3 (left), 0.6 (middle) and 0.9 (right).

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 $\beta$ 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 $\beta$. (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