PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network

Zichao Long [a], Yiping Lu [a], Bin Dong [b,c,∗]

[a] *School of Mathematical Sciences, Peking University, Beijing, China*
[b] *Beijing International Center for Mathematical Research, Peking University, Beijing, China*
[c] *Center for Data Science, Peking University, Beijing, China*

ARTICLE INFO

ABSTRACT

Partial differential equations (PDEs) are commonly derived based on empirical observations. However, recent advances of technology enable us to collect and store massive amount of data, which offers new opportunities for data-driven discovery of PDEs. In this paper, we propose a new deep neural network, called PDE-Net 2.0, to discover (time-dependent) PDEs from observed dynamic data with minor prior knowledge on the underlying mechanism that drives the dynamics. The design of PDE-Net 2.0 is based on our earlier work [1] where the original version of PDE-Net was proposed. PDE-Net 2.0 is a combination of numerical approximation of differential operators by convolutions and a symbolic multi-layer neural network for model recovery. Comparing with existing approaches, PDE-Net 2.0 has the most flexibility and expressive power by learning both differential operators and the nonlinear response function of the underlying PDE model. Numerical experiments show that the PDE-Net 2.0 has the potential to uncover the hidden PDE of the observed dynamics, and predict the dynamical behavior for a relatively long time, even in a noisy environment.

© 2019 Elsevier Inc. All rights reserved.

1. Introduction

Differential equations, especially partial differential equations (PDEs), play a prominent role in many disciplines to describe the governing physical laws underlying a given system of interest. Traditionally, PDEs are derived mathematically or physically based on some basic principles, e.g. from Schrödinger's equations in quantum mechanics to molecular dynamic models, from Boltzmann equations to Navier-Stokes equations, etc. However, the mechanisms behind many complex systems in modern applications (such as many problems in multiphase flow, neuroscience, finance, biological science, etc.) are still generally unclear, and the governing equations of these systems are commonly obtained by empirical formulas [2,3]. With the recent rapid development of sensors, computational power, and data storage in the last decade, huge quantities of data can now be easily collected, stored and processed. Such vast quantity of data offers new opportunities for data-driven discovery of (potentially new) physical laws. Then, one may ask the following interesting and intriguing question: can we learn a PDE model to approximate the observed complex dynamic data?

---

* Corresponding author.
 E-mail address: dongbin@math.pku.edu.cn (B. Dong).

1.1. Existing work and motivations

Earlier attempts on data-driven discovery of hidden physical laws include [4,5]. Their main idea is to compare numerical differentiations of the experimental data with analytic derivatives of candidate functions, and apply the symbolic regression and the evolutionary algorithm to determining the nonlinear dynamic system. When the form of the nonlinear response function of a PDE is known, except for some scalar parameters, [6] presented a framework to learn these unknown parameters by introducing regularity between two consecutive time step using Gaussian process. Later in [7], a PDE constraint interpolation method was introduced to uncover the unknown parameters of the PDE model. An alternative approach is known as the sparse identification of nonlinear dynamics (SINDy) [8–13]. The key idea of SINDy is to first construct a dictionary of simple functions and partial derivatives that are likely to appear in the equations. Then, it takes the advantage of sparsity promoting techniques (e.g. ℓ_1 regularization) to select candidates that most accurately represent the data. In [14], the authors studied the problem of sea surface temperature prediction (SSTP). They assumed that the underlying physical model was an advection-diffusion equation. They designed a special neural network according to the general solution of the equation. Comparing with traditional numerical methods, their approach showed improvements in accuracy and computation efficiency.

Recent work greatly advanced the progress of PDE identification from observed data. However, SINDy requires to build a sufficiently large dictionary which may lead to high memory load and computation cost, especially when the number of model variables is large. Furthermore, the existing methods based on SINDy treat spatial and temporal information of the data separately and does not take full advantage of the temporal dependence of the PDE model. Although the framework presented by [6,7] is able to learn hidden physical laws using less data than the approach based on SINDy, the explicit form of the PDEs is assumed to be known except for a few scalar learnable parameters. The approach of [14] is specifically designed for advection-diffusion equations, and cannot be readily extended to other types of equations. Therefore, extracting governing equations from data in a less restrictive setting remains a great challenge.

The main objective of this paper is to design a *transparent* deep neural network to uncover hidden PDE models from observed complex dynamic data with minor prior knowledge on the mechanisms of the dynamics, and to perform accurate predictions at the same time. The reason we emphasize on both model recovery and prediction is because: 1) the ability to conduct accurate long-term prediction is an important indicator of accuracy of the learned PDE model (the more accurate is the prediction, the more confident we have on the underlying recovered PDE model); 2) the trained neural network can be readily used in applications and does not need to be re-trained when initial conditions are altered. Our inspiration comes from the latest development of deep learning techniques in computer vision. An interesting fact is that some popular networks in computer vision, such as ResNet[15,16], have close relationship with ODEs/PDEs and can be naturally merged with traditional computational mathematics in various tasks [17–26]. However, existing deep networks designed in deep learning mostly emphasis on expressive power and prediction accuracy. These networks are not transparent enough to be able to reveal the underlying PDE models, although they may perfectly fit the observed data and perform accurate predictions. Therefore, we need to carefully design the network by combining knowledge from deep learning and numerical PDEs.

1.2. Our approach

The proposed deep neural network is an upgraded version of our original PDE-Net [1]. The main difference is the use of a symbolic network to approximate the nonlinear response function, which significantly relaxes the requirement on the prior knowledge on the PDEs to be recovered. During training, we no longer need to assume the general type of the PDE (e.g. convection, diffusion, etc.) is known. Furthermore, due to the lack of prior knowledge on the general type of the unknown PDE models, more carefully designed constraints on the convolution filters as well as the parameters of the symbolic network are introduced. We refer to this upgraded network as PDE-Net 2.0.

Assume that the PDE to be recovered takes the following generic form

$$U_t = F(U, \nabla U, \nabla^2 U, \ldots), \quad x \in \Omega \subset \mathbb{R}^2, \quad t \in [0, T].$$

PDE-Net 2.0 is designed as a feed-forward network by discretizing the above PDE using forward Euler in time and finite difference in space. The forward Euler approximation of temporal derivative makes PDE-Net 2.0 ResNet-like [15,18,21], and the finite difference is realized by convolutions with trainable kernels (or filters). The nonlinear response function F is approximated by a symbolic neural network, which shall be referred to as $SymNet$. All the parameters of the $SymNet$ and the convolution kernels are jointly learned from data. To grant full transparency to the PDE-Net 2.0, proper constraints are enforced on the $SymNet$ and the filters. Full details on the architecture and constraints will be presented in Section 2.

1.3. Relation with model reduction

Data-driven discovery of hidden physical laws and model reduction have a lot in common. Both of them concern on representing observed data using relatively simple models. The main difference is that, model reduction emphasis more on numerical precision rather than acquiring the analytic form of the model.

It is common practice in model reduction to use a function approximator to express the unknown terms in the reduced models, such as approximating subgrid stress for large-eddy simulation [27–29] or approximating interatomic forces for coarse-grained molecular dynamic systems [30,31]. Our work may serve as an alternative approach to model reduction and help with analyzing the reduced models.

1.4. Novelty

The particular novelties of our approach are that we impose appropriate constraints on the learnable filters and use a properly designed symbolic neural network to approximate the response function F. Using learnable filters makes the PDE-Net 2.0 more flexible, and enables more powerful approximation of unknown dynamics and longer time prediction (see numerical experiments in Section 3 and Section 4). Furthermore, the constraints on the learnable filters and the use of a deep symbolic neural network enable us to uncover the analytic form of F with minor prior knowledge on the dynamic, which is the main advantage of PDE-Net 2.0 over the original PDE-Net. In addition, the composite representation by the symbolic network is more efficient and flexible than SINDy. Therefore, the proposed PDE-Net 2.0 is distinct from the existing learning based methods to discover PDEs from data.

2. PDE-Net 2.0: architecture, constraints and training

Given a series of measurements of some physical quantities $\{U(t, x, y) : t = t_0, t_1, \cdots, (x, y) \in \Omega \subset \mathbb{R}^2\} \subset \mathbb{R}^d$ with d being the number of physical quantities of interest, we want to discover the governing PDEs from the observed data $\{U(t, x, y)\}$. We assume that the observed data are associated with a PDE that takes the following general form:

$$U_t(t, x, y) = F(U, U_x, U_y, U_{xx}, U_{xy}, U_{yy}, \ldots),\tag{1}$$

here $U(t, \cdot) : \Omega \mapsto \mathbb{R}^d$, $F(U, U_x, U_y, U_{xx}, U_{xy}, U_{yy}, \ldots) \in \mathbb{R}^d$, $(x, y) \in \Omega \subset \mathbb{R}^2$, $t \in [0, T]$. Our objective is to design a feed-forward network, called PDE-Net 2.0, to approximate the unknown PDE (1) from its solution samples in the way that: 1) we are able to reveal the analytic form of the response function F and the differential operators involved; 2) we can conduct long-term prediction on the dynamical behavior of the equation for any given initial conditions. There are two main components of the PDE-Net 2.0 that are combined together in the same network: one is automatic determination on the differential operators involved in the PDE and their discrete approximations; the other is to approximate the nonlinear response function F. In this section, we start with discussions on the overall framework of the PDE-Net 2.0 and then introduce the details on these two components. Regularization and training strategies will be given near the end of this section.

2.1. Architecture of PDE-Net 2.0

Inspired by the dynamic system perspective of deep neural networks [17–22], we consider forward Euler as the temporal discretization of the evolution PDE (1), and unroll the discrete dynamics to a feed-forward network. One may consider more sophisticated temporal discretization which naturally leads to different network architectures [21]. For simplicity, we focus on forward Euler in this paper.

2.1.1. δt-block:
Let $\tilde{U}(t + \delta t, \cdot)$ be the predicted value at time $t + \delta t$ based on $\tilde{U}(t, \cdot)$. Then, we design an approximation framework as follows

$$\tilde{U}(t + \delta t, \cdot) \approx \tilde{U}(t, \cdot) + \delta t \cdot SymNet_m^k(D_{00}\tilde{U}, D_{01}\tilde{U}, D_{10}\tilde{U}, \cdots).\tag{2}$$

Here, the operators D_{ij} are convolution operators with the underlying filters denoted by q_{ij}, i.e. $D_{ij}u = q_{ij} \circledast u$. The operators D_{10}, D_{01}, D_{11}, etc. approximate differential operators, i.e. $D_{ij}u \approx \frac{\partial^{i+j}u}{\partial^i x \partial^j y}$. In particular, the operators D_{00} is a certain averaging operator. The purpose of introducing the average operators in stead of simply using the identity is to improve the expressive power of the network and enables it to capture more complex dynamics.

Other than the assumption that the observed dynamics is governed by a PDE of the form (1), we assume that the highest order of the PDE is less than some positive integer. Then, we can assume that F is a function of m variables with known m. The task of approximating F in (1) is equivalent to a multivariate regression problem. In order to be able to identify the analytic form of F, we use a symbolic neural network denote by $SymNet_m^k$ to approximate F, where k denotes the depth of the network. Note that, if F is a vector function, we use multiple $SymNet_m^k$ to approximate the components of F separately.

Combining the aforementioned approximation of differential operators and the nonlinear response function, we obtain an approximation framework (2) which will be referred to as a δt-block (see Fig. 1). Details of these two components can be found later in Section 2.2 and Section 2.3.
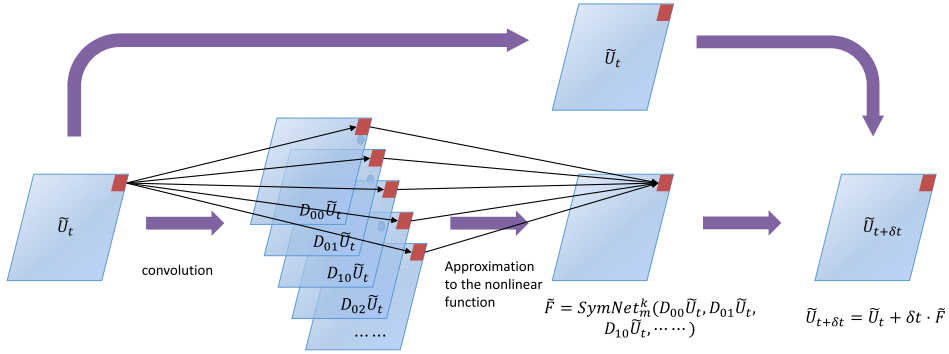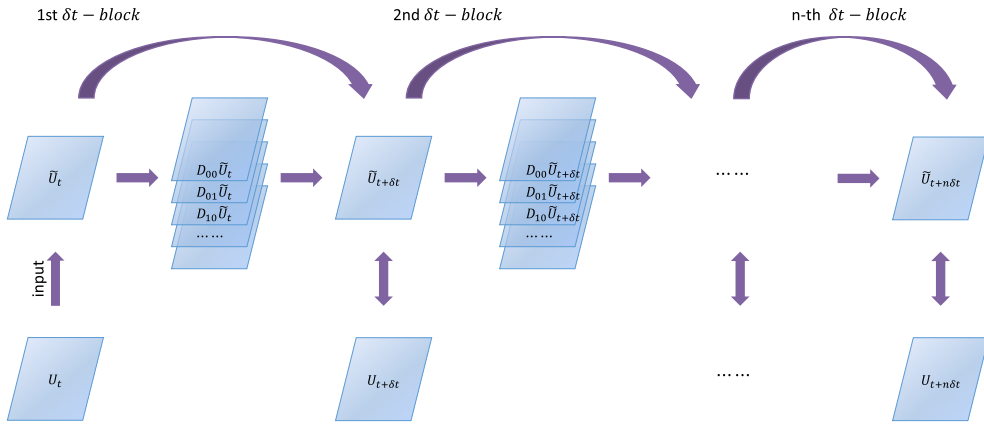
Fig. 1. The schematic diagram of a δt-block.

Fig. 2. The schematic diagram of the PDE-Net 2.0.

2.1.2. Multiple δt-blocks:

One δt-block only guarantees the accuracy of one-step dynamics, which does not take error accumulation into consideration. In order to facilitate a long-term prediction, we stack multiple δt-blocks into a deep network, and call this network the *PDE-Net 2.0* (see Fig. 2). The importance of stacking multiple δt-blocks will be demonstrated by our numerical experiments in Section 3 and 4.

The PDE-Net 2.0 can be easily described as: (1) stacking one δt-block multiple times; (2) sharing parameters in all δt-blocks. Given a set of observed data $\{U(t, \cdot)\}$, training a PDE-Net 2.0 with n δt-blocks needs to minimize the accumulated error $||U(t + n\delta t, \cdot) - \tilde{U}(t + n\delta t, \cdot)||_2^2$, where $\tilde{U}(t + n\delta t, \cdot)$ is the output from the PDE-Net 2.0 (i.e. n δt-blocks) with input $U(t, \cdot)$, and $U(t + n\delta t, \cdot)$ is observed training data.

2.2. Convolutions and differentiations

In the original PDE-Net [1], the learnable filters are properly constrained so that we can easily identify their correspondence to differential operators. PDE-Net 2.0 adopts the same constrains as the original version of the PDE-Net. For completeness, we shall review the related notions and concepts, and provide more details.

A profound relationship between convolutions and differentiations was presented by [32,33], where the authors discussed the connection between the order of sum rules of filters and the orders of differential operators. Note that the definition of convolution we use follows the convention of deep learning, which is defined as

$$(f \circledast q)[l_1, l_2] := \sum_{k_1, k_2} q[k_1, k_2] f[l_1 + k_1, l_2 + k_2].$$

This is essentially correlation instead of convolution in the mathematics convention. Note that if f is of finite size, we use periodic boundary condition.

The order of sum rules is closely related to the order of vanishing moments in wavelet theory [34,35]. We first recall the definition of the order of sum rules.

Definition 2.1 *(Order of sum rules).* For a filter q, we say q to have sum rules of order $\alpha = (\alpha_1, \alpha_2)$, where $\alpha \in \mathbb{Z}_+^2$, provided that

$$\sum_{k \in \mathbb{Z}^2} k^\beta q[k] = 0 \tag{3}$$

for all $\beta = (\beta_1, \beta_2) \in \mathbb{Z}_+^2$ with $|\beta| := \beta_1 + \beta_2 < |\alpha|$ and for all $\beta \in \mathbb{Z}_+^2$ with $|\beta| = |\alpha|$ but $\beta \neq \alpha$. If (3) holds for all $\beta \in \mathbb{Z}_+^2$ with $|\beta| < K$ except for $\beta \neq \bar{\beta}$ with certain $\bar{\beta} \in \mathbb{Z}_+^2$ and $|\bar{\beta}| = J < K$, then we say q to have total sum rules of order $K \backslash \{J + 1\}$.

In practical implementation, the filters are normally finite and can be understood as matrices. For an $N \times N$ filter q (N being an odd number), assuming the indices of q start from $-\frac{N-1}{2}$, (3) can be written in the following simpler form

$$\sum_{l=-\frac{N-1}{2}}^{\frac{N-1}{2}} \sum_{m=-\frac{N-1}{2}}^{\frac{N-1}{2}} l^{\beta_1} m^{\beta_2} q[l, m] = 0.$$

The following proposition from [33] links the orders of sum rules with orders of differential operators.

Proposition 2.1. *Let q be a filter with sum rules of order $\alpha \in \mathbb{Z}_+^2$. Then for a smooth function $F(x)$ on $\mathbb{R}^2$, we have*

$$\frac{1}{\varepsilon^{|\alpha|}} \sum_{k \in \mathbb{Z}^2} q[k] F(x + \varepsilon k) = C_\alpha \frac{\partial^\alpha}{\partial x^\alpha} F(x) + O(\varepsilon), \text{ as } \varepsilon \to 0. \tag{4}$$

If, in addition, q has total sum rules of order $K \backslash \{|\alpha| + 1\}$ for some $K > |\alpha|$, then

$$\frac{1}{\varepsilon^{|\alpha|}} \sum_{k \in \mathbb{Z}^2} q[k] F(x + \varepsilon k) = C_\alpha \frac{\partial^\alpha}{\partial x^\alpha} F(x) + O(\varepsilon^{K-|\alpha|}), \text{ as } \varepsilon \to 0. \tag{5}$$

According to Proposition 2.1, an αth order differential operator can be approximated by the convolution of a filter with α order of sum rules. Furthermore, according to (5), one can obtain a high order approximation of a given differential operator if the corresponding filter has an order of total sum rules with $K > |\alpha| + k, k \geqslant 1$. For example, consider filter

$$q = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}.$$

It has a sum rules of order $(1, 0)$, and a total sum rules of order $3 \backslash \{2\}$. Thus, up to a constant and a proper scaling, q corresponds to a discretization of $\frac{\partial}{\partial x}$ with second order accuracy.

For an $N \times N$ filter q, define the *moment matrix* of q as

$$M(q) = (m_{i,j})_{N \times N}, \tag{6}$$

where

$$m_{i,j} = \frac{1}{i!j!} \sum_{k_1,k_2=-\frac{N-1}{2}}^{\frac{N-1}{2}} k_1^i k_2^j q[k_1, k_2], i, j = 0, 1, \ldots, N - 1. \tag{7}$$

We shall call the (i, j)-element of $M(q)$ the (i, j)-moment of q for simplicity. For any smooth function $f : \mathbb{R}^2 \to \mathbb{R}$, we apply convolution on the sampled version of f with respect to the filter q. By Taylor's expansion, one can easily obtain the following formula

$$\sum_{k_1,k_2=-\frac{N-1}{2}}^{\frac{N-1}{2}} q[k_1, k_2] f(x + k_1 \delta x, y + k_2 \delta y)$$

$$= \sum_{k_1,k_2=-\frac{N-1}{2}}^{\frac{N-1}{2}} q[k_1, k_2] \sum_{i,j=0}^{N-1} \frac{\partial^{i+j} f}{\partial^i x \partial^j y} \bigg|_{(x,y)} \frac{k_1^i k_2^j}{i!j!} \delta x^i \delta y^j + o(|\delta x|^{N-1} + |\delta y|^{N-1})$$

$$= \sum_{i,j=0}^{N-1} m_{i,j} \delta x^i \delta y^j \cdot \frac{\partial^{i+j} f}{\partial^i x \partial^j y} \bigg|_{(x,y)} + o(|\delta x|^{N-1} + |\delta y|^{N-1}). \tag{8}$$
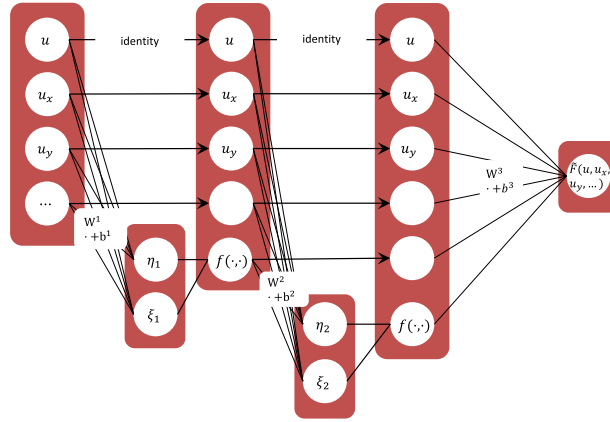
Fig. 3. The schematic diagram of $SymNet$.

From (8) we can see that filter q can be designed to approximate any differential operator with prescribed order of accuracy by imposing constraints on $M(q)$.

For example, if we want to approximate $\frac{\partial u}{\partial x}$ (up to a constant) by convolution $q \circledast u$ where q is a 3×3 filter, we can consider the following constrains on $M(q)$:

$$\begin{pmatrix} 0 & 0 & \star \\ 1 & \star & \star \\ \star & \star & \star \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & \star \\ 0 & \star & \star \end{pmatrix}. \tag{9}$$

Here, $\star$ means no constraint on the corresponding entry. The constraints described by the moment matrix on the left of (9) guarantee the approximation accuracy is at least first order, and the one on the right guarantees an approximation of at least second order. In particular, when all entries of $M(q)$ are constrained, e.g.

$$M(q) = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$

the corresponding filter can be uniquely determined. In the PDE-Net 2.0, all filters are learned subjected to partial constraints on their associated moment matrices, with at least second order accuracy.

2.3. Design of $SymNet$: a symbolic neural network

The symbolic neural network $SymNet_m^k$ of the PDE-Net 2.0 is introduced to approximate the multivariate nonlinear response function F of (1). Neural networks have recently been proven effective in approximating multivariate functions in various scenarios [36–40]. For the problem we have in hand, we not only require the network to have good expressive power, but also good transparency so that the analytic form of F can be readily inferred after training. Our design of $SymNet_m^k$ is motivated by EQL/EQL$^{\div}$ proposed by [41,42].

The $SymNet_m^k$, as illustrated in Fig. 3, is a network that takes an m dimensional vector as input and has k hidden layers. Fig. 3 shows the symbolic neural network with two hidden layers, i.e. $SymNet_m^2$, where f is a dyadic operation unit, e.g. multiplication or division. In this paper, we focus on multiplication, i.e. we take $f(a, b) = a \times b$. Different from EQL/EQL$^{\div}$, each hidden layer of the $SymNet_m^k$ directly takes the outputs from the preceding layer as inputs, rather than a linear combination of them. Furthermore, it adds one additional variable (i.e. $f(\cdot, \cdot)$) at each hidden layer. To better understand $SymNet_m^k$, we present an example in Algorithm 1 showing how $SymNet_6^2$ is constructed. In particular, when $b^i = 0, i = 1, 2, 3$,

$$W^1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}, W^2 = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

and $W^3 = (0, 0, 0, 0, 0, 0, -1, -1)$, then $SymNet_6^2(u, u_x, u_y, v, v_x, v_y) = -uu_x - vu_y$ which is the right-hand-side of $\frac{\partial u}{\partial t}$ of the Burgers' equation without viscosity.

The $SymNet_m^k$ can represent all polynomials of variables $(x_1, x_2, \ldots, x_m)$ with the total number of multiplications not exceeding k. If needed, one can add more operations to the $SymNet_m^k$ to increase the capacity of the network.

Algorithm 1 $SymNet_6^2$.

Input: $u, u_x, u_y, v, v_x, v_y \in \mathbb{R}$,

$(\eta_1, \xi_1)^\top = W^1 \cdot (u, u_x, u_y, v, v_x, v_y)^\top + b^1, W^1 \in \mathbb{R}^{2 \times 6}, b^1 \in \mathbb{R}^2$;

$f_1 = \eta_1 \cdot \xi_1$;

$(\eta_2, \xi_2)^\top = W^2 \cdot (u, u_x, u_y, v, v_x, v_y, f_1)^\top + b^2, W^2 \in \mathbb{R}^{2 \times 7}, b^2 \in \mathbb{R}^2$;

$f_2 = \eta_2 \cdot \xi_2$;

Output: $F = W^3 \cdot (u, u_x, u_y, v, v_x, v_y, f_1, f_2)^\top + b^3 \in \mathbb{R}, W^3 \in \mathbb{R}^{1 \times 8}, b^3 \in \mathbb{R}$.

Now, we show that $SymNet_m^k$ is more compact than the dictionaries of SINDy. For that, we first introduce some notions.

Definition 2.2. Define the set of all polynomials of m variables $(x_1, \cdots, x_m)$ with the total number of multiplications not exceeding k as $\mathcal{P}^k[x_1, \cdots, x_m]$. Here, the total number of multiplications of $\mathcal{P}^k[x_1, \cdots, x_m]$ is counted as follows:

- For any monomial of degree k, if $k \geq 2$, then the number of multiplications of the monomial is counted as $k - 1$. When $k = 1$ or 0, the count is 0.
- For any polynomial P, the total number of multiplications is counted as the sum of the number of multiplications of its monomials.

For example, $\sum_{i=1}^{m} x_i + \sum_{i=1}^{k} x_i x_{i+1}$ and $\prod_{i=1}^{k+1} x_i$ with $k < m$ are all members of $\mathcal{P}^k[x_1, \cdots, x_m]$. The elements in $\mathcal{P}^k[x_1, \cdots, x_m]$ are of simple forms when k is relatively small. The following proposition shows that $SymNet_m^k$ can represent all polynomials of variables $(x_1, x_2, \ldots, x_m)$ with the total number of multiplications not exceeding k. Note that the actual capacity of $SymNet_m^k$ is larger than $\mathcal{P}^k[x_1, \cdots, x_m]$, i.e. $\mathcal{P}^k[x_1, \cdots, x_m]$ is a subset of the set of functions that $SymNet_m^k$ can represent.

Proposition 2.2. For any $P \in \mathcal{P}^k[x_1, \cdots, x_m]$, there exists a set of parameters for $SymNet_m^k$ such that

$$P = SymNet_m^k(x_1, \cdots, x_m).$$

Proof. We prove this proposition by induction. When $k = 1$, the conclusion obviously holds. Suppose the conclusion holds for k. For any polynomial $P \in \mathcal{P}^{k+1}[x_1, \cdots, x_m]$, we only need to consider the cases when P has a total number of multiplications greater than 1.

We take any monomial of P that has degree greater than 1, which we suppose take the form $x_1 x_2 \cdot A$ where A is a monomial of variable $(x_1, \ldots, x_m)$. Then, P can be written as $P = x_1 x_2 A + Q$. Define new variable $x_{m+1} = x_1 x_2$. Then, we have

$$P = x_{m+1} A + Q \in \mathcal{P}^k[x_1, \cdots, x_m, x_{m+1}].$$

By the induction hypothesis, there exists a set of parameters such that $P = SymNet_{m+1}^k(x_1, \cdots, x_{m+1})$.

We take the linear transform between the input layer and the first hidden layer of $SymNet_m^{k+1}$ as

$$W^1 \cdot (x_1, \cdots, x_m)^\top + b_1 = (x_1, x_2)^\top.$$

Then, the output of the first hidden layer is $x_1, x_2, \cdots, x_m, x_1 x_2$. If we use it as the input of $SymNet_{m+1}^k$, we have

$$\begin{aligned} P(x_1, \cdots, x_m) &= SymNet_{m+1}^k(x_1, \cdots, x_m, x_1 x_2) \\ &= SymNet_m^{k+1}(x_1, \cdots, x_m), \end{aligned}$$

which concludes the proof. $\square$

SINDy constructs a dictionary that includes all possible monomials up to a certain degree. Observe that there are totally $\binom{m+l}{l}$ monomials with m variables and a degree not exceeding $l (\in \mathbb{N})$. Our symbolic network, however, is more compact than SINDy. The following proposition compares the complexity of $SymNet_m^k$ and SINDy, whose proof is straightforward.

Proposition 2.3. Let $P \in \mathcal{P}^k[x_1, \cdots, x_m]$ and suppose P have monomials of degree $\leq l$.

- The memory load of $SymNet_m^k$ that approximates P is $O(m + k)$. The number of flops for evaluating $SymNet_m^k$ is $O(k(m + k))$.
- Constructing a dictionary with all possible polynomials of degree l requires a memory load of $\binom{m+l}{l}$, and evaluation of a linear combination of dictionary members requires $O(\binom{m+l}{l})$ flops.

We use the following example to show the advantage of $SymNet$ over SINDy. Consider two variables u, v and all of their derivatives of order ≤ 2:

$$(u, v, u_x, u_y, \cdots, v_{yy}) \in \mathbb{R}^{12}.$$

Suppose the polynomial to be approximated is $P = -uu_x - vu_y + u_{xx}$. For $k = l = 3$, the size of the dictionary of SINDy is $\binom{15}{3} = 455$ and the computation of linear combination of the elements requires 909 flops. The memory load of $SymNet_{12}^3$, however, is 15 and an evaluation of the network requires 180 flops. Therefore, $SymNet$ can significantly reduce memory load and computation cost when input data is large. Note that when k is large and l small, $SymNet_m^k$ is worse than SINDy. However, for system identification problems, we normally wish to obtain a compact representation (i.e. smaller k). Thus, $SymNet$ takes full advantage of this prior knowledge and can significantly save on memory and computation cost which is crucial in the training of the PDE-Net 2.0.

2.4. Loss function and regularization

We adopt the following loss function to train the proposed PDE-Net 2.0:

$$L = L^{data} + \lambda_1 L^{moment} + \lambda_2 L^{SymNet},$$

where the hyper-parameters λ_1 and λ_2 are chosen as $\lambda_1 = 0.001$ and $\lambda_2 = 0.005$. Now, we present details on each of the term of the loss function and introduce pseudo-upwind as an additional constraint on PDE-Net 2.0.

2.4.1. Data approximation L^{data}

Consider the data set $\{U_j(t_i, \cdot) : 1 \leq i \leq n, 1 \leq j \leq N\}$, where n is the number of δt-blocks and N is the total number of samples. The index j indicates the j-th solution path with a certain initial condition of the unknown dynamics. Note that one can split a long solution path into multiple shorter ones. We would like to train the PDE-Net 2.0 with n δt-blocks. For a given $n \geq 1$, every pair of the data $\{U_j(t_0, \cdot), U_j(t_i, \cdot)\}$, for each j and $i \leq n$, is a training sample, where $U_j(t_0, \cdot)$ is the input and $U_j(t_i, \cdot)$ is the label that we need to match with the output from the network. For that, we define the data approximation term L^{data} as:

$$L^{data} = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{N} l_{ij}/\delta t^2, \text{ where } l_{ij} = ||U_j(t_i, \cdot) - \tilde{U}_j(t_i, \cdot)||_2^2,$$

where $\tilde{U}_j(t_i, \cdot)$ is the output of the PDE-Net 2.0 with $\tilde{U}_j(t_0, \cdot) = U_j(t_0, \cdot)$ as the input.

2.4.2. Regularization: L^{moment} and L^{SymNet}

For a given threshold s, defined Huber's loss function ℓ_1^s as

$$\ell_1^s(x) = \begin{cases} |x| - \frac{s}{2} & \text{if } |x| > s \\ \frac{1}{2s}x^2 & \text{else.} \end{cases}$$

Then, we define L^{moment} as

$$L^{moment} = \sum_{i,j} \sum_{i_1,j_1} \ell_1^s(M(q_{ij})[i_1, j_1]),$$

where q_{ij} are the filters of PDE-Net 2.0 and $M(q)$ is the moment matrix of q. We use this loss function to regularize the learnable filters to reduce overfitting. In our numerical experiments, we will use $s = 0.01$.

Given ℓ_1^s as defined above, we use it to enforce sparsity on the parameters of $SymNet$. This will help to reduce overfitting and enable more stable prediction. The loss function L^{SymNet} is defined as

$$L^{SymNet} = \sum_{p \in \text{parameters of } SymNet} \ell_1^s(p).$$

We set $s = 0.001$ in our numerical experiments.

2.4.3. Pseudo-upwind

In numerical PDEs, to ensure stability of a numerical scheme, we need to design conservation schemes or use upwind schemes [43–45]. This is also important for PDE-Net 2.0 during inferencing. However, the challenge we face is that we do not know apriori the form or the type of the PDE. Therefore, we introduce a method called *pseudo-upwind* to help with maintaining stability of the PDE-Net 2.0.

Given a 2D filter q, define the flipping operators $\text{flip}_x(q)$ and $\text{flip}_y(q)$ as

$$\left(\text{flip}_x(q)\right)[k_1, k_2] = -q[-k_1, k_2], \quad k_1, k_2 = -(N-1)/2, \cdots, (N-1)/2$$

and

$$\left(\text{flip}_y(q)\right)[k_1, k_2] = -q[k_1, -k_2], \quad k_1, k_2 = -(N-1)/2, \cdots, (N-1)/2.$$

In each δt-block of the PDE-Net 2.0, before we apply convolution with a filter, we first use $SymNet$ to determine whether we should use the filter or flip it first. We use the following univariate PDE as an example to demonstrate our idea. Given PDE $u_t = F(u, \cdots)$, suppose the input of a δt-block is u. The algorithm of pseudo-upwind is described by the following Algorithm 2.

Algorithm 2 Pseudo-upwind in δt-block.

Input: u

$u_{01} = q_{01} \circledast u, u_{10} = q_{10} \circledast u$
$\tilde{F}_0 = SymNet_6^k(u, u_{01}, u_{10}, q_{02} \circledast u, q_{11} \circledast u, q_{20} \circledast u)$
if $\frac{\partial \tilde{F}_0}{\partial u_{01}} > 0$ **then**
$\quad u_x = u_{01}$
else
$\quad u_x = \text{flip}_x(q_{01}) \circledast u$
end if
if $\frac{\partial \tilde{F}_0}{\partial u_{10}} > 0$ **then**
$\quad u_y = u_{10}$
else
$\quad u_y = \text{flip}_y(q_{10}) \circledast u$
end if
$\tilde{F} = SymNet_6^k(u, u_x, u_y, q_{02} \circledast u, q_{11} \circledast u, q_{20} \circledast u)$
Return: $u + \delta t \tilde{F}$

Remark 2.1. Note that the algorithm does not exactly enforce upwind in general. This is why we call it pseudo-upwind. We further note that:

- Given a PDE of the form $u_t = G(u)u_x + H(u)u_y + \lambda(u_{xx} + u_{yy})$, we can use $G(u)$ and $H(u)$ to determine whether we should flip a filter or not.
- For a vector PDE, such as $U = (u, v)^\top$, we can use, for instance, $\frac{\partial \tilde{F}_0}{\partial u_{01}}, \frac{\partial \tilde{F}_0}{\partial v_{10}}$ to determine how we should approximate u_x and u_y in the δt-block [44].

2.5. Initialization and training

In the PDE-Net 2.0, parameters can be divided into three groups: 1) moment matrices to generate convolution kernels; 2) the parameters of $SymNet$; and 3) hyper-parameters, such as the number of filters, the size of filters, the number of δt-Blocks and number of hidden layers of $SymNet$, regularization weights $\lambda_1, \lambda_2, \lambda_3$, etc. The parameters of the $SymNet$ are shared across the computation domain Ω, and are initialized by random sampling from a Gaussian distribution. For the filters, we initialize D_{01}, D_{10} with second order pseudo-upwind scheme and central difference for all other filters. For example, if the size of the filters were set to be 5×5, then the initial values of the convolution kernels $q_{01}, q_{02} \in \mathbb{R}^{5 \times 5}$ are

$$q_{01} = \begin{pmatrix} & & \mathbf{0} & & \\ 0 & 0 & -3 & 4 & -1 \\ & & \mathbf{0} & & \end{pmatrix}, \quad q_{02} = \begin{pmatrix} & & \mathbf{0} & & \\ 0 & 1 & -2 & 1 & 0 \\ & & \mathbf{0} & & \end{pmatrix}.$$

We use layer-wise training to train the PDE-Net 2.0. We start with training the PDE-Net 2.0 on the first δt-block with a batch of data, and then use the results of the first δt-block as the initialization and restart training on the first two δt-blocks with another batch. Repeat this procedure until we complete all n δt-blocks. Note that all the parameters in each of the δt-block are shared across layers. In addition, we add a warm-up step before the training of the first δt-block by fixing filters and setting regularization term to be 0 (i.e. $\lambda_1 = \lambda_2 = 0$). The warm-up step is to obtain a good initial guess of the parameters of $SymNet$.

To demonstrate the necessity of having learnable filters, we will compare the PDE-Net 2.0 containing learnable filters with the PDE-Net 2.0 having fixed filters. To differentiate the two cases, we shall call the PDE-Net 2.0 with fixed filters the

Table 1
PDE model identification.

Correct PDE	$u_t = -uu_x - vu_y + 0.05(u_{xx} + u_{yy})$
	$v_t = -uv_x - vv_y + 0.05(v_{xx} + v_{yy})$
Frozen-PDE-Net 2.0	$u_t = -0.906uu_x - 0.901vu_y + 0.033u_{xx} + 0.037u_{yy}$
	$v_t = -0.907vv_y - 0.902uv_x + 0.039v_{xx} + 0.032v_{yy}$
PDE-Net 2.0	$u_t = -0.979uu_x - 0.973u_yv + 0.052u_{xx} + 0.051u_{yy}$
	$v_t = -0.973uv_x - 0.977vv_y + 0.053v_{xx} + 0.051v_{yy}$

"Frozen-PDE-Net 2.0". Note that for Frozen-PDE-Net 2.0, the filters are fixed to be the initial values we choose to train the regular PDE-Net 2.0. This is a natural choice since when we know apriori that the PDE is Burgers' equation, it would be a stable finite difference scheme. However, intuitively speaking, freezing any finite difference approximations of the differential operators during training of PDE-Net 2.0 is not ideal, because you cannot possibly know which numerical scheme is best to use without knowing the form of the PDE. Therefore, for inverse problem, it is better to learn both the PDE model and the discretization of the PDE model simultaneously. This assertion is supported by our empirical comparisons between frozen and regular PDE-Net 2.0 in Tables 1 and 3.

3. Numerical studies: Burgers' equation

Burgers' equation is a fundamental partial differential equation in many areas such as fluid mechanics and traffic flow modeling. It has a lot in common with the Navier-Stokes equation, e.g. the same type of advective nonlinearity and the presence of viscosity.

3.1. Simulated data, training and testing

In this section we consider a 2-dimensional Burger's equation with periodic boundary condition on $\Omega = [0, 2\pi] \times [0, 2\pi]$,

$$\begin{cases} \frac{\partial U}{\partial t} & = -U \cdot \nabla U + \nu \Delta U, U = (u, v)^\top \\ U|_{t=0} & = U_0(x, y), \end{cases} \tag{10}$$

with $(t, x, y) \in [0, 4] \times \Omega$, where $\nu = 0.05$.

The training data is generated by a finite difference scheme on a 128×128 mesh and then restricted to a 32×32 mesh. The temporal discretization is 2nd order Runge-Kutta with time step $\delta t = \frac{1}{1600}$, the spatial discretization uses a 2nd order upwind scheme for ∇ and the central difference scheme for Δ. The initial value $u_0(x, y), v_0(x, y)$ takes the following form,

$$w(x, y) = \frac{2w_0(x, y)}{\max_{x,y}|w_0|} + c \tag{11}$$

where $w_0(x, y) = \sum_{|k|,|l| \leq 4} \lambda_{k,l} \cos(kx + ly) + \gamma_{k,l} \sin(kx + ly)$, $\lambda_{k,l}, \gamma_{k,l} \sim \mathcal{N}(0, 1)$, $c \sim \mathcal{U}(-2, 2)$. Here, $\mathcal{N}(0, 1), \mathcal{U}(-2, 2)$ represents the standard normal distribution and uniform distribution on $[-2, 2]$ respectively. We also add noise to the generated data:

$$\widehat{U}(t, x, y) = U(t, x, y) + 0.001 \times MW \tag{12}$$

where $M = \max_{x,y,t}\{U(t, x, y)\}$, $W \sim \mathcal{N}(0, 1)$.

Suppose we know a priori that the order of the underlying PDE is no more than 2, we can use two $SymNet_{12}^5$ to approximate the right-hand side nonlinear response function of (10) component-wise. Let $U = (u, v)^\top$. We denote the two $SymNet_{12}^5$ as Net_u and Net_v respectively. Then, each δt-block of the PDE-Net 2.0 can be written as

$$\tilde{u}(t_{i+1}, \cdot) = \tilde{u}(t_i, \cdot) + \delta t \cdot Net_u(D_{00}\tilde{u}, D_{01}\tilde{u}, \cdots, D_{20}\tilde{u}, D_{00}\tilde{v}, \cdots, D_{20}\tilde{v})$$

$$\tilde{v}(t_{i+1}, \cdot) = \tilde{v}(t_i, \cdot) + \delta t \cdot Net_v(D_{00}\tilde{u}, D_{01}\tilde{u}, \cdots, D_{20}\tilde{u}, D_{00}\tilde{v}, \cdots, D_{20}\tilde{v})$$

where $\{D_{ij} : 0 \leq i + j \leq 2\}$ are convolution operators.

During training and testing, the data is generated on-the-fly. The size of the filters that will be used is 5×5. The total number of parameters in Net_u and Net_v is 336, and the number of trainable parameters in moment matrices is 105 for 5×5 filters ($6 \times 5 \times 5 - 45$ (constraint on moment)). During training, we use BFGS, instead of SGD, to optimize the parameters.

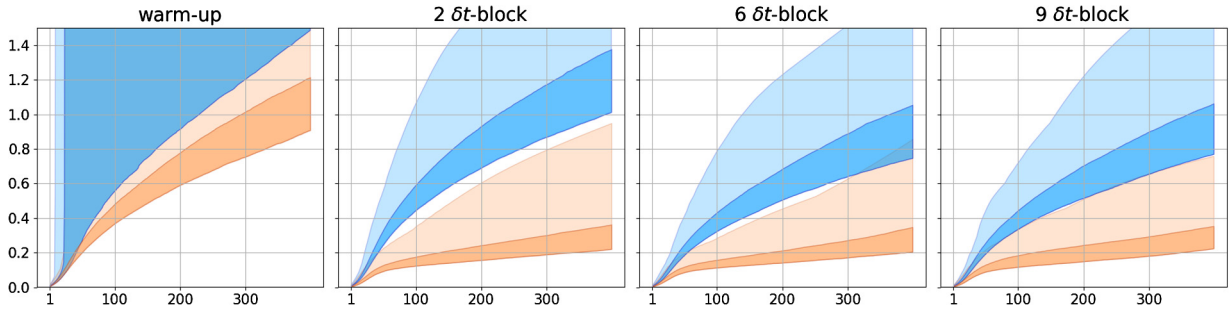**Fig. 4.** Burgers' equation: Prediction errors of the PDE-Net 2.0(orange) and Frozen-PDE-Net 2.0(blue) with 5×5 filters. In each plot, the horizontal axis indicates the time of prediction in the interval $(0, 400 \times \delta t] = (0, 4]$, and the vertical axis shows the relatively errors. The banded curves indicate the 25%-100% percentile of the relative errors among 1000 test samples. The darker regions indicate the 25%-75% percentile of the relative errors, which shows that PDE-Net 2.0 can predict very well in most cases. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)
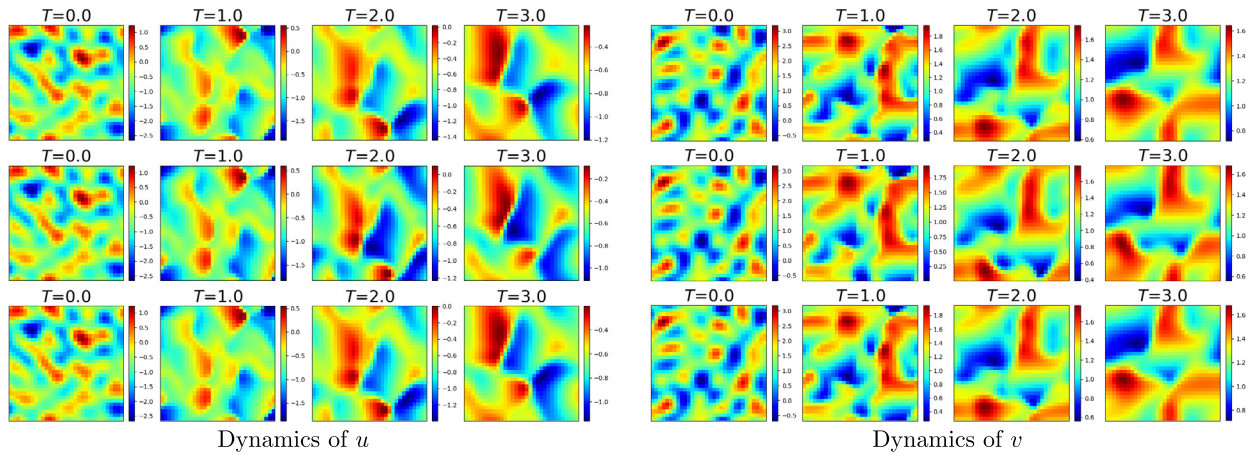
Fig. 5. Burgers' equation: The first row shows the images of the true dynamics. The last two rows show the images of the predicted dynamics using the Frozen-PDE-Net 2.0 (the second row) and PDE-Net 2.0 with 9 δt-blocks (the third row). Time step $\delta t = 0.01$.

We use 28 data samples per batch to train each δt-block and we only construct the PDE-Net 2.0 up to 9 layers, which requires totally 420 data samples during the whole training procedure.

3.2. Results and discussions

We first demonstrate the ability of the trained PDE-Net 2.0 to recover the analytic form of the unknown PDE model. We use the symbolic math tool in python to obtain the analytic form of $SymNet$. Results are summarized in Table 1. As one can see from Table 1 that we can recover the terms of the Burgers' equation with good accuracy, and using learnable filters helps with the identification of the PDE model. Furthermore, the terms that are not included in the Burgers' equation all have relatively small weights in the $SymNet$ (see Fig. 7).

We also demonstrate the ability of the trained PDE-Net 2.0 in prediction, i.e. the ability to generalize. After the PDE-Net 2.0 with n δt-blocks ($1 \leq n \leq 9$) is trained, we randomly generate 1000 initial guesses based on (11) and (12), feed them to the PDE-Net 2.0, and measure the relative error between the predicted dynamics (i.e. the output of the PDE-Net 2.0) and the actual dynamics (obtained by solving (10) using high precision numerical scheme). The relative error between the true data U and the predicted data $\tilde{U}$ is defined as $\epsilon = \frac{\|\tilde{U} - U\|_2^2}{\|U - \bar{U}\|_2^2}$, where $\bar{U}$ is the spatial average of U. The error plots are shown in Fig. 4. Some of the images of the predicted dynamics are presented in Fig. 5 and the errors maps are presented in Fig. 6. As we can see that, even trained with noisy data, the PDE-Net 2.0 is able to perform long-term prediction. Having multiple δt-blocks indeed improves predict accuracy. Furthermore, PDE-Net 2.0 performs significantly better than Frozen-PDE-Net 2.0. This suggests that when we do not know the PDE, we cannot possibly know how to properly discretize it. Therefore, for inverse problems, it is better to learn both the PDE model and its discretization simultaneously.
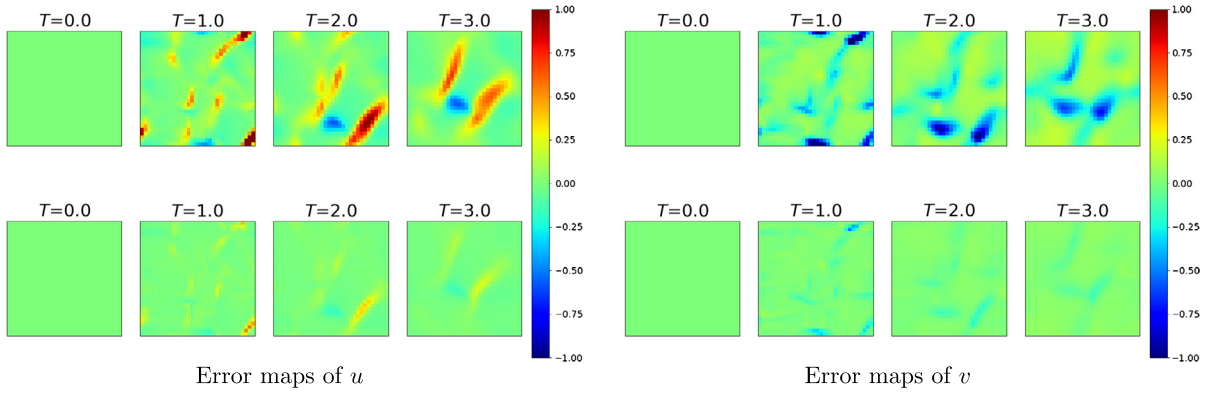
Fig. 6. Burgers' equation: The error maps of the Frozen-PDE-Net 2.0 (the first row) and PDE-Net 2.0 (second row) having 9 δt-blocks. Time step $\delta t = 0.01$.
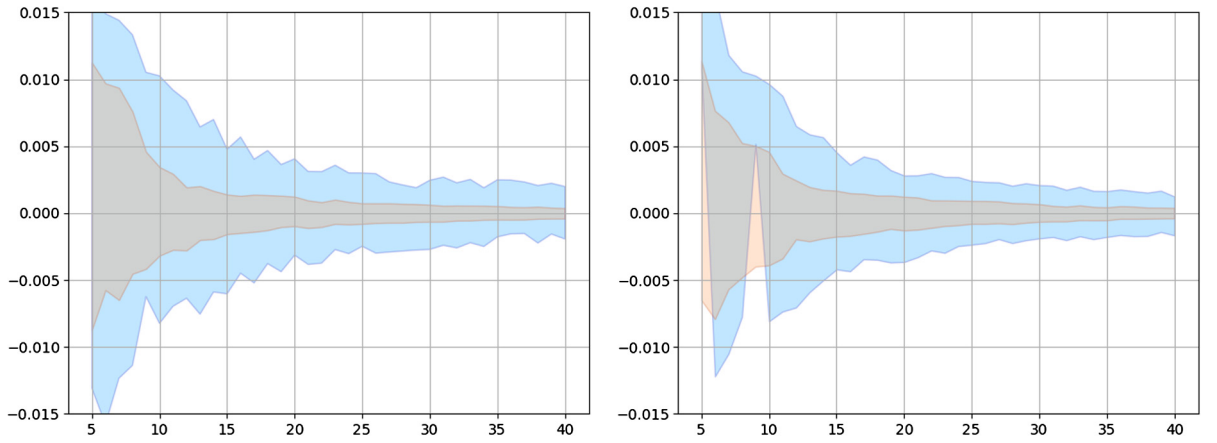

Fig. 7. Burgers' equation: The coefficients of the remainder terms for the u-component (left) and v-component (right) with the sparsity constraint on the *SymNet* (orange) and without the sparsity constraint (blue). The bands for both cases are computed based on 20 independent training.
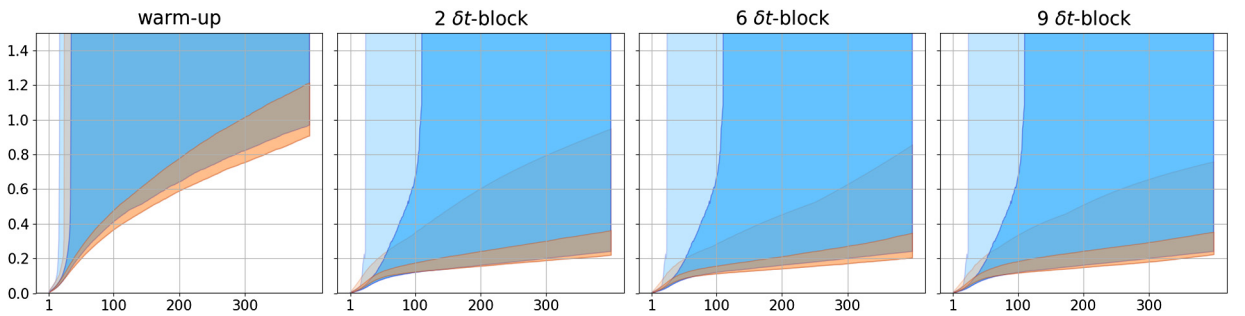

Fig. 8. Burgers' equation: Prediction errors of the PDE-Net 2.0 with (orange) and without (blue) sparsity constraint on the *SymNet*. In each plot, the horizontal axis indicates the time of prediction in the interval $(0, 400 \times \delta t] = (0, 4]$, and the vertical axis shows the relatively errors. The banded curves indicate the 25%-100% percentile of the relative errors among 1000 test samples.

3.3. Importance of L^{SymNet} and pseudo-upwind

This subsection demonstrates the importance of enforcing sparsity on the *SymNet* and using pseudo-upwind. As we can see from Fig. 7 that having sparsity constraints on the *SymNet* helps with suppressing the weights on the terms that do not exist in the Burgers' equation. Furthermore, Fig. 8 and Fig. 9 show that having sparsity constraint on the *SymNet* or using pseudo-upwind can significantly reduce prediction errors.
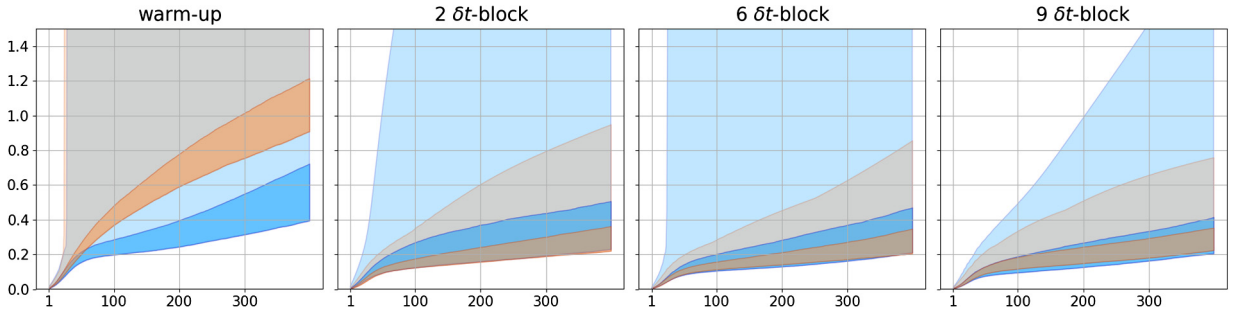
Fig. 9. Burgers' equation: Prediction errors of the PDE-Net 2.0 with (orange) and without (blue) pseudo-upwind in each δt-block. In each plot, the horizontal axis indicates the time of prediction in the interval $(0, 400 \times \delta t] = (0, 4]$, and the vertical axis shows the relatively errors. The banded curves indicate the 25%-100% percentile of the relative errors among 1000 test samples.

Table 2
PDE model identification. Note that the largest term of the remainders (i.e. the ones that are not included in the table) is $8e\text{-}4u_y$ (for Frozen-PDE-Net 2.0) and $6e\text{-}5u$ (for PDE-Net 2.0).

Correct PDE	$u_t = 0.1(u_{xx} + u_{yy})$
Frozen-PDE-Net 2.0	$u_t = 0.103u_{xx} + 0.103u_{yy}$
PDE-Net 2.0	$u_t = 0.999u_{xx} + 0.998u_{yy}$

4. Numerical studies: diffusion equation

Diffusion phenomenon has been studied in many applications in physics e.g. the collective motion of micro-particles in materials due to random movement of each particle, or modeling the distribution of temperature in a given region over time.

4.1. Simulated data, training and testing

Consider the 2-dimensional heat equation with periodic boundary condition on $\Omega = [0, 2\pi] \times [0, 2\pi]$

$$\begin{cases} \frac{\partial u}{\partial t} & = c\Delta u, (t, x, y) \in [0, 1] \times \Omega, \\ u|_{t=0} & = u_0(x, y), \end{cases} \tag{13}$$

where $c = 0.1$. The training data of the heat equation is generated by 2nd order Runge-Kutta in time with $\delta t = \frac{1}{1600}$, and central difference scheme in space on a 128×128 mesh. We then restrict the data to a 32×32 mesh. The initial value $u_0(x, y)$ is also generated from (11).

4.2. Results and discussions

The demonstration on the ability of the trained PDE-Net 2.0 to identify the PDE model is given in Table 2. As one can see from Table 2 that we can recover the terms of the heat equation with good accuracy. Furthermore, all the terms that are not included in the heat equation have much smaller weights in the *SymNet*.

We also demonstrate the ability of the trained PDE-Net 2.0 in prediction. The testing method is exactly the same as the method described in Section 3. Comparisons between PDE-Net 2.0 and Frozen-PDE-Net 2.0 are shown in Fig. 10, where we can clearly see the advantage of learning the filters. Visualization of the predicted dynamics is given in Fig. 11. All these results show that the learned PDE-Net 2.0 performs well in prediction.

5. Numerical studies: convection diffusion equation with a reactive source

Convection diffusion systems are mathematical models which correspond to the transferring of some physical quantities such as energy or materials due to diffusion and convection. Specifically, a convection diffusion system with a reactive source can be used to model a large range of chemical systems in which the transferring of materials competes with productions of materials induced by several chemical reactions.
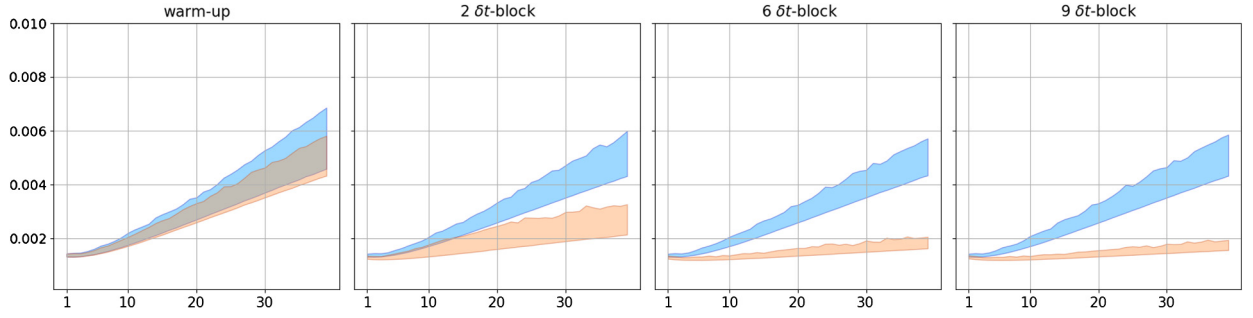
Fig. 10. Diffusion equation: Prediction errors of the PDE-Net 2.0 (orange) and Frozen-PDE-Net 2.0 (blue). In each plot, the horizontal axis indicates the time of prediction in the interval $(0, 150 \times \delta t] = (0, 1.5]$, and the vertical axis shows the relative errors. The banded curves indicate the 25%-100% percentile of the relative errors among 1000 test samples.
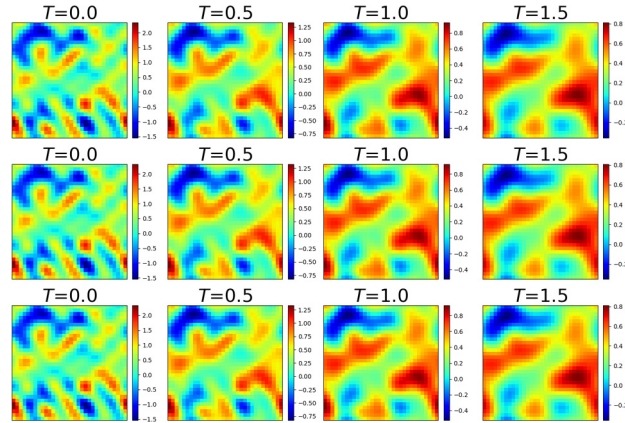

Fig. 11. Diffusion equation: The first row shows the images of the true dynamics. The second row shows the predicted dynamics using the Frozen-PDE-Net 2.0 with 9 δt-blocks. The third row shows the predicted dynamics using the PDE-Net 2.0 with 9 δt-blocks. Here, $\delta t = 0.01$.

5.1. Simulated data, training and testing

Consider a 2-dimensional convection diffusion equation with a reactive source and the periodic boundary condition on $\Omega = [0, 2\pi] \times [0, 2\pi]$:

$$
\begin{aligned}
u_t &= -uu_x - vu_y + \nu\Delta u + \lambda(A)u - \omega(A)v, \\
v_t &= -uv_x - vv_y + \nu\Delta v + \omega(A)u + \lambda(A)v, \\
A^2 &= u^2 + v^2, \omega = -\beta A^2, \lambda = 1 - A^2,
\end{aligned}
\tag{14}
$$

where $(t, x, y) \in [0, 1.5] \times \Omega$, and $\nu = 0.1, \beta = 1$. Training data is generated the same way as what we did for Burgers' equation in Section 3. A 2nd order Runge-Kutta with time step $\delta t = 1/10000$ is adopted for temporal discretization. We choose a 2nd order upwind scheme for the convection terms and the central difference scheme for Δ on a 128×128 mesh. We then restrict the data to a 32×32 mesh. Noise is added the same way as the Burgers' equation. The initial values $u_0(x, y), v_0(x, y)$ are also generated from (11).

5.2. Results and discussions

The capability of the trained PDE-Net 2.0 to identify the underlying PDE model is demonstrated in Table 3. As one can see that we can recover the terms of the reaction convection diffusion equation with good accuracy. Furthermore, all the terms that are not included in this equation have relatively small weights in the *SymNet*.

We also demonstrate the ability of the trained PDE-Net 2.0 in prediction. The testing method is exactly the same as the method described in Section 3. Comparisons between PDE-Net 2.0 and Frozen-PDE-Net 2.0 are shown in Fig. 12. Visualization of the predicted dynamics and errors maps are given in Fig. 13 and Fig. 14. Similar to what we observed in Section 3, we can clearly see the benefit from learning discretizations. PDE-Net 2.0 obtains more accurate estimations of the coefficients for the nonlinear convection terms (i.e. the term $-uu_x - vu_y$ in Table 3) and makes more accurate predictions (Fig. 12) than Frozen-PDE-Net 2.0.

Table 3
PDE model identification.

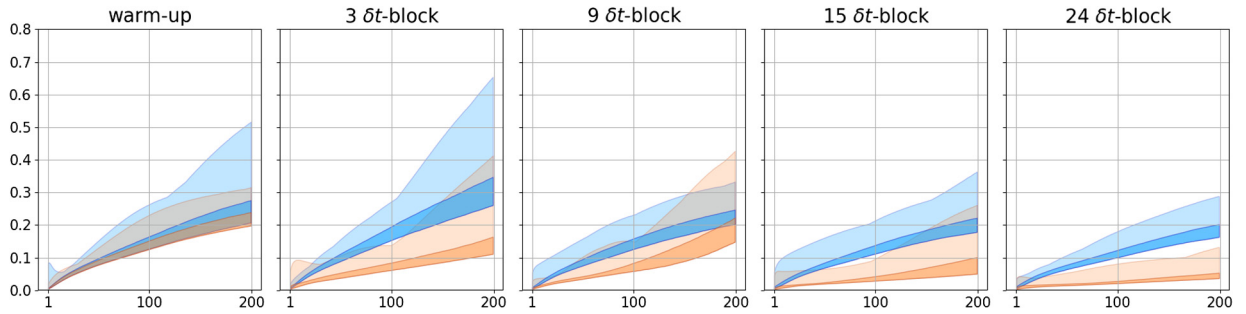| Correct PDE	$u_t = -uu_x - vu_y + 0.1\Delta u + (v - u)(u^2 + v^2) + u$
	$v_t = -uv_x - vv_y + 0.1\Delta v - (v + u)(u^2 + v^2) + v$
Frozen-PDE-Net 2.0	$u_t = -0.86uu_x - 0.90vu_y + 0.09u_{xx} + 0.09u_{yy}$
	$\quad + 1.02u^2v - 1.02u^3 - 1.01uv^2 + 1.01u + 0.99v^3$
	$v_t = -0.87uv_x - 0.85vv_y + 0.09v_{xx} + 0.09v_{yy}$
	$\quad + 1.04u^2v - 1.02uv^2 - 1.01v^3 + 0.99v - 0.99u^3$
PDE-Net 2.0	$u_t = -0.98vu_y - 0.93uu_x + 0.10u_{xx} + 0.10u_{yy}$
	$\quad - 1.05uv^2 + 0.99v^3 - 0.98u^3 + 0.98u + 0.97u^2v$
	$v_t = -0.99uv_x - 0.96vv_y + 0.10v_{yy} + 0.10v_{xx}$
	$\quad - 1.04u^2v - 1.02v^2 - 1.02uv^2 + 1.01v - 1.00u^3$

Fig. 12. Reaction convection diffusion: Prediction errors of the PDE-Net 2.0 (orange) and Frozen-PDE-Net 2.0 (blue). In each plot, the horizontal axis indicates the time of prediction in the interval $(0, 200 \times \delta t] = (0, 2]$, and the vertical axis shows the relative errors. The banded curves indicate the 25%-100% percentile of the relative errors among 1000 test samples.
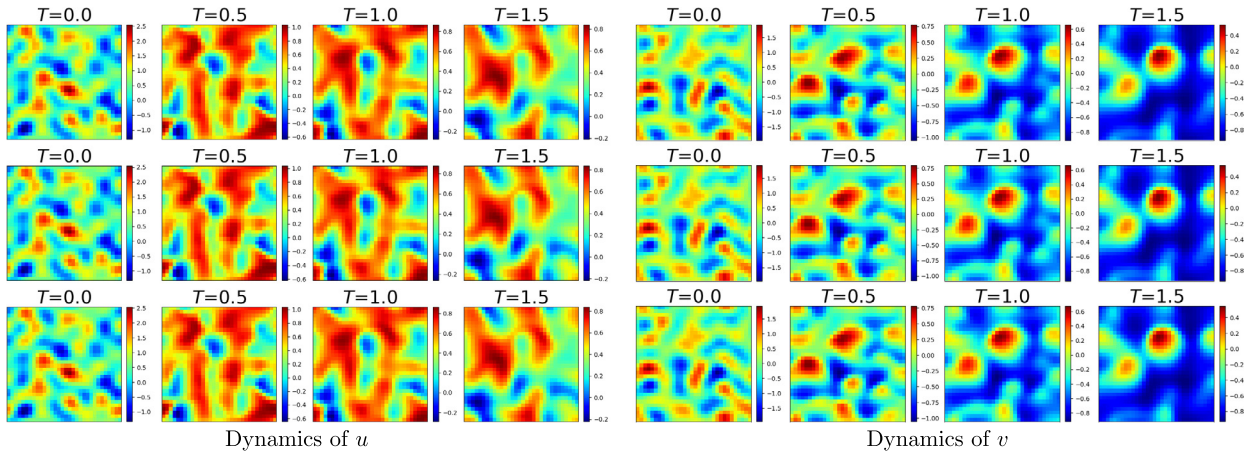

Fig. 13. Reaction convection diffusion: The first row shows the images of the true dynamics. The second row shows the predicted dynamics using the Frozen-PDE-Net 2.0 with 24 δt-blocks. The third row shows the predicted dynamics using the PDE-Net 2.0 with 24 δt-blocks. Here, $\delta t = 0.01$.

6. Conclusions and future work

In this paper, we proposed a numeric-symbolic hybrid deep network, called PDE-Net 2.0, for PDE model recovery from observed dynamic data. PDE-Net 2.0 is able to recover the analytic form of the PDE model with minor assumptions on the mechanisms of the observed dynamics. For example, it is able to recover the analytic form of Burgers' equation with good confidence without any prior knowledge on the type of the equation. Therefore, PDE-Net 2.0 has the potential to uncover potentially new PDEs from observed data. Furthermore, after training, the network can perform accurate long-term prediction without re-training for new initial conditions. The limitations and possible future extensions of the current version of PDE-Net 2.0 is twofold: 1) having only addition and multiplication in the *SymNet* may still be too restrictive, and one may include division as an additional operation in *SymNet* to further improve its expressive power; 2) using forward Euler as temporal discretization is the most straightforward treatment, while a more sophisticated temporal scheme may help with the model recovery and prediction. Both of these worth further exploration. Furthermore, we would like to apply the
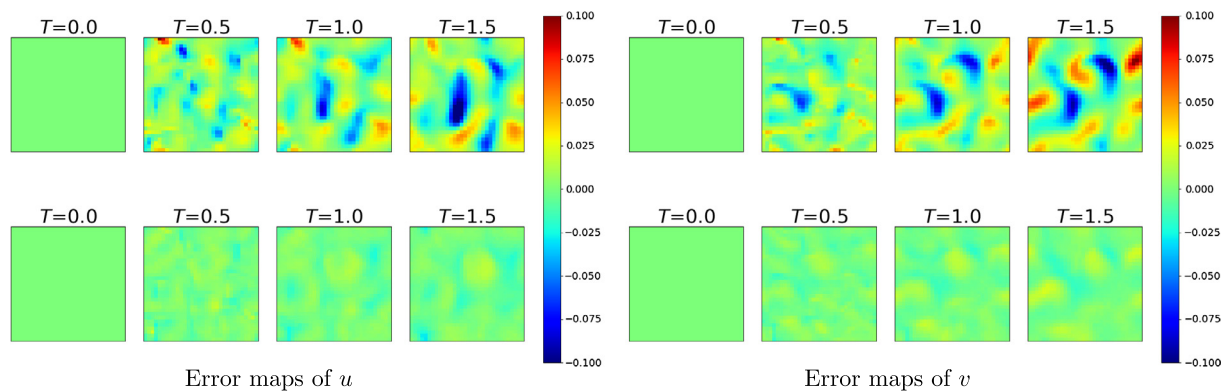
Fig. 14. Reaction convection diffusion: The error maps of the Frozen-PDE-Net 2.0 (the first row) and PDE-Net 2.0 (second row) having 24 δt-blocks. Time step $\delta t = 0.01$.

network to real biological dynamic data. We will further investigate the reliability of the network and explore the possibility to uncover new dynamical principles that have meaningful scientific explanations.

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.

Acknowledgements

References

[1] Z. Long, Y. Lu, X. Ma, B. Dong, Pde-net: learning pdes from data, in: International Conference on Machine Learning, 2018, pp. 3214–3222.
[2] C.E. Brennen, C.E. Brennen, Fundamentals of Multiphase Flow, Cambridge University Press, 2005.
[3] M. Efendiev, Mathematical modeling of mitochondrial swelling.
[4] J. Bongard, H. Lipson, Automated reverse engineering of nonlinear dynamical systems, Proc. Natl. Acad. Sci. USA 104 (24) (2007) 9943–9948.
[5] M. Schmidt, H. Lipson, Distilling free-form natural laws from experimental data, Science 324 (5923) (2009) 81–85.
[6] M. Raissi, G.E. Karniadakis, Hidden physics models: machine learning of nonlinear partial differential equations, J. Comput. Phys. 357 (2018) 125–141.
[7] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (part II): data-driven discovery of nonlinear partial differential equations, arXiv preprint, arXiv:1711.10566.
[8] 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. USA (2016) 201517384.
[9] H. Schaeffer, Learning partial differential equations via data discovery and sparse optimization, Proc. R. Soc. A 473 (2197) (2017) 20160446.
[10] S.H. Rudy, S.L. Brunton, J.L. Proctor, J.N. Kutz, Data-driven discovery of partial differential equations, Sci. Adv. 3 (4) (2017) e1602614.
[11] H. Chang, D. Zhang, Identification of physical processes via combined data-driven and data-assimilation methods, arXiv preprint, arXiv:1810.11977.
[12] H. Schaeffer, G. Tran, R. Ward, L. Zhang, Extracting structured dynamical systems using sparse optimization with very few samples, arXiv preprint, arXiv:1805.04158.
[13] 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.
[14] E. de Bezenac, A. Pajot, P. Gallinari, Deep learning for physical processes: incorporating prior scientific knowledge, arXiv preprint, arXiv:1711.07970.
[15] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[16] K. He, X. Zhang, S. Ren, J. Sun, Identity mappings in deep residual networks, in: European Conference on Computer Vision, Springer, 2016, pp. 630–645.
[17] Y. Chen, W. Yu, T. Pock, On learning optimized reaction diffusion processes for effective image restoration, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 5261–5269.
[18] E. Weinan, A proposal on machine learning via dynamical systems, Commun. Math. Stat. 5 (1) (2017) 1–11.
[19] E. Haber, L. Ruthotto, Stable architectures for deep neural networks, Inverse Probl. 34 (1) (2017) 014004.
[20] S. Sonoda, N. Murata, Double continuum limit of deep neural networks, in: ICML Workshop Principled Approaches to Deep Learning, 2017.
[21] Y. Lu, A. Zhong, Q. Li, B. Dong, Beyond finite layer neural networks: bridging deep architectures and numerical differential equations, in: International Conference on Machine Learning, 2018, pp. 3282–3291.
[22] B. Chang, L. Meng, E. Haber, F. Tung, D. Begert, Multi-level residual networks from dynamical systems view, in: ICLR, 2018.
[23] T.Q. Chen, Y. Rubanova, J. Bettencourt, D. Duvenaud, Neural ordinary differential equations, arXiv preprint, arXiv:1806.07366.
[24] T. Qin, K. Wu, D. Xiu, Data driven governing equations approximation using deep neural networks, arXiv preprint, arXiv:1811.05537.
[25] S. Wiewel, M. Becher, N. Thuerey, Latent-space physics: towards learning the temporal evolution of fluid flow, arXiv preprint, arXiv:1802.10123.

[26] B. Kim, V.C. Azevedo, N. Thuerey, T. Kim, M. Gross, B. Solenthaler, Deep fluids: a generative network for parameterized fluid simulations, arXiv preprint, arXiv:1806.02071.
[27] K. Duraisamy, Z.J. Zhang, A.P. Singh, New approaches in turbulence and transition modeling using data-driven techniques, in: 53rd AIAA Aerospace Sciences Meeting, 2015, p. 1284.
[28] K. Duraisamy, G. Iaccarino, H. Xiao, Turbulence modeling in the age of data, Annu. Rev. Fluid Mech. 51 (2019) 357–377.
[29] C. Ma, J. Wang, et al., Model reduction with memory and the machine learning of dynamical systems, arXiv preprint, arXiv:1808.04258.
[30] W. Noid, J.-W. Chu, G.S. Ayton, V. Krishna, S. Izvekov, G.A. Voth, A. Das, H.C. Andersen, The multiscale coarse-graining method. I. A rigorous bridge between atomistic and coarse-grained models, J. Chem. Phys. 128 (24) (2008) 244114.
[31] L. Zhang, J. Han, H. Wang, R. Car, E. Weinan, Deep potential molecular dynamics: a scalable model with the accuracy of quantum mechanics, Phys. Rev. Lett. 120 (14) (2018) 143001.
[32] J.-F. Cai, B. Dong, S. Osher, Z. Shen, Image restoration: total variation, wavelet frames, and beyond, J. Am. Math. Soc. 25 (4) (2012) 1033–1089.
[33] B. Dong, Q. Jiang, Z. Shen, Image restoration: wavelet frame shrinkage, nonlinear evolution pdes, and beyond, Multiscale Model. Simul. 15 (1) (2017) 606–660.
[34] I. Daubechies, Ten Lectures on Wavelets, vol. 61, SIAM, 1992.
[35] S. Mallat, A Wavelet Tour of Signal Processing, Elsevier, 1999.
[36] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, Q. Liao, Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review, Int. J. Autom. Comput. 14 (5) (2017) 503–519.
[37] U. Shaham, A. Cloninger, R.R. Coifman, Provable approximation properties for deep neural networks, Appl. Comput. Harmon. Anal. 44 (3) (2018) 537–557.
[38] H. Montanelli, Q. Du, Deep relu networks lessen the curse of dimensionality, arXiv preprint, arXiv:1712.08688.
[39] Q. Wang, et al., Exponential convergence of the deep neural network approximation for analytic functions, arXiv preprint, arXiv:1807.00297.
[40] J. He, L. Li, J. Xu, C. Zheng, Relu deep neural networks and linear finite elements, arXiv preprint, arXiv:1807.03973.
[41] G. Martius, C.H. Lampert, Extrapolation and learning equations, arXiv preprint, arXiv:1610.02995.
[42] S. Sahoo, C. Lampert, G. Martius, Learning equations for extrapolation and control, in: International Conference on Machine Learning, 2018, pp. 4439–4447.
[43] X.-D. Liu, S. Osher, T. Chan, Weighted essentially non-oscillatory schemes, J. Comput. Phys. 115 (1) (1994) 200–212.
[44] C.-W. Shu, Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws, in: Advanced Numerical Approximation of Nonlinear Hyperbolic Equations, Springer, 1998, pp. 325–432.
[45] R.J. LeVeque, Finite Volume Methods for Hyperbolic Problems, vol. 31, Cambridge University Press, 2002.