# MULTISCALE AND NONLOCAL LEARNING FOR PDES USING DENSELY CONNECTED RNNS[*]

RICARDO A. DELGADILLO[†], JINGWEI HU[‡], AND HAIZHAO YANG[§]

**Abstract.** Learning time-dependent partial differential equations (PDEs) that govern evolutionary observations is one of the core challenges for data-driven inference in many fields. In this work, we propose to capture the essential dynamics of numerically challenging PDEs arising in multiscale modeling and simulation - kinetic equations. These equations are usually nonlocal and contain scales/parameters that vary by several orders of magnitude. We introduce an efficient framework, Densely Connected Recurrent Neural Networks (DC-RNNs), by incorporating high-order numerical schemes of time-dependent PDEs into RNN structure design to identify analytic representations of multiscale and nonlocal PDEs from discrete-time observations generated from heterogeneous experiments. If present in the observed data, our DC-RNN can capture transport operators, nonlocal projection or collision operators, equilibrium state dynamics (macroscopic diffusion limit), and other dynamics. We provide numerical results demonstrating the advantage of our proposed framework over existing methods.

**Key words.** Multiscale; Nonlocal; time-dependent PDE Recovery; Machine Learning; Densely Connected RNN.

**AMS subject classifications.** 35C20; 35C99; 68T99.

## 1. Introduction.

Data-driven discovery of partial differential equations is experiencing unprecedented development over the past few years, wherein various kinds of PDEs (featuring e.g., time dependence and nonlinearity) have been studied. In this work, we consider the learning problem for a class of PDEs that involve multiple time/spatial scales and nonlocal operators – kinetic equations. These are an important class of equations in multiscale modeling hieriarchy which bridges microscopic atomistic models (such as N-body Newton equations) and macroscopic continuum models (such as Navier-Stokes equations). For a variety of scientific problems ranging from gas/plasma dynamics, radiative transfer to social/biological systems, kinetic equations have demonstrated their ability to accurately model the dynamics of many complex systems [41]. To the best of our knowledge, learning of multiscale kinetic equations, albeit important, has never been explored in the literature.

Specifically, we are interested in developing an efficient symbolic neural network to fit time-dependent data for a large class of multiscale kinetic equations. The overall goal is to identify an explicit formula of the map $\mathcal{F}$ that determines the evolution $u(\boldsymbol{x},t) \to u(\boldsymbol{x},t+\Delta t)$ for $\boldsymbol{x} \in \Omega$ and $\Delta t > 0$. Therefore, a symbolic neural network $\mathcal{F}(u;\boldsymbol{\theta},w_\varepsilon)$ with parameters $\boldsymbol{\theta}$ and $w_\varepsilon$ is constructed and the following loss function is minimized to find the best parameter set:

$$L(\boldsymbol{\theta},w_\varepsilon) = \frac{1}{N_t}\sum_{j=1}^{N_t}\left\| u(\boldsymbol{x},t_{j+1}) - u(\boldsymbol{x},t_j) - \int_{t_j}^{t_j+\Delta t} \mathcal{F}(u(\boldsymbol{x},s);\boldsymbol{\theta},w_\varepsilon)\,ds \right\|_{L^1(\Omega)}. \tag{1.1}$$

$\mathcal{F}$ approaches the correct model as $L(\boldsymbol{\theta},w_\varepsilon) \to 0$. The choice of the norm above is flexible. In this paper, we focus on the $L^1$-norm because our numerical experiments show that it is slightly better than others, e.g., the $L^2$-norm. Due to the multiscale and nonlocal feature of our target equations, existing learning schemes may not be efficient. We will propose novel symbolic neural networks, new formulations of the loss function in (1.1), and new regularization methods in this paper to tackle this challenge.

Our first main contribution is a new symbolic neural network $\mathcal{F}(u;\boldsymbol{\theta},w_\varepsilon)$ build with multiscale and nonlocal features. The key idea for capturing multiscale phenomena is to construct $\mathcal{F}$ as a sum of different components at different scales of order $\varepsilon_{pred}^n$, where $n$ is an integer degree and $\varepsilon_{pred}$ is a trainable multiscale separator defined by:

$$\varepsilon_{pred}(w_\varepsilon) = \frac{1}{2}(\tanh(w_\varepsilon)+1) \tag{1.2}$$

with $w_\varepsilon$ as a trainable parameter. In particular, we propose

[*]Submitted to the editors DATE.

**Funding:** RD was partially supported by the Fog Research Institute under contract no. FRI-454. JH was partially supported by NSF CAREER grant DMS-1654152. HY was partially supported by NSF CAREER grant DMS-1945029.

[†]Department of Civil and Environmental Engineering, National University of Singapore, Singapore (ceerad@nus.edu.sg).
[‡]Department of Mathematics, Purdue University, West Lafayette, IN 47906, USA (jingweihu@purdue.edu).
[§]Department of Mathematics, Purdue University, West Lafayette, IN 47906, USA (haizhao@purdue.edu).

$$\mathcal{F}(u;\boldsymbol{\theta},w_\varepsilon)=\sum_{n=0}^{N}\frac{1}{\varepsilon^n_{pred}(w_\varepsilon)}\mathcal{F}^n(u;\boldsymbol{\theta}_n). \tag{1.3}$$

where $\boldsymbol{\theta}:=(\boldsymbol{\theta}_1,\boldsymbol{\theta}_2,\cdots,\boldsymbol{\theta}_n)$. Thus, unlike conventional deep learning recovery algorithms as in [34, 35, 34, 27, 14, 43, 46], our algorithm is aware of different scales and thus more accurately captures different components at scale $\mathcal{O}(\varepsilon^n_{pred})$.

The key idea to make $\mathcal{F}(u;\boldsymbol{\theta},w_\varepsilon)$ capable of capturing nonlocal phenomena is to incorperate nonlocal operators in $\mathcal{F}^n$ in (1.3) to construct $\mathcal{F}$. Conventionally, $\mathcal{F}$ is typically constructed as a linear combination of mathematical operators in a pre-specified dictionary, and the combination coefficients are learned via minimizing (1.1) with sparsity regularization to obtain sparse linear combinations as in [19, 37, 30, 6, 46]. For high-dimensional problems, constructing such a dictionary can be very costly. Hence, we will apply symbolic recurring neural network (RNN) of mathematical operators as in [28, 27] without specifying a large dictionary. Intuitively, due to the high expressiveness of our symbolic RNNs, the class of RNNs with different parameters can form a large dictionary without pre-specifying a costly dictionary. It might be computationally more efficient to use symbolic RNNs to classify the dynamics of data and choose a trainable symbolic model to model data.

The most basic elements of our RNN are a set of (either local or nonlocal) basic mathematical operators $\mathcal{A}_1,\cdots,\mathcal{A}_n$ from a function space to another function space commonly used in dynamics modeling for kinetic equations, such as transport, collision, and other nonlocal operators. The trainable compositions of these basic operators form a basis of our RNN, i.e., each term $\mathcal{F}^n$ in (1.3) is a trainable linear combination of the compositions defined below:

$$\mathcal{A}_{\pi(1)}\circ\cdots\circ\mathcal{A}_{\pi(m)}, \tag{1.4}$$

where $\pi=\big(\pi(1),\cdots,\pi(m)\big)\in\mathbb{Z}^m$ with entries in $\{1,\cdots,n\}$. More precisely, we have

$$\mathcal{F}^n(u;\boldsymbol{\theta}_n)=\sum_{m\geq1}\sum_{\pi\in\mathcal{D}}a_{\pi(1),\cdots,\pi(m)}(\boldsymbol{\theta}_n)\mathcal{A}_{\pi(1)}\circ\cdots\circ\mathcal{A}_{\pi(m)}(u), \tag{1.5}$$

where coefficients $a_{\pi(1),\cdots,\pi(m)}(\boldsymbol{\theta}_n)$ depend on trainable parameters $\boldsymbol{\theta}_n$, and $\mathcal{D}$ is a set of index vectors $\pi$ specified by our symbolic RNN as we shall see later. Similar to polynomial regression [9, 13], our RNN returns a multivariate polynomial of the operators $\mathcal{A}_1,\cdots,\mathcal{A}_n$. Due to the expressive power of neural networks [44, 38, 29, 23, 31, 24], our symbolic RNN of a small size can generate a sufficiently large index vector set $\mathcal{D}$. The formulation in (1.5) is also natural in physics, equations derived from asymptotic analysis often have recursive structure similar to the compositional operators in (1.5), e.g., see [39].

Our second main contribution is to propose novel loss functions based on high-order implicit-explicit schemes to discretize of the integral in (1.1). The most typical numerical method, the forward-Euler scheme, results in the loss function:

$$L(\boldsymbol{\theta},w_\varepsilon)=\frac{1}{N_t}\sum_{j=1}^{N_t}\left\|u(\boldsymbol{x},t_{j+1})-u(\boldsymbol{x},t_j)-\Delta t\mathcal{F}(u(\boldsymbol{x},t_j);\boldsymbol{\theta},w_\varepsilon)\right\|_{L^1(\Omega)}, \tag{1.6}$$

which is commonly used in the discovery of governing equations. Though explicit higher order approximations using multistep methods have been investigated in [34, 20, 33, 12], there is no existing research on the effectiveness of implicit-explicit schemes in the literature of discovering governing equations. To predict future state dynamics, we propagate data using Implicit-Explicit Runge-Kutta (IMEX) schemes. We use IMEX schemes as they are especially suited to solve stiff problems in kinetic theory [25, 3], and they are able to describe systems either depending on the past or future states. The collection of RNNs together with our propagation scheme will make up our “densely connected recurrent neural network” (DC-RNN).

Our third main contribution is to propose physics-based regularization to the loss function in (1.1) to improve optimization efficiency and avoid over-fitting. First, a physically correct model is usually described with a small number of mathematical operators in (1.5), while an over-fitting model would have a large number of operators for a better fitting capacity. Thus, inspired by the lasso approaches in [40, 5, 47], we propose sparse regularization to avoid over-fitting and remove undesirable features in the governing equation, e.g., adding a $L^1$-norm penalty term to the coefficients in (1.5). Second, a micro-macro decomposition of kinetic equations [16, 18, 17, 22] are applied to transfer a challenging recovery problem with a single PDE to an easier recovery problem with a coupled PDE system, enforcing our recovery results to be more physically meaningful. Furthermore, the macroscopic part, denoted as $g$, satisfies

$$\langle g \rangle := \int_{[-1,1]} g(v, x, t) dv = 0, \tag{1.7}$$

which will be used as a constraint of our recovery. Finally, in most cases, kinetic equations have spatial-dependent coefficients, which motivates us to design spatial-dependent parameters $\boldsymbol{\theta}(x)$ in (1.5) and the regularity in terms of $x$ can also be considered as a regularization penalty.

To summarize, the main highlights of our learning algorithm are as follows:

- DC-RNN built for transport, collision, and nonlocal operators typically involved in kinetic equations.
- Multiscale-aware RNN structures and learning rates for the recovery of time-dependent PDEs.
- Novel optimization loss function inspired by high-order IMEX for stiff equations.
- Physics-aware loss function and regularization specialized for kinetic equations.
- Efficient arithmetic and memory cost.

We structure this manuscript as follows. In Section 2, an exemplary PDE for our learning problem is introduced to motivate our algorithm. In Section 3, we mathematically formulate an ansatz that we will use to fit data to PDEs. In Section 4, our physics-aware loss function is introduced to learn PDEs from data. In Section 5, we will carry out several numerical experiments to test our algorithm. Finally, concluding remarks are made in the Section 6.

## 2. Model Equation: the Linear Transport Kinetic Equation.

We now present a model equation, the linear transport equation, to motivate our learning algorithm. The linear transport equation is a prototype kinetic equation describing particles such as neutrons or photons interacting with a background medium [8, 10]. This equation highlights some of the challenging aspects that an efficient learning algorithm should account for. That is, our model equation will allow us to understand the hypothesis space (the set of functions describing kinetic equations) better. This will lead us to devise ways to capture multiple scales, nonlocal operators, and regularity conditions. In addition, we will be able to discern appropriate numerical techniques needed to carry out our learning algorithm.

In the simple 1D case, the linear transport equation reads

$$\partial_t f + \frac{1}{\varepsilon} v \partial_x f = \frac{\sigma^S}{\varepsilon^2}(\langle f \rangle - f) - \sigma^A f + G, \tag{2.1}$$

where $f = f(t, x, v)$ is the probability density function of time $t \geq 0$, position $x \in \Omega \subset \mathbb{R}$, and velocity $v \in [-1, 1]$; $\langle \cdot \rangle := \frac{1}{2}\int_{-1}^{1} \cdot \, dv$ is a projection operator; $\sigma^S(x)$ and $\sigma^A(x)$ are the scattering and absorption coefficients; and $G(x)$ is a given source. Finally, $\varepsilon$ is a dimensionless parameter indicating the strength of the scattering. Indeed, when $\varepsilon \sim O(1)$, the equation (2.1) is in the fully kinetic regime (all operators balance); when $\varepsilon \to 0$, the scattering is so strong that (2.1) approaches a diffusion limit. To see this, consider the so-called micro-macro decomposition of $f$:

$$f = \rho + \varepsilon g, \quad \rho := \langle f \rangle, \tag{2.2}$$

where $\rho$ is the macro part (density) of the solution, and $g$ is the micro part. A crucial condition we use is

$$\langle g \rangle = 0. \tag{2.3}$$

Equation (2.3) is the conservation condition and will be numerically indispensable since it allows us to impose exact conditions satisfied by kinetic equations. Substituting (2.2) into (2.1), one can derive the following coupled system for $\rho$ and $g$, equivalent to (2.1):

$$\partial_t \rho = -\partial_x \langle vg \rangle - \sigma^A \rho + G, \tag{2.4}$$

$$\partial_t g = -\frac{1}{\varepsilon}\left(\mathcal{I} - \langle\ \rangle\right)(v \partial_x g) - \frac{1}{\varepsilon^2} v \partial_x \rho - \frac{\sigma^S}{\varepsilon^2} g - \sigma^A g, \tag{2.5}$$

where $\mathcal{I}$ denotes the identity operator.

In (2.5), if $\varepsilon \to 0$, one obtains

$$g = -\frac{1}{\sigma^S} v\partial_x \rho + \mathcal{O}(\varepsilon), \tag{2.6}$$

which, when substituted into (2.4), yields

$$\partial_t \rho = \partial_x \left( \frac{1}{3\sigma^S} \partial_x \rho \right) - \sigma^A \rho + G + \mathcal{O}(\varepsilon). \tag{2.7}$$

So $\rho$ follows the dynamics of a diffusion equation. We now go through a few things that we can learn from the linear transport equation following the notations used in Section 1.

• **Involved Basic Mathematical Operators: Identity, Advection, and Projection.** Notice that each of Equations (2.1), (2.4), and (2.5) can be recovered from the ansatz:

$$\begin{aligned}\partial_t u &= a_1\mathcal{A}_1(g) + a_2\mathcal{A}_2(g) + a_3\mathcal{A}_3(g) + \sum_{i=1}^{3}\sum_{j=1}^{3} a_{i,j}\mathcal{A}_i \circ \mathcal{A}_j(g) \\ &\quad + b_1\mathcal{A}_1(\rho) + b_2\mathcal{A}_2(\rho) + b_3\mathcal{A}_3(\rho) + \sum_{i=1}^{3}\sum_{j=1}^{3} b_{i,j}\mathcal{A}_i \circ \mathcal{A}_j(\rho) + B\end{aligned} \tag{2.8}$$

for $u = f$, $g$, or $\rho$. For example, the equation for $g(v, x, t)$ can be recovered provided

$$\mathcal{A}_1 = \mathcal{I}, \qquad \mathcal{A}_2 = v\partial_x, \qquad \mathcal{A}_3 = \langle \cdot \rangle, \tag{2.9}$$

with coefficients

$$\begin{aligned} a_1 = \frac{\sigma^S(x)}{\varepsilon^2} - \sigma^A(x), \qquad a_2 = -\frac{1}{\varepsilon}, \qquad a_{3,2} = \frac{1}{\varepsilon}, \qquad b_2 = -\frac{1}{\varepsilon^2} \\ a_3 = 0, \qquad a_{i\neq 3, j\neq 2} = 0, \qquad b_{i,j} = 0, \qquad B = 0. \end{aligned} \tag{2.10}$$

Thus, at the very minimum, our hypothesis space in Equation (1.3) should involve operators in (2.9). We expect to see these operators for general kinetic equations. Potentially one can also have cubic or higher order nonlinearities in our hypothesis space. Therefore, we want to generalize Equation (2.8) to involve greater number of compositions.

• **Functions of** $x$**.** Equations (2.1), (2.4), and (2.5) involve the functions $\sigma^A(x)$, $\sigma^S(x)$, and $G(x)$. Therefore the coefficients $\{a_i, a_{i,j}, b_i, b_{i,j}, B\}$ should be allowed to depend on $x$.

• **Scale Disparity.** If we want to determine the correct order of each term, then we need to make an asymptotic expansion:

$$\begin{aligned} a_i &= a_i^0 + \frac{1}{\varepsilon}a_i^1 + \frac{1}{\varepsilon^2}a_i^2, \qquad & a_{i,j} &= a_{i,j}^0 + \frac{1}{\varepsilon}a_{i,j}^1 + \frac{1}{\varepsilon^2}a_{i,j}^2, \\ b_i &= b_i^0 + \frac{1}{\varepsilon}b_i^1 + \frac{1}{\varepsilon^2}b_i^2, \qquad & b_{i,j} &= b_{i,j}^0 + \frac{1}{\varepsilon}b_{i,j}^1 + \frac{1}{\varepsilon^2}b_{i,j}^2, \\ B &= B^0 + \frac{1}{\varepsilon}B^1 + \frac{1}{\varepsilon^2}B^2, \end{aligned} \tag{2.11}$$

where it is understood that the upper index labels the order of the scale. The multiscale phenomenon here is the main motivation of the multiscale model in Equation (1.3).

• **Exact Conditions.** Typically, adding regularization to machine learning problems can vastly improve the outcome of the prediction. There is one obvious constraint for our target kinetic equation: Equation (2.3). An added feature about this condition is that it is independent of $\varepsilon$ and thus helpful for modeling dynamics between the small and large scale limits. We also note that for $\varepsilon \ll 1$, potentially our learning algorithm could recover either Equation (2.7) or (2.4). Thus, we should not conclude that our algorithm made an error. Of course, Equation (2.7) is lower dimensional, thus it would be a welcomed surprise to be able to obtain (2.7).

• **Sparsity.** The large number of basis terms in our hypothesis space means that we might have overfitting issues. Thus, the following sparsity regularization term could be considered:

$$\sum_n \left( (\sum_i ||a_i^n||_{L^1} + ||b_i^n||_{L^1} + ||B^n||_{L^1}) + (\sum_{i,j} ||a_{i,j}^n||_{L^1} + ||b_{i,j}^n||_{L^1}) \right), \tag{2.12}$$

which can be enforced by adding the following regularization term to our loss function in Equation (1.1):

$$||\boldsymbol{\theta}(x)||_{L^1}, \tag{2.13}$$

since $\boldsymbol{\theta}(x)$ is the actual parameters to be optimized in our model in Equation (1.3).

• **Numerical Techniques.** Finally, we need to consider numerical methods for arriving at the correct set of trainable parameters. For this reason, we will use the class of implicit-explicit Runge-Kutta schemes for propagating $u(t_n) \to u(t_{n+1})$, where the time rate of change is given by an ansatz like Equation (2.8). We will use gradient descent, specifically the Adam algorithm, to update trainable parameters.

In sum, the discussion above illustrates the motivation of our optimization problem, model design, and regularization terms introduced in Section 1.

## 3. Formulating an Ansatz to Fit Data to Kinetic Equations.

In this section, we will construct an ansatz capable of representing Equations (2.4), (2.5), and other kinetic equations. For simplicity, let us focus on the case when the spatial variable $x$ is one-dimensional. It is easy to generalize the evaluation to high-dimensional cases. We start by introducing notations which will be used through the paper.

**Notation.** The functions involved in (2.4) or (2.5) are multidimensional, e.g. $\rho = \rho(x,t)$ and $g = g(v,x,t)$. The values of $\rho$ and $g$ will be defined on a mesh $(x_j, t_k)$ and $(v_i, x_j, t_k)$ for $i \in 1, \cdots N_v$, $j \in 1, \cdots N_x$, and $k \in 1, \cdots N_t$. To further simplify the notation, we will use $u := u_{i,j}$ to denote $u$ as a scalar function evaluated at the $(i,j)$-th position corresponding to $(v_i, x_j)$. The upper index $n$ in $u^n := u(\cdot, t_n)$ will correspond to time with $u^{1:N_t} := \{u(\cdot, t_i);\text{ for } i \in 1 \cdots N_t\}$ denoting $u$ evaluated at a time sequence. Matrices will be written with capital letters while operators applied to the data will mainly be written using script letters.

### 3.1. Operator Evaluation.

We will describe the evaluation of commonly used operators in Equations (2.4), (2.5), and other kinetic equations below.

*1) Identity operator.* The identity operator is defined by

$$\mathcal{I}(u) := u. \tag{3.1}$$

The evaluation of $\mathcal{I}(u)$ at the point $(v_i, x_j)$ simply follows $\mathcal{I}(u)_{i,j} := u_{i,j}$.

*2) Pseudo-upwind for the advection operator.* We define the advection operator acting on $u$ as the dot-product:

$$v \cdot \nabla_x u, \tag{3.2}$$

where $v$ is a velocity distribution. We note that many stable schemes use an upwind stencil for the advection operator. The first-order upwind stencil gives:

$$\partial_x u_{i,j}^+ = \frac{u_{i,j+1} - u_{i,j}}{\Delta x} \qquad \text{for } v > 0,$$

$$\partial_x u_{i,j}^- = \frac{u_{i,j} - u_{i,j-1}}{\Delta x} \qquad \text{for } v < 0,$$

and

$$v\partial_x u_{i,j} = v_- \partial_x u_{i,j}^+ + v_+ \partial_x u_{i,j}^-,$$

which is the evaluation of the advection operator in (3.2) at the point $(v_i, x_j)$ in the one-dimensional case. This stencil is suitable for a first-order-in-time IMEX-scheme. For higher-order IMEX schemes, one should use higher-order stencils.

*3) Projection operators.* We define the projection operator as an integral with respect to the variable $v$ of a function $u(v,x)$. In one-dimension, we have:

$$\langle u \rangle := \frac{1}{2}\int_{-1}^{1} u(v,x)\ dv, \tag{3.3}$$

which can be discretized as a finite sum using Gaussian quadrature:

$$\langle u \rangle_j \approx \frac{1}{2}\sum_{i=1}^{N_v} u_{i,j}\cdot w_i$$

with quadrature weights $\{w_i\}$. Note that the data corresponding to $u$ is represented by a two-index tensor $u_{i,j}$ with $i,j$ corresponding to the values $v_i$ and $x_j$, respectively. The above quadrature maps $u$ to a one-index tensor $\langle u \rangle_j$. To make dimensions consistent, we extend this to a two-index tensor by $\langle u \rangle_{i,j} := \langle u \rangle_j$ for each $i$.

*4) Other differential operators.* Higher-order differential operators such as the Laplacian will be computed by using central difference formulas.

## 3.2. Ansatz for Fitting PDEs to Data.

In this section, we will form an ansatz that will be used to fit a PDE to data, i.e., identifying the governing PDE to which the observed data is a discrete solution. We will consider the following two typical examples for simplicity. The generalization to other cases is simple.

**Scalar equation ansatz.** Let us consider a first-order in time PDE, then the equation ansatz is built as

$$\partial_t u = \mathcal{F}(u) \tag{3.4}$$

with $F$ split into $M$ multiscale components following our main model in (1.3):

$$\mathcal{F} := \mathcal{F}^0(u) + \frac{1}{\varepsilon_{pred}}\mathcal{F}^1(u) + \frac{1}{\varepsilon_{pred}^2}\mathcal{F}^2(u) + \cdots + \frac{1}{\varepsilon_{pred}^M}\mathcal{F}^M(u).$$

The integer $M$ depends on the number of multiscale components for the problem being considered. If one only expects one fast scale and one slow scale component, $M$ is set to $M = 1$. For slow, medium, and fast scales, $M$ is set to $M = 2$, etc. $\varepsilon_{pred}$ is a learnable scaling number defined in (1.2) restricted to $0 < \varepsilon_{pred} \leq 1$ but not necessarily equal to $\varepsilon$. The operators $\mathcal{F}^0, \mathcal{F}^1, \mathcal{F}^2, \cdots$ will be differential operators acting on $u$ and constructed as in (1.5). The construction detail will be provided in the next section.

**Two-component vector equation ansatz.** For vectorized equations, we build an ansatz for each component individually as

$$\begin{aligned} \partial_t g = \mathcal{F}_1(g,\rho) &= \sum_{m=0}^{M} \frac{1}{\varepsilon_{pred}^m}(\mathcal{F}_{1,1}^m(g) + \mathcal{F}_{1,2}^m(\rho)) \\ \partial_t \rho = \mathcal{F}_2(g,\rho) &= \sum_{m=0}^{M} \frac{1}{\varepsilon_{pred}^m}(\mathcal{F}_{2,1}^m(g) + \mathcal{F}_{2,2}^m(\rho)). \end{aligned} \tag{3.5}$$

The $\mathcal{F}_{q,p}^m$ are generally different operators for each $q$,$p$, and $m$ following the construction in (1.5). Each $\mathcal{F}_{q,p}^m$ has an individual set of network parameters. $\varepsilon_{pred}$ is a learnable scaling number as in the previous example. For the remainder of the manuscript, $\mathcal{F}_1$ will denote the right hand side of the $g$-equation. $\mathcal{F}_2$ will denote the right hand side of the $\rho$-equation. The construction of $\mathcal{F}_{q,p}^m$ using an RNN structure will be presented in detail in the next section.

*Remark* 3.1. Equation (3.5) is our chosen ansatz. There are many alternative ways to construct an ansatz. For example, we only consider the linear combination of $\mathcal{F}_{q,p}^m$ and it is also possible to explore the products of $\mathcal{F}_{q,p}^m$. We leave the exploration of different ansatz to the reader.

**3.3. Building a Dictionary Using RNNs.** We construct the operators $\mathcal{F}^m$ in Equation (3.4) for the single-component case. For the multicomponent case as in (3.5), we construct $\mathcal{F}^m_{q,p}$ in the same manner. The only difference is that each $F^m_{q,p}$ will have a different set of parameters depending on $(q,p)$. We will omit the $(q,p)$ index for clarity. To begin with, we will need to supply the RNN with a few basic mathematical operators as mentioned in the introduction. In particular, we consider operators $\mathcal{A}_1(u) = \mathcal{I}(u)$, $\mathcal{A}_2(u) = v\cdot\nabla u$, and $\mathcal{A}_3(u) = \langle u\rangle$ discussed in Section 3.1. It is potentially better to include more basic mathematical operators such as $\mathcal{A}_4(u) = \nabla u$, $\mathcal{A}_5(u) = g^2\cdot u$, $\mathcal{A}_6(u) = \exp\left(-(u)^2\right)$, etc. Weather or not to include more basic mathematical operators will be a choice left to the reader.

Next, a symbolic RNN will be introduced to generate a complicated operator $\mathcal{F}^m$ using basic mathematical operators. Given basic mathematical operators $\{\mathcal{A}_1,\cdots,\mathcal{A}_n\}$, we build a $k$-layer RNN for each $m = 0,1,2,\cdots$, by successively applying a weight matrix $W_{2,n}\in\mathbb{R}^{2\times n}$ to the operator vector $[\mathcal{A}_1,\cdots,\mathcal{A}_n]^T$ and then adding a bias vector $B_{2,1} = [b_1,b_2]^T\in\mathbb{R}^2$ times $\mathcal{I}$:

$$\begin{aligned} W_{2,n}[\mathcal{A}_1,\cdots,\mathcal{A}_n]^T + B_{2,1}\mathcal{I} &:= \begin{bmatrix} w_{1,1}\mathcal{A}_1 + w_{1,2}\mathcal{A}_2 + \cdots + w_{1,n}\mathcal{A}_n + b_1\mathcal{I} \\ w_{2,1}\mathcal{A}_1 + w_{2,2}\mathcal{A}_2 + \cdots + w_{2,n}\mathcal{A}_n + b_2\mathcal{I}\end{bmatrix} \\ &:= \begin{bmatrix}\mathcal{C}_1\\ \mathcal{C}_2\end{bmatrix}. \end{aligned} \tag{3.6}$$

Because $\mathcal{C}_1$ and $\mathcal{C}_2$ are operators, they can be applied to generate a more expressive formulation with a special "composition" denoted as $\odot$ defined below:

$$\begin{aligned} \mathcal{C}_1\odot\mathcal{C}_2 := {} & w_{1,1}w_{2,1}\mathcal{A}_1\circ\mathcal{A}_1 + \cdots + w_{1,1}w_{2,n}\mathcal{A}_1\circ\mathcal{A}_n + \cdots \\ & + w_{1,n}w_{2,1}\mathcal{A}_n\circ\mathcal{A}_1 + \cdots + w_{1,n}w_{2,n}\mathcal{A}_n\circ\mathcal{A}_n \\ & + (w_{1,1}b_2 + w_{2,1}b_1)\mathcal{A}_1 + \cdots + (w_{1,n}b_2 + w_{2,n}b_1)\mathcal{A}_n, \end{aligned} \tag{3.7}$$

where $\circ$ denotes the standard composition.

Now we define $\mathcal{F}^m$:

$$\begin{aligned} \boldsymbol{\xi}^{(1)} &:= W^{1,m}_{2,n}[\mathcal{A}_1,\mathcal{A}_2,\cdots,\mathcal{A}_n]^T + B^{1,m}_{2,1}\mathcal{I} \\ \mathcal{B}_1 &:= \mathcal{C}^{(1)}_1\odot\mathcal{C}^{(1)}_2 \\ \boldsymbol{\xi}^{(2)} &:= W^{2,m}_{2,n+1}[\mathcal{A}_1,\mathcal{A}_2,\cdots,\mathcal{A}_n,\mathcal{B}_1]^T + B^{2,m}_{2,1}\mathcal{I} \\ \mathcal{B}_2 &:= \mathcal{C}^{(2)}_1\odot\mathcal{C}^{(2)}_2 \\ \boldsymbol{\xi}^{(3)} &:= W^{3,m}_{2,n+2}[\mathcal{A}_1,\mathcal{A}_2,\cdots,\mathcal{A}_n,\mathcal{B}_1,\mathcal{B}_2]^T + B^{3,m}_{2,1}\mathcal{I} \\ \mathcal{B}_3 &:= \mathcal{C}^{(3)}_1\odot\mathcal{C}^{(3)}_2 \\ &\vdots \\ \boldsymbol{\xi}^{(K)} &:= W^{K,m}_{2,n+K-1}[\mathcal{A}_1,\mathcal{A}_2,\cdots,\mathcal{A}_n,\mathcal{B}_1,\cdots,\mathcal{B}_{K-1}]^T + B^{K,m}_{2,1}\mathcal{I} \\ \mathcal{B}_K &:= \mathcal{C}^{(K)}_1\odot\mathcal{C}^{(K)}_2 \\ \mathcal{F}^m &:= W^{K+1,m}_{1,n+K}[\mathcal{A}_1,\mathcal{A}_2,\cdots,\mathcal{A}_n,\mathcal{B}_1,\cdots,\mathcal{B}_K]^T, \end{aligned} \tag{3.8}$$

where the weight matrices are given by:

$$W^{k,m}_{2,n+k-1} := \begin{bmatrix} w^{k,m}_{1,1} & w^{k,m}_{1,2} & \cdots & w^{k,m}_{1,n+k-1} \\ w^{k,m}_{2,1} & w^{k,m}_{2,2} & \cdots & w^{k,m}_{2,n+k-1}\end{bmatrix} \tag{3.9}$$

for $k = 1,\cdots,K$ and,

$$W^{K+1,m}_{1,n+K} := \begin{bmatrix} w^{K+1,m}_1 & w^{K+1,m}_2 & \cdots & w^{K+1,m}_{n+K}\end{bmatrix}$$

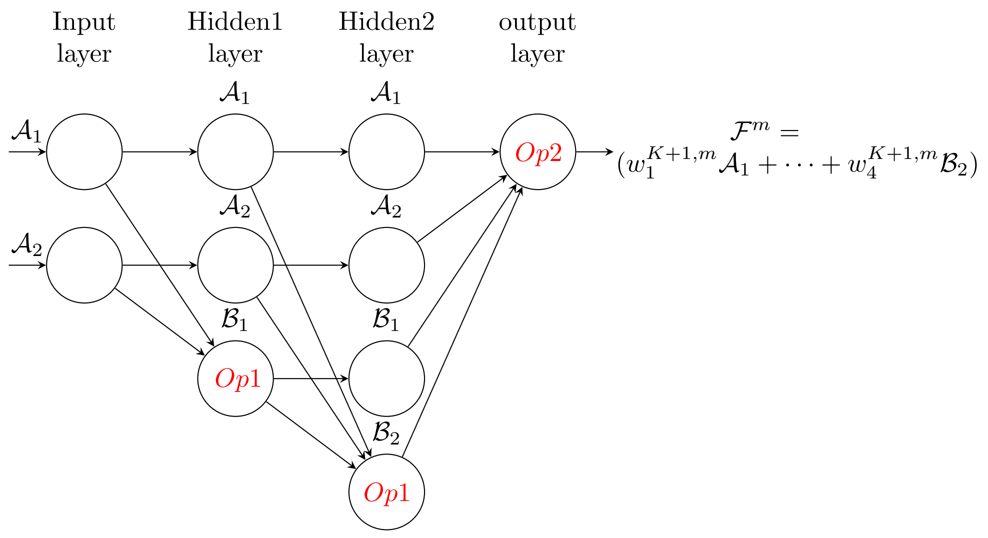

FIG. 1. *Example RNN with $K = 2$ hidden layers. The output $\mathcal{F}^m$ makes up the order $\varepsilon^m$ part of the right hand side of the PDE. Op1 is the mathematical operation in Equation (3.6) and 3.7 which takes linear combination plus bias of the previous layer and performing a composition. Op2 is the operation of forming a linear combination of the previous layer (the last line of Equation (3.8)).*

with each $w_{i,j}^{k,m} \in \mathbb{R}$. The biases are given by:

$$B_{2,1}^{k,m} := \begin{bmatrix} b_1^{k,m} \\ b_2^{k,m} \end{bmatrix}$$

with $b_j^{k,m} \in \mathbb{R}$.

The operator built by the recursive compositions in (3.8) is a symbolic RNN operator, the evaluation of which on a given function follows in the basic evaluation rules introduced in Section 3.1. will have to be evaluated at both data sets $\{g(v, x, t_i)\}$ and $\{\rho(x, t_i)\}$, since our model problem depends on both $g$ and $\rho$. A diagrammatic representation of this RNN is shown in Figure 1.

*Remark* 3.2. We have adopted the recursive framework introduced in [27] to build our RNN. The main difference between the RNN in this manuscript and the RNN in [27] is that our RNN can learn nonlocal and multiscale operators. Other RNN frameworks may also be good alternatives. Optimizing the RNN framework is not a focus in this paper.

*Remark* 3.3. The weights and biases can be trainable space-dependent functions such that our algorithm can learn more space-dependent operators, e.g., let

$$w_{i,j}^{k,m}(x) : \mathbb{R} \to \mathbb{R}, \quad \text{and} \quad b_j^{k,m}(x) : \mathbb{R} \to \mathbb{R}. \tag{3.10}$$

In more particular, one can also replace these weights and biases with neural networks in the spatial variable $x$ at the cost of using more parameters. We will let the reader explore these possibilities but, we will also present a yet different alternative to treating space-dependent weights and biases in the next section.

**3.4. An Example when $K = 1$.** Using $K = 1$ and two basic mathematical operators $\mathcal{A}_1$ and $\mathcal{A}_2$ in (3.8), for the PDE model in (3.4), we produce an RNN as a scalar PDE ansatz of the form:

$$\begin{aligned}
\partial_t u = &\sum_{m=0}^{M} \frac{1}{\varepsilon_{pred}^m} \Big( w_1^{2,m}\mathcal{A}_1 + w_2^{2,m}\mathcal{A}_2 + w_3^{2,m}\Big[ (w_{1,1}^{1,m}\mathcal{A}_1 + w_{1,2}^{1,m}\mathcal{A}_2 \\
&+ b_1^{1,m} I.d.) \circ (w_{2,1}^{1,m}\mathcal{A}_1 + w_{2,2}^{1,m}\mathcal{A}_2 + b_2^{1,m} I.d.)\Big]\Big)(u) \\
= &\sum_{m=0}^{M} \frac{1}{\varepsilon_{pred}^m} \Big[ w_3^{2,m} b_1^{1,m} b_2^{1,m} u + (w_1^{2,m} + w_3^{2,m}(w_{1,1}^{1,m} b_2^{1,m} + b_1^{1,m} w_{2,1}^{1,m}))\mathcal{A}_1(u) \\
&+ (w_2^{2,m} + w_3^{2,m}(w_{1,2}^{1,m} b_2^{1,m} + b_1^{1,m} w_{2,1}^{1,m}))\mathcal{A}_2(u) \\
&+ w_3^{2,m}(w_{1,1}^{1,m} w_{2,1}^{1,m}\mathcal{A}_1 \circ \mathcal{A}_1(u) + w_{1,1}^{1,m} w_{2,2}^{1,m}\mathcal{A}_1 \circ \mathcal{A}_2(u)) \\
&+ w_3^{2,m}(w_{1,2}^{1,m} w_{2,1}^{1,m}\mathcal{A}_2 \circ \mathcal{A}_1(u) + w_{1,2}^{1,m} w_{2,2}^{1,m}\mathcal{A}_2 \circ \mathcal{A}_2(u))\Big],
\end{aligned} \tag{3.11}$$

with $\varepsilon_{pred}$ given by Equation (5.1) or (5.2).

The weights and biases are determined by minimizing a loss function defined in the next section.

## 4. Loss Functions for Learning PDEs.

To deduce the weights and biases for our PDE ansatz, we need to minimize a loss function. We begin by describing an unregularized loss function for learning PDEs from data.

### 4.1. Unregularized Loss Function.

Let us first focus on the case of a single scalar equation ansatz in (3.4). We build an unregularized loss that will be a data-dependent function with the following abstract notation:

$$L(\boldsymbol{\theta}) = \frac{1}{N_t - q} \sum_{n=1}^{N_t - q} ||\mathcal{K}_u^n(D^{n,q}; \boldsymbol{\theta})||_* \tag{4.1}$$

where $\boldsymbol{\theta}$ denotes the set of all parameters in our RNN and

$$\mathcal{K}_u^n(D^{n,q}; \boldsymbol{\theta}) \tag{4.2}$$

relates $q+1$-tuple data points:

$$D^{n,q} := \{u(x,t_n), u(x,t_{n+1}), \cdots, u(x,t_{n+q})\}, \qquad n = 1, \cdots N_t - q.$$

The idea is that as $L \to 0$ with respect to a suitable norm $\|\cdot\|_*$, $\mathcal{F}$ approaches the correct PDE. Commonly used norms for loss minimization include $\ell_1$, $\ell_2$, and the Huber loss (see [27, 34]).

To be precise, the relation of $D^{n,q}$ is specified by a time-stepping scheme, e.g., the Implicit-Explicit Runge-Kutta scheme. However, to give the reader a greater understanding of $\mathcal{K}_u^n(D^{n,q}; \boldsymbol{\theta})$, we will start with simpler schemes here. The the symbolic RNN introduced in the previous section together with our IMEX schemes here will make up our Densely Connected Recurrent Neural Network (DC-RNN).

**Forward Euler scheme.** The forward Euler scheme only involves two time steps and, hence, $q = 1$. We can specify $\mathcal{K}_u^n(D^{n,1}; \boldsymbol{\theta})$ to relate the data pair

$$D^{n,1} = \{u(x,t_n), u(x,t_{n+1})\} \tag{4.3}$$

using a forward finite difference approximation for $\partial_t u = \mathcal{F}(u(x,t); \boldsymbol{\theta})$, the right hand side of which is a symbolic RNN as an equation ansatz. This gives us the forward Euler fitting scheme:

$$\mathcal{K}_u^n(D^{n,1}; \boldsymbol{\theta}) = u(x,t_{n+1}) - u(x,t_n) - \Delta t \cdot \mathcal{F}(u(x,t_n); \boldsymbol{\theta}). \tag{4.4}$$

Minimizing the loss in Equation (4.1) will determine a PDE governing the training data with time accuracy $\Delta t$. We display in Figure 2 a DC-RNN for determining the equation satisfied by $g(v,x,t)$ based on the Forward Euler scheme.

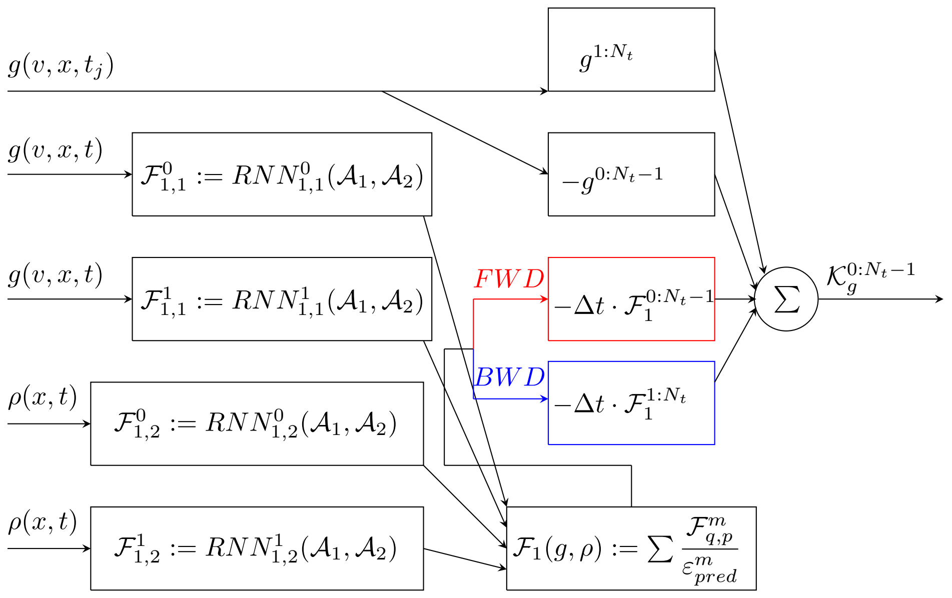
$g(v,x,t_j)$
$g^{1:N_t}$
$g(v,x,t)$
$\mathcal{F}^0_{1,1} := RNN^0_{1,1}(\mathcal{A}_1,\mathcal{A}_2)$
$-g^{0:N_t-1}$
$g(v,x,t)$
$\mathcal{F}^1_{1,1} := RNN^1_{1,1}(\mathcal{A}_1,\mathcal{A}_2)$
$FWD$
$-\Delta t\cdot\mathcal{F}_1^{0:N_t-1}$
$\sum$
$\mathcal{K}_g^{0:N_t-1}$
$BWD$
$-\Delta t\cdot\mathcal{F}_1^{1:N_t}$
$\rho(x,t)$
$\mathcal{F}^0_{1,2} := RNN^0_{1,2}(\mathcal{A}_1,\mathcal{A}_2)$
$\rho(x,t)$
$\mathcal{F}^1_{1,2} := RNN^1_{1,2}(\mathcal{A}_1,\mathcal{A}_2)$
$\mathcal{F}_1(g,\rho) := \sum \frac{\mathcal{F}^m_{q,p}}{\varepsilon^m_{pred}}$

FIG. 2. *Example DC-RNN for determining the g-Equation (2.5) based on Forward Euler (Red) and Backward Euler (Blue) schemes. The inputs are $\rho(t_n)$ and $g(t_n)$ for $n = 0,1,2,\cdots,N_t$. The dictionary contains order $\mathcal{O}(1)$ and $\mathcal{O}(\varepsilon)$ operators. These operators are generated by the RNNs corresponding to orders $\varepsilon^{-m}$ $m = 0,1$ using $\mathcal{A}_1$, $\mathcal{A}_2$. The output $\mathcal{K}_g^n$ $(n = 0,1,\cdots,N_t-1)$ is to be minimized with respect to a chosen norm.*

**Backward Euler scheme.** The Backward Euler scheme for $\partial_t u = \mathcal{F}(u(x,t);\boldsymbol{\theta})$ relates the data pair

$$D^{n,1} = \{u(x,t_n), u(x,t_{n+1})\} \tag{4.5}$$

using the backward Euler fitting scheme:

$$\mathcal{K}_u^n(D^{n,1};\boldsymbol{\theta}) = u(x,t_{n+1}) - u(x,t_n) - \Delta t\cdot\mathcal{F}(u(x,t_{n+1});\boldsymbol{\theta}). \tag{4.6}$$

Minimizing the loss in Equation (4.1) will determine a PDE governing the training data with accuracy $\Delta t$. We display in Figure 2 a DC-RNN for determining the equation satisfied by $g(v,x,t)$ based on the Backward-Euler scheme.

**Fourth-order Explicit Runge-Kutta.** Higher-order schemes like the K-stage Runge-Kutta scheme can also be used to relate $D^{n,1} = \{u(x,t_n), u(x,t_{n+1})\}$ for $\partial_t u = \mathcal{F}(u(x,t);\boldsymbol{\theta})$:

$$\begin{aligned}
\mathcal{K}_u^n(D^{n,1};\boldsymbol{\theta}) &= u(x,t_{n+1}) - u(x,t_n) - \frac{\Delta t}{6}\cdot\left(\mathcal{K}_1 + 2\mathcal{K}_2 + 2\mathcal{K}_3 + \mathcal{K}_4\right),\\
\mathcal{K}_1 &= \mathcal{F}(u(x,t_n);\boldsymbol{\theta}),\\
\mathcal{K}_2 &= \mathcal{F}(u(x,t_n + \frac{\Delta t}{2}) + \frac{\Delta t}{2}\mathcal{K}_1;\boldsymbol{\theta}),\\
\mathcal{K}_3 &= \mathcal{F}(u(x,t_n + \frac{\Delta t}{2}) + \frac{\Delta t}{2}\mathcal{K}_2;\boldsymbol{\theta}),\\
\mathcal{K}_4 &= \mathcal{F}(u(x,t_n + \Delta t) + \Delta t\mathcal{K}_3;\boldsymbol{\theta})
\end{aligned} \tag{4.7}$$

with $\mathcal{K}_l$ as the $l$-th stage. The Runge-Kutta schemes tend to be more computationally expensive as they require computation of the intermediate stages $\mathcal{K}_l$.

We only focused on the scalar equation in (3.4) to illustrate the loss function for the above schemes. The construction for the two-component vector equation in (3.5) is similar. The loss function is the sum of the loss function for each component

$$L(\boldsymbol{\theta}) = \frac{1}{N_t - q}\sum_{n=1}^{N_t-q} ||\mathcal{K}_g^n(D^{n,q};\boldsymbol{\theta})||_* + ||\mathcal{K}_\rho^n(D^{n,q};\boldsymbol{\theta})||_*, \tag{4.8}$$

where $\mathcal{K}_g^n(D^{n,q};\boldsymbol{\theta})$ and $\mathcal{K}_\rho^n(D^{n,q};\boldsymbol{\theta})$ relate the data in

$$D^{n,q} := \{g(x,t_n), g(x,t_{n+1}), \cdots, g(x,t_{n+q}), \rho(x,t_n), \rho(x,t_{n+1}), \cdots, \rho(x,t_{n+q})\}, \qquad n = 1, \cdots N_t - q.$$

In this paper, we are interested in Equations (2.4) and (2.5) and, hence, will use schemes specialized for them. Specifically, we consider the class of schemes belonging to the Implicit-Explicit Runge-Kutta (IMEX) methods as in [3]. First, we define the first-order IMEX scheme. Then, we use this scheme to specify $\mathcal{K}_g^n(D^{n,q};\boldsymbol{\theta})$ and $\mathcal{K}_\rho^n(D^{n,q};\boldsymbol{\theta})$ in the loss (4.8) for learning Equations (2.4) and (2.5) using DC-RNN.

**First-Order IMEX Runge-Kutta.** We introduce the first-order IMEX Runge-Kutta [22] for solving Equations (2.4) and (2.5). The first-order IMEX scheme is given by:

$$\begin{aligned} g_{i+1/2}^{n+1} = g_{i+1/2}^n + \Delta t \Bigg\{ &\frac{1}{\varepsilon}\left(I - \langle\ \rangle\right)\left(v^+ \frac{g_{i+1/2}^n - g_{i-1/2}^n}{\Delta x} + v^- \frac{g_{i+3/2}^n - g_{i+1/2}^n}{\Delta x}\right) \\ &- \frac{\sigma_{i+1/2}^S}{\varepsilon^2} g_{i+1/2}^{n+1} - \frac{1}{\varepsilon^2} v \frac{\rho_{i+1}^n - \rho_i^n}{\Delta x} - \sigma_{i+1/2}^A g_{i+1/2}^n \Bigg\}, \end{aligned} \tag{4.9}$$

$$\rho_i^{n+1} = \rho_i^n + \Delta t \left\{ \left\langle v \frac{g_{i+1/2}^{n+1} - g_{i-1/2}^{n+1}}{\Delta x} \right\rangle - \sigma_i^A \rho_i^n + G_i \right\}, \tag{4.10}$$

where $v^+ = \dfrac{v + |v|}{2}$ and $v^- = \dfrac{v - |v|}{2}$. From this, we see that Equation (4.9) gives a relationship among $D^{n,1}$ that can be generalized to the ansatz:

$$\begin{aligned} \mathcal{K}_g^n(D^{n,1};\boldsymbol{\theta}) = &(1 + \Delta t \frac{\sigma^S(x)}{\varepsilon^2}) g(v,x,t_{n+1}) - (1 - \Delta t \sigma^A(x)) g(v,x,t_n) \\ &- \Delta t \cdot \mathcal{F}_1(g(v,x,t_n), \rho(x,t_n);\boldsymbol{\theta}), \end{aligned} \tag{4.11}$$

where $\mathcal{F}_1$ is the operator ansatz introduced in (3.5).

Equation (4.10) gives a relationship between data in $D^{n,1}$ via:

$$\begin{aligned} \mathcal{K}_\rho^n(D^{n,1};\boldsymbol{\theta}) = &\rho(x,t_{n+1}) - (1 - \Delta t \sigma_A(x)) \rho(x,t_n) - \Delta t G(x) \\ &- \Delta t \cdot \mathcal{F}_2(g(v,x,t_{n+1}), \rho(x,t_n);\boldsymbol{\theta}). \end{aligned} \tag{4.12}$$

$\mathcal{F}_1$ and $\mathcal{F}_2$ will be learned by minimizing the loss function (4.8). In fact, one does not need to assume that the functions $\sigma^S(x)$, $\sigma^A(x)$, and $G(x)$ are known. One can learn these functions during the training process by replacing them with neural networks. For the special case when $\sigma^S(x)$ and $\sigma^A(x)$ are constants, we can replace them with trainable parameters $w_S$ and $w_A$, respectively.

We display in Figure 3 a DC-RNN for determining the equation satisfied by $g(v,x,t)$ based on the First order IMEX scheme. One can go higher order with higher-order IMEX schemes. These will either introduce more intermediate stages or relate more data points to each other by increasing $q$. For larger $q$, we will use the IMEX-BDF schemes. We leave details concerning higher-order schemes in the Appendix section.

## 4.2. Physics-aware loss function.

To improve physically accurate predictions, we propose to add a physics-based regularization term $R(\boldsymbol{\theta})$ to the unregularized loss function in (4.1) or (4.8). Let us take the example of (4.1) below:

$$L(\boldsymbol{\theta}) = \frac{1}{N_t - q} \sum_{n=1}^{N_t - q} ||\mathcal{K}_g^n(D;\boldsymbol{\theta})|| + R(\boldsymbol{\theta}). \tag{4.13}$$

We will discuss three kinds of regularization terms $R(\boldsymbol{\theta})$.

**Regularization via $\langle g \rangle = 0$.**

As seen in Equation (2.3), the true $g$-solution must satisfy the $\langle g \rangle = 0$ constraint. This can be incorporated by imposing

$$\langle \mathcal{F}_1(g,\rho) \rangle = 0, \tag{4.14}$$

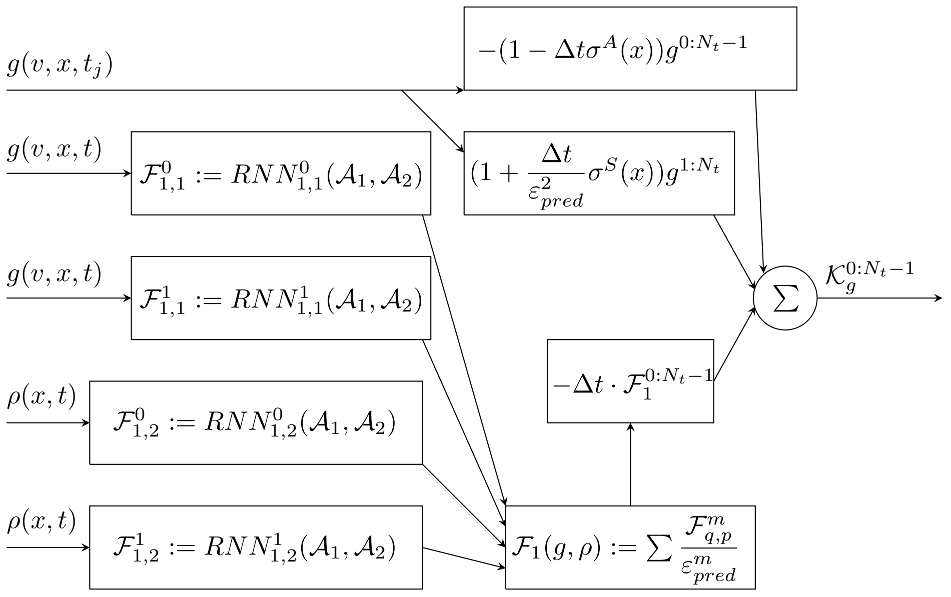
$g(v,x,t_j)$
$g(v,x,t)$
$g(v,x,t)$
$\rho(x,t)$
$\rho(x,t)$
$-(1-\Delta t\sigma^A(x))g^{0:N_t-1}$
$(1+\frac{\Delta t}{\varepsilon^2_{pred}}\sigma^S(x))g^{1:N_t}$
$\mathcal{F}^0_{1,1} := RNN^0_{1,1}(\mathcal{A}_1,\mathcal{A}_2)$
$\mathcal{F}^1_{1,1} := RNN^1_{1,1}(\mathcal{A}_1,\mathcal{A}_2)$
$\mathcal{F}^0_{1,2} := RNN^0_{1,2}(\mathcal{A}_1,\mathcal{A}_2)$
$\mathcal{F}^1_{1,2} := RNN^1_{1,2}(\mathcal{A}_1,\mathcal{A}_2)$
$-\Delta t\cdot\mathcal{F}_1^{0:N_t-1}$
$\mathcal{F}_1(g,\rho) := \sum \frac{\mathcal{F}^m_{q,p}}{\varepsilon^m_{pred}}$
$\sum$
$\mathcal{K}_g^{0:N_t-1}$

FIG. 3. *Example DC-RNN for determining the g-Equation (2.5) based on the First order IMEX scheme. The inputs are $\rho(t_n)$ and $g(t_n)$ for $n = 0,1,2,\cdots,N_t$. The dictionary contains order $\mathcal{O}(1)$ and $\mathcal{O}(\varepsilon)$ operators. These operators are generated by the RNNs corresponding to orders $\varepsilon^{-m}$ $m = 0,1$ using $\mathcal{A}_1$, $\mathcal{A}_2$. The output $\mathcal{K}^n_g$ $(n = 0,1,\cdots,N_t-1)$ is to be minimized with respect to a chosen norm.*

to the right hand side of the first component (g-equation) during training. The loss enforcing Equation (4.14) is:

$$L(\boldsymbol{\theta}) = \frac{1}{N_t-q}\sum_{n=1}^{N_t-q} ||\mathcal{K}^n_g(D;\boldsymbol{\theta})|| + ||\Delta t\cdot\langle\mathcal{F}_1(g,\rho;\boldsymbol{\theta})\rangle||. \tag{4.15}$$

The constraint in Equation (4.14) for the Forward-Euler case can be justified by performing the following calculation:

$$\begin{aligned}\langle g(v,x,t_{n+1})\rangle &\approx \langle g(v,x,t_n) + \Delta t\cdot\mathcal{F}_1\rangle \quad \text{(using Forward Euler)}\\ &= \Delta t\cdot\langle\mathcal{F}_1\rangle \qquad\qquad\qquad \text{(by linearity and } \langle g\rangle = 0).\end{aligned} \tag{4.16}$$

Thus, $\langle g\rangle = 0$ implies (4.14). The justification of (4.14) for other Runge-Kutta schemes is similar.

*Remark* 4.1. The factor $\Delta t$ is multiplied to $\mathcal{F}_1$ and remains in the loss (4.15) during training. $\Delta t$ is our Lagrange-multiplier.

**Regularization via Sparsity.** If the size of the dictionary is too large, one is more likely to over-fit data to an incorrect PDE. To help omit terms that do not appear in the PDE, we impose sparsity in weights and biases in the DC-RNN. Denote the set of all trainable parameters excluding $w_{eps}$, the parameter which trains $\varepsilon_{pred}$, by $\boldsymbol{\theta}$. The regularization term

$$R(\boldsymbol{\theta}) = \gamma\cdot||\boldsymbol{\theta}||_{l^1} \text{ with } \gamma\in\mathbb{R}^+ \tag{4.17}$$

is one natural choice to produce a PDE with the fewest possible terms. $\gamma$ is typically chosen to be a small number. In our numerical examples, $\gamma$ is chosen to be of order $10^{-4}$.

**Regularization via the Continuity of Weights and Biases.** We note that if $\sigma^S(x)$, $\sigma^A(x)$, $G(x)$, weights, or biases are not constant in $x$, we need to introduce neural networks to parametrize them to capture the dynamics of these functions of $x$. in this paper, each of these neural networks have $N_x$ trainable parameters. In this case the parameter set $\boldsymbol{\theta}$ of our RNN is a function in $x$. To promote the continuity in $x$, one choice is to add an extra regularization term $R(\boldsymbol{\theta})$ to the loss function of the form:

$$R(\boldsymbol{\theta}) = \gamma\cdot||\nabla_x\boldsymbol{\theta}(x)||_{l^1} \text{ with } \gamma\in\mathbb{R}^+. \tag{4.18}$$

Physically, Equation (4.18) is used to lessen the jump discontinuity in the learned functions in $x$.

## 5. Optimization.

In this section we discuss how we update parameters $(\varepsilon_{pred}, \boldsymbol{\theta})$ to reach a minimum for our loss function. Since we do not assume that $\varepsilon$ is known a priori, we will have to train this parameter. At the end of this section, we argue why our algorithm is expected to be superior to existing algorithms. Our reasoning suggests that terms of order $\mathcal{O}(\varepsilon^n)$ should be updated with a learning rate proportional to $\varepsilon^n$. We will verify this claim through several numerical experiments in the next section.

### 5.1. Training of $\varepsilon_{pred}$.

A major goal for our algorithm is to determine approximately the magnitude of the scale $\varepsilon$ involved in the multiscale dynamics. To fulfill the condition $0 < \varepsilon \leq 1$, the $\varepsilon_{pred}$ in our algorithm is set to

$$\varepsilon_{pred} = \frac{1}{2}(\tanh(w_\varepsilon) + 1), \tag{5.1}$$

where $w_\varepsilon$ is a trainable parameter. However, if one can parallelize, it makes more sense to restrict $\varepsilon_{pred}$ over several intervals spanning $(0, 1]$. For instance, let $s(i) = 0.1^i$ and

$$\varepsilon_{pred} = \frac{s(i) - s(i+1)}{2}(\tanh(w^i_\varepsilon) + \min_i), \tag{5.2}$$

where $(0, 1] = [s(1), s(2)] \cup [s(2), s(3)] \cup [s(3), s(4)] \cup \cdots$. After training over each interval, one can choose the PDE corresponding to the lowest loss. Thus, with Equation (5.2), one has better control over where local minimums of the loss function occur.

### 5.2. Training of parameters.

The parameters for the loss function (4.13), can be trained using our suggested algorithm: Adam method [21]. This algorithm is great at training a relatively large number of parameters efficiently. Other gradient descent methods are possible including stochastic gradient descent. We will discuss an implementation of stochastic gradient descent below using loss equation (4.13) with respect to the $L_1$ and $L_2$. While not necessary, we will simplify the calculations by using the forward Euler approximation and assuming $R(\theta)$ involves only sparse regularity. We can rewrite equation (4.13) as:

$$\widehat{L}(w_\varepsilon, \boldsymbol{\theta}, \boldsymbol{x}) = L(w_\varepsilon, \boldsymbol{\theta}, \boldsymbol{x}) + R_1 \tag{5.3}$$

where the regularization is

$$\begin{aligned} R_1 &:= \gamma_1 ||\boldsymbol{\theta}||_{L^1} \\ &= \gamma_1 \sum_i |\theta_i| \end{aligned} \tag{5.4}$$

and $L$ is given by

$$L(w_\varepsilon, \boldsymbol{\theta}, \boldsymbol{x}) = \frac{1}{N_t}\sum_{j=1}^{N_t} ||u(\boldsymbol{x}, t_{j+1}) - u(\boldsymbol{x}, t_j) + \int_{t_j}^{t_j+\Delta t} \sum_n \frac{1}{\varepsilon(w_\varepsilon)^n}\mathcal{F}^n(u(\boldsymbol{x}, s), \boldsymbol{\theta}_n)\, ds||_*. \tag{5.5}$$

Using the Forward Euler approximation,

$$\begin{aligned} L(w_\varepsilon, \boldsymbol{\theta}, \boldsymbol{x}) &= L_{Fwrd}(w_\varepsilon, \boldsymbol{\theta}, \boldsymbol{x}) \\ &= \frac{1}{N_t}\sum_{j=1}^{N_t} ||\mathcal{K}^j_u(u(\boldsymbol{x}, t_j), \boldsymbol{\theta})||_* \\ &= \frac{1}{N_t}\sum_{j=1}^{N_t} ||u(\boldsymbol{x}, t_{j+1}) - u(\boldsymbol{x}, t_j) + \Delta t \sum_n \frac{1}{\varepsilon(w_\varepsilon)^n}\mathcal{F}^n(u(\boldsymbol{x}, t_j), \boldsymbol{\theta}_n)||_* \end{aligned} \tag{5.6}$$

where $(w_\varepsilon, \boldsymbol{\theta}) := (w_\varepsilon, \boldsymbol{\theta}_0, \boldsymbol{\theta}_1, \cdots)$. The gradient of equation (5.3) with respect to $L_1$ and $L_2$ is given by

$$\begin{aligned} \nabla_{(w_\varepsilon, \boldsymbol{\theta})}\widehat{L} &= \frac{1}{N_t}\sum_{j=1}^{N_t} \text{sign}(L)(\partial_{w_\varepsilon} L, \Delta t \nabla_{\boldsymbol{\theta}_0}\mathcal{F}^0, \Delta t \frac{1}{\varepsilon(w_\varepsilon)}\nabla_{\boldsymbol{\theta}_1}\mathcal{F}^1, \cdots) + \gamma_1 \text{sign}(\boldsymbol{\theta}) \qquad \text{using } L^1 \\ \nabla_{(w_\varepsilon, \boldsymbol{\theta})}\widehat{L} &= \frac{1}{N_t}\sum_{j=1}^{N_t} 2L(\partial_{w_\varepsilon} L, \Delta t \nabla_{\boldsymbol{\theta}_0}\mathcal{F}^0, \Delta t \frac{1}{\varepsilon(w_\varepsilon)}\nabla_{\boldsymbol{\theta}_1}\mathcal{F}^1, \cdots) + \gamma_1 \text{sign}(\boldsymbol{\theta}) \qquad \text{using } L^2 \end{aligned} \tag{5.7}$$

If one records many data (large $N_t$), the summation in equation (5.7) can be slow to compute. Thus, one can reduce computational resources by using our suggested **stochastic gradient descent**: The sum is taken over a random smaller subset of $\{1, 2, \cdots, N_t\}$ of size $N_s < N_t$ and we replace $N_t$ with $N_s$ in equation (5.7).

**5.3. Discussion.** We will now discuss the effect of asymptotic expansion and our sparse regularization method. Because it is difficult to obtain clean algebraic expressions using "all of" $\widehat{L}$, we will consider a quadratic Taylor series truncation of $\mathcal{K} := \mathcal{K}_u^j(u(\boldsymbol{x}, t_j), \boldsymbol{\theta})$ near the point that minimizes $\widehat{L}$ which we denote by $(w_\varepsilon^*, \boldsymbol{\theta}_0^*, \boldsymbol{\theta}_1^*, \cdots)$. We will again assume the forward Euler approximation for $\mathcal{K}$. The point $(w_\varepsilon^{(0)}, \boldsymbol{\theta}_0^{(0)}, \boldsymbol{\theta}_1^{(0)}, \cdots)$ will denote the initial values for the training parameters and $(w_\varepsilon^{(k)}, \boldsymbol{\theta}_0^{(k)}, \boldsymbol{\theta}_1^{(k)}, \cdots)$ will denote the $k$-th step taken by the gradient descent process. The gradient at the $k$-th step is given by,

$$
\begin{aligned}
\boldsymbol{z} :=& \nabla_{(w_\varepsilon, \boldsymbol{\theta})} \mathcal{K}_u^j(w_\varepsilon^{(k)}, \boldsymbol{\theta}^{(k)}) \\
=& (\partial_{w_\varepsilon} \mathcal{K}_u^j, \Delta t \nabla_{\boldsymbol{\theta}_0} \mathcal{F}^0, \Delta t \frac{1}{\varepsilon(w_\varepsilon)} \nabla_{\boldsymbol{\theta}_1} \mathcal{F}^1, \cdots, \Delta t \frac{1}{\varepsilon(w_\varepsilon)^M} \nabla_{\boldsymbol{\theta}_M} \mathcal{F}^M)|_{(w_\varepsilon^{(k)}, \boldsymbol{\theta}_0^{(k)}, \boldsymbol{\theta}_1^{(k)}, \cdots, \boldsymbol{\theta}_M^{(k)})} \\
=& (\partial_{w_\varepsilon} \mathcal{K}_u^j, \widetilde{\boldsymbol{z}}) \\
=& (\partial_{w_\varepsilon} \mathcal{K}_u^j, \widetilde{\boldsymbol{z}}_0, \widetilde{\boldsymbol{z}}_1, \cdots, \widetilde{\boldsymbol{z}}_M)
\end{aligned}
\tag{5.8}
$$

where we defined

$$
\begin{aligned}
\widetilde{\boldsymbol{z}} &:= (\widetilde{\boldsymbol{z}}_0, \widetilde{\boldsymbol{z}}_1, \cdots, \widetilde{\boldsymbol{z}}_M) \\
&:= (\Delta t \nabla_{\boldsymbol{\theta}_0} \mathcal{F}^0, \Delta t \frac{1}{\varepsilon(w_\varepsilon)} \nabla_{\boldsymbol{\theta}_1} \mathcal{F}^1, \cdots, \Delta t \frac{1}{\varepsilon(w_\varepsilon)^M} \nabla_{\boldsymbol{\theta}_M} \mathcal{F}^M)
\end{aligned}
\tag{5.9}
$$

in order to simplify the notation. The hessian is given by

$$
\begin{aligned}
H :&= \nabla^2 \mathcal{K}_u^j(w_\varepsilon^{(k)}, \boldsymbol{\theta}^{(k)}) \\
&= \Delta t \begin{bmatrix}
\frac{1}{\Delta t} \partial^2_{w_\varepsilon} \mathcal{K}_u^j & [\mathbf{0}]_{1\times d} & -\frac{1}{\varepsilon^2} \partial_{w_\varepsilon} \varepsilon \nabla_{\boldsymbol{\theta}_1} \mathcal{F}^1 & -\frac{2}{\varepsilon^3} \partial_{w_\varepsilon} \varepsilon \nabla_{\boldsymbol{\theta}_2} \mathcal{F}^2 & \cdots \\
\nabla_{\boldsymbol{\theta}_0} (\partial_{w_\varepsilon} \mathcal{K}_u^j)^T & [\nabla^2_{\boldsymbol{\theta}_0} \mathcal{F}^0]_{d\times d} & [0]_{d\times d} & [0]_{d\times d} & \\
\nabla_{\boldsymbol{\theta}_1} (\partial_{w_\varepsilon} \mathcal{K}_u^j)^T & [0]_{d\times d} & [\frac{1}{\varepsilon} \nabla^2_{\boldsymbol{\theta}_1} \mathcal{F}^1]_{d\times d} & [0]_{d\times d} & \\
\nabla_{\boldsymbol{\theta}_2} (\partial_{w_\varepsilon} \mathcal{K}_u^j)^T & [0]_{d\times d} & [0]_{d\times d} & [\frac{1}{\varepsilon^2} \nabla^2_{\boldsymbol{\theta}_2} \mathcal{F}^2]_{d\times d} & \\
\vdots & & & & \ddots
\end{bmatrix}
\end{aligned}
\tag{5.10}
$$

where $d$ is the dimension of each $\boldsymbol{\theta}_n$. We also denote the $(d \cdot M - 1) \times (d \cdot M - 1)$ submatrix of $H$ by

$$
\widetilde{H} = \Delta t \begin{bmatrix}
[\nabla^2_{\boldsymbol{\theta}_0} \mathcal{F}^0]_{d\times d} & [0]_{d\times d} & [0]_{d\times d} & \\
[0]_{d\times d} & [\frac{1}{\varepsilon} \nabla^2_{\boldsymbol{\theta}_1} \mathcal{F}^1]_{d\times d} & [0]_{d\times d} & \\
[0]_{d\times d} & [0]_{d\times d} & [\frac{1}{\varepsilon^2} \nabla^2_{\boldsymbol{\theta}_2} \mathcal{F}^2]_{d\times d} & \\
\vdots & & & \ddots
\end{bmatrix}
\tag{5.11}
$$

**The effect of asymptotic expansion on learning rate.** We now consider the effect of updating the parameters $\boldsymbol{\theta}$ via gradient descent. To further simplify algebraic expressions, we will assume that $\varepsilon(w_\varepsilon^*) = \varepsilon^* = \varepsilon$ is the constant optimal value. According to the gradient descent method:

$$
\begin{aligned}
\boldsymbol{\theta}^{(k+1)} \leftarrow& \boldsymbol{\theta}^{(k)} - \boldsymbol{\alpha} \otimes \widetilde{\boldsymbol{z}} \\
:=& (\boldsymbol{\theta}^{(k)}) - (\alpha_0 \widetilde{\boldsymbol{z}}_0, \alpha_1 \widetilde{\boldsymbol{z}}_1, \cdots, \alpha_M \widetilde{\boldsymbol{z}}_M)
\end{aligned}
\tag{5.12}
$$

where the parameter $\boldsymbol{\alpha} := (\alpha_0, \alpha_1, \alpha_2, \cdots)$ will denote the learning rate which updates step $k \to k+1$. $\alpha_n$ will denote the learning rate for the terms of order $\mathcal{O}(\varepsilon^{-n})$. Our goal is to understand the optimal behaviour of the learning rates $\alpha_n$. Substituting equation (5.12) into our quadratic truncation of $\mathcal{K}_n^j$ yields:

$$\mathcal{K}_n^j(w_\varepsilon^*, \boldsymbol{\theta}^{(k+1)}) = \mathcal{K}_n^j(w_\varepsilon^*, \boldsymbol{\theta}^{(k)}) - (\boldsymbol{\alpha} \otimes \widetilde{\boldsymbol{z}})^T \widetilde{\boldsymbol{z}} + (\boldsymbol{\alpha} \otimes \widetilde{\boldsymbol{z}})^T \widetilde{H} (\boldsymbol{\alpha} \otimes \widetilde{\boldsymbol{z}}) \tag{5.13}$$

when $(\boldsymbol{\alpha} \otimes \widetilde{\boldsymbol{z}})^T H (\boldsymbol{\alpha} \otimes \widetilde{\boldsymbol{z}})$ is positive we can solve for the optimal values for $\boldsymbol{\alpha}$:

$$\boldsymbol{\alpha} = \frac{\widetilde{\boldsymbol{z}}^T \widetilde{\boldsymbol{z}}}{\widetilde{\boldsymbol{z}}^T \widetilde{H} \widetilde{\boldsymbol{z}}} \qquad \Longleftrightarrow \qquad \alpha_n = \frac{\varepsilon^{*n} \widetilde{\boldsymbol{z}}_n^T \widetilde{\boldsymbol{z}}_n}{\Delta t \widetilde{\boldsymbol{z}}_n^T [\nabla^2_{\boldsymbol{\theta}_n} \mathcal{F}^n]_{d\times d} \widetilde{\boldsymbol{z}}_n} \tag{5.14}$$

The meaning of the above calculations is summarize below:

**Observation.** If $\mathcal{F}^n$ is well approximated by a quadratic function with $\varepsilon(w_\varepsilon) = \epsilon^*$, then $\boldsymbol{\theta}_n^{(k+1)} \leftarrow \boldsymbol{\theta}_n^{(k)}$ should be updated (according to equations (5.13) and (5.14)) in the direction of $\nabla_{\theta_n} \mathcal{F}^n$. The optimal learning rate is proportional to $\frac{1}{\varepsilon^n}$. The eigenvalues and vectors of $[\nabla^2_{\boldsymbol{\theta}_n} \mathcal{F}^n]_{d\times d}$ determine the stability of the learning process. In the worst case scenario, $\widetilde{\boldsymbol{z}}_n$ is in the direction corresponding to the largest eigenvector of $[\nabla^2_{\boldsymbol{\theta}_n} \mathcal{F}^n]_{d\times d}$.

Example 5.1. *Because the observation above made use of several simplifying assumptions, we provide some numerical evidence to support this claim. What we observe through repeated numerical tests is that the multiscale fitting methods tend to converge to the correct model using less training time and iterations. Evidence of this is shown in figure 4.*

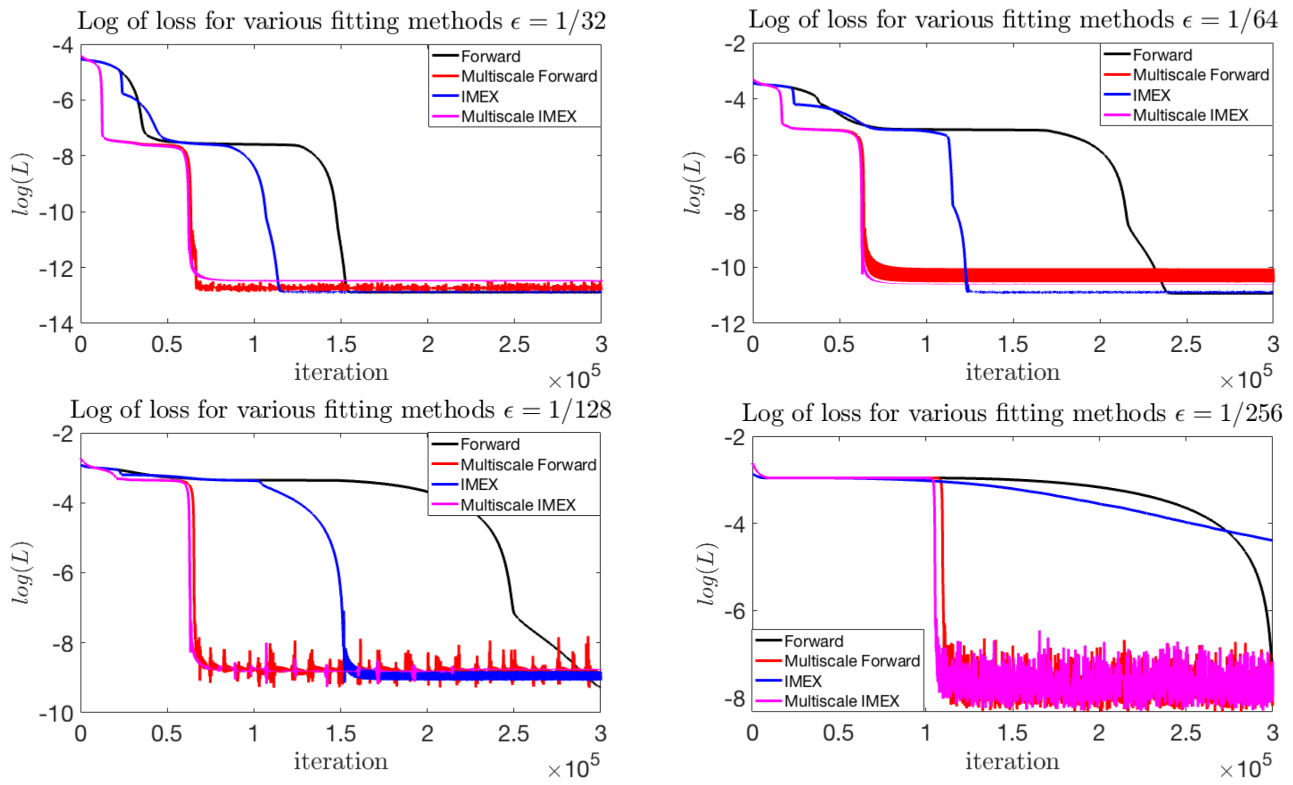

Fig. 4. *The behaviour of the loss function for example 6.1 is displayed above. We see quick convergence to the correct answer when using the multiscale fitting methods.*

*Remark* 5.1. We would like to remark that the efficiency of the proposed DC-RNN is demonstrated experimentally. As we made some simplifying assumptions on the terms $\mathcal{F}^n(\boldsymbol{\theta}_n)$ generated by our RNNs, we acknowledge that theoretical analysis remains vastly open, though several seminal works have been available [1, 2, 20, 45, 23].

*Remark* 5.2. other algorithms such as [26, 28, 34, 35, 32, 7, 42], do not have an adaptive $\varepsilon_{pred}$. Numerically, having an adaptive $\varepsilon_{pred}$ mimics adaptive gradient descent methods. The behavior of the loss functions vs iteration, has typically converged with respect to the number of iterations when compared to using the Adam algorithm with no multiscale expansion.

**The effect of sparse regularization.** The conventional Lasso method uses sparse regularization where one fits the data to an ansatz with an $L_1$ penalty as in equation (5.3). Typically the coefficients for the basis set of functions in the hypothesis space is set to be a sparse vector. Using our notation, this means

$$R_1 := \gamma_1 \sum_{m\geq 1} \sum_{\pi \in \mathcal{D}} |a_{\pi(1),\cdots,\pi(m)}(\boldsymbol{\theta})| \qquad (\text{for some } \gamma_1 \in \mathbb{R}^+) \tag{5.15}$$

is added to the loss function (1.1). However, in the case that some of these coefficients are large $\mathcal{O}(\varepsilon^{-n})$ for $n > 0$ and $0 < \varepsilon \ll 1$, equation (5.15) will create problems for the learning process. The issue is that there will be conflicting goals: keeping a particular set of coefficients $a_{\pi(1),\cdots,\pi(m)}(\boldsymbol{\theta})$ large in magnitude while at the same time minimizing $R_1$ as much as possible. Even if one sets $\gamma_1$ to be a very small value, one will still run into the trouble of setting appropriate learning rates as mentioned earlier. This conflict is clearly solved by our algorithm using the regularization

$$R_1 := \gamma_1 \sum_{m\geq 1} \sum_{\pi \in \mathcal{D}} |a_{\pi(1),\cdots,\pi(m)}(\boldsymbol{\theta}_0)| + |a_{\pi(1),\cdots,\pi(m)}(\boldsymbol{\theta}_1)| + \cdots + |a_{\pi(1),\cdots,\pi(m)}(\boldsymbol{\theta}_M)| \tag{5.16}$$

and setting the coefficients of the basis terms to

$$a_{\pi(1),\cdots,\pi(m)}(\boldsymbol{\theta}_0) + \frac{a_{\pi(1),\cdots,\pi(m)}(\boldsymbol{\theta}_1)}{\varepsilon} + \cdots + \frac{a_{\pi(1),\cdots,\pi(m)}(\boldsymbol{\theta}_M)}{\varepsilon^M}. \tag{5.17}$$

With this design we are able to have both sparsity and large $\mathcal{O}(\varepsilon^{-n})$ coefficients. The only drawback is that we had to introduce more parameters for our design.

## 6. Numerical Examples.

In this section, we test our DC-RNN using the PDE example in (2.4) and (2.5) with various values of $\varepsilon$. In the numeral results presented in this section, the predicted coefficients are of the form

$$\text{predicted} = (\text{exact} + \text{difference}), \tag{6.1}$$

where in red we highlight the difference from the predicted to the exact value. The smaller the magnitude of the difference, the better the prediction. We also include the percentage error:

$$\text{percentage error} = \frac{\sum |\text{exact coefficients} - \text{predicted coefficients}|}{\sum |\text{exact coefficients}|} \times 100\%. \tag{6.2}$$

The right hand side of the learned PDE will contain many terms, for the sake of readability, we display only the terms involved in either Equation (2.4) or (2.5). The terms that we don't present are typically minute in magnitude due to our sparsity regularization.

**Data Gathering.** The data that we produce in our examples are computed with IMEX-ARS(2,2,2) schemes using small mesh size $\Delta x = \frac{1}{1000}$ and $\Delta t = \frac{1}{2}\Delta x^2$. Thus, the data can be assumed to be nearly an exact solution to Equations (2.4) and (2.5). The CourantFriedrichsLewy (CFL) condition given in [22] for the first order IMEX scheme is given by

$$\Delta t \leq \left(\frac{3}{2}\Delta x^2 + \frac{\varepsilon \Delta x}{2}\right). \tag{6.3}$$

We note that the fitting method should also satisfy the appropriate CourantFriedrichsLewy conditions. For the Forward-Euler scheme, the stability depends on the stiffness of the PDE. Typically $\Delta t$ has to be quiet small for which we choose

$$\Delta t = \mathcal{O}(\Delta x^2) \tag{6.4}$$

to preform our Forward Euler fititng. The velocity distribution we use in our examples is the standard 16-point Gaussian quadrature set in $[-1, 1]$ as in [22].

The training data is prepared by taking a subset of the exact data to reduce the memory cost. We define the training set number of grid points by $\widetilde{N}_x$ and $\widetilde{N}_t$ and note that for all examples $\widetilde{N}_v = N_v = 16$. To obtain a subset of the data points a coarser grid is chosen: $\widetilde{\Delta x} \geq \Delta x$ and $\widetilde{\Delta t} \geq \Delta t$ with $\widetilde{\Delta x}$ and $\widetilde{\Delta t}$ satisfying Equation (6.3) and (6.4). Code will be made available at https://github.com/Ricard0000.

| $\varepsilon$ | Multiscale | learned $g$-equation using Foward Euler Scheme | Error |
|---|---|---|---|
| 1/ 16 | No | $\partial_t g = -(16^2 + 0.836\cdot 10^{-1})g - (16 + 2.052\cdot 10^{-1})v\cdot\partial_x g$ $+(16 + 2.066\cdot 10^{-1})\langle v\partial_x g\rangle - (16^2 - 1.145\cdot 10^{-1})v\cdot\partial_x\rho + \cdots$ | 0.11 % |
| 1/ 32 | No | $\partial_t g = -(32^2 - 1.855)g - (32 + 3.220\cdot 10^{-1})v\cdot\partial_x g$ $+(32 + 3.272\cdot 10^{-1})\langle v\partial_x g\rangle - (32^2 - 2.061)v\cdot\partial_x\rho + \cdots$ | 0.21% |
| 1/ 64 | No | $\partial_t g = -(64^2 - 3.302\cdot 10^{1})g - (64 + 1.389\cdot 10^{-1})v\cdot\partial_x g$ $+(64 + 1.321\cdot 10^{-1})\langle v\partial_x g\rangle - (64^2 - 3.317\cdot 10^{1})v\cdot\partial_x\rho + \cdots$ | 0.79 % |
| 1/ 128 | No | $\partial_t g = -(128^2 - 0.524\cdot 10^{3})g - (128 - 3.512)v\cdot\partial_x g$ $+(128 - 3.541)\langle v\partial_x g\rangle - (16384 - 0.524\cdot 10^{3})v\cdot\partial_x\rho + \cdots$ | 3.19 % |
| 1/ 256 | No | $\partial_t g = -(256^2 - 0.788\cdot 10^{4})g - (256 - 3.059\cdot 10^{1})v\cdot\partial_x g$ $+(256 - 3.066\cdot 10^{1})\langle v\partial_x g\rangle - (256^2 - 0.788\cdot 10^{4})v\cdot\partial_x\rho + \cdots$ | 12.03 % |
| 1/ 512 | No | $\partial_t g = -(512^2 + -2.348\cdot 10^{5})g - (512 - 4.590\cdot 10^{2})v\cdot\partial_x g$ $+(512 - 4.592\cdot 10^{2})\langle v\partial_x g\rangle - (512^2 + 2.348\cdot 10^{5})v\cdot\partial_x\rho + \cdots$ | 89.59 % |

Table 1
*Learned g-equation using the DC-RNN algorithm based on Forward-Euler schemes.*

| $\varepsilon$ | Multiscale | learned $g$-equation using IMEX1 Scheme | Error |
|---|---|---|---|
| 1/ 16 | No | $\partial_t g = -(16^2 + 3.344\cdot 10^{-1})g - (16 + 2.210\cdot 10^{-1})v\cdot\partial_x g$ $+(16 + 2.227\cdot 10^{-1})\langle v\partial_x g\rangle - (16^2 + 1.542\cdot 10^{-1})v\cdot\partial_x\rho + \cdots$ | 0.17 % |
| 1/ 32 | No | $\partial_t g = -(32^2 - 1.319)g - (32 + 3.478\cdot 10^{-1})v\cdot\partial_x g$ $+(32 + 3.539\cdot 10^{-1})\langle v\partial_x g\rangle - (32^2 - 1.538)v\cdot\partial_x\rho + \cdots$ | 0.16 % |
| 1/ 64 | No | $\partial_t g = -(64^2 + 2.291\cdot 10^{1})g - (64 + 1.000)v\cdot\partial_x g$ $+(64 + 1.004)\langle v\partial_x g\rangle - (64^2 + 2.224\cdot 10^{1})v\cdot\partial_x\rho + \cdots$ | 0.56 % |
| 1/ 128 | No | $\partial_t g = -(128^2 + 4.745\cdot 10^{2})g - (128 + 4.348)v\cdot\partial_x g$ $+(128 + 4.335)\langle v\partial_x g\rangle - (128^2 + 4.755\cdot 10^{2})v\cdot\partial_x\rho + \cdots$ | 2.90 % |
| 1/ 256 | No | $\partial_t g = -(256^2 + 2.915\cdot 10^{3})g - (256 + 1.1598\cdot 10^{1})v\cdot\partial_x g$ $+(256 + 1.156\cdot 10^{1})\langle v\partial_x g\rangle - (256^2 - 2.931\cdot 10^{3})v\cdot\partial_x\rho + \cdots$ | 4.46 % |
| 1/ 512 | No | $\partial_t g = -(512^2 - 2.575\cdot 10^{5})g - (512 - 0.503\cdot 10^{3})v\cdot\partial_x g$ $+(512 - 0.503\cdot 10^{3})\langle v\partial_x g\rangle - (512^2 - 2.575\cdot 10^{5})v\cdot\partial_x\rho + \cdots$ | 98.25 % |

Table 2
*Learned g-equation using the DC-RNN algorithm based on IMEX1 schemes.*

Example 6.1. ***Forward Euler vs IMEX: Multiscale vs Non-multiscale.*** *In this example, we show that when it comes to multiscale data, choosing a low-order Forward Euler scheme or the First-order IMEX time-stepping scheme is not enough to obtain an accurate prediction to data dynamics. It is crucial to assume a correct multiscale ansatz as in Equation (3.4). Evidence of this is provided in Tables 1,2 3, and 4. It is clear that after assuming a multiscale ansatz, a more accurate prediction is obtained. For this numerical example, we ran our algorithm using the ansatz produced by our RNN (3.8) using one layer. For non-multiscale methods we run our algorithm using $M = 0$ in Equation (3.4). For the multiscale method, we use $M = 2$. The data was produced with $\sigma^s(x) = 1$, $\sigma^A(x) = 0$. We choose $\widetilde{N}_x = 1000$ and $\widetilde{N}_t = 56$. We only attempt to learn the dynamics of the g-Equation (2.5).*

Example 6.2. ***Higher Order Methods: Multiscale vs Non-multiscale.*** *In this example, we again show that if the dynamics of the data is multiscale, then it is crucial that the ansatz should also be multiscale. We show that it is not enough to choose a higher-order (second order in time) IMEX scheme to obtain a good fitting to the data. In this experiment, we again set $\sigma^S(x) = 1$ and $\sigma^A(x) = 0$ and use two layers in our RNN. Tests are done using the IMEX fitting schmes ARS(2,2,2), BDF-2 (see Appendix), and different values for $\varepsilon$. Results are recorded in Tables 5, 6,7, and 8 for multiscale ($M = 2$) and non-multiscale ($M = 0$) ansatz. For this example we use $\widetilde{N}_x = 1000$ and $\widetilde{N}_t = 56$, thus we are using the same data as in the the Forward Euler and IMEX methods of the previous example. It is clear that the ARS(2,2,2) method with the multiscale assumption out-preforms the IMEX-BDF-2 and the first order methods from the previous example. We note that the results can be further improved by using a greater $\widetilde{N}_t$.*

| $\varepsilon$ | Multiscale | learned $g$-equation using Foward Euler Scheme | Error |
|---|---|---|---|
| 1/ 16 | Yes | $\partial_t g = -(16^2 - 3.616 \cdot 10^{-2})g - (16 + 2.161 \cdot 10^{-1})v \cdot \partial_x g$ $+(16 + 2.116 \cdot 10^{-1})\langle v\partial_x g\rangle - (16^2 - 0.866 \cdot 10^{-1})v \cdot \partial_x \rho + \cdots$ | 0.10 % |
| 1/ 32 | Yes | $\partial_t g = -(32^2 - 2.113)g - (32 + 3.833 \cdot 10^{-1})v \cdot \partial_x g$ $+(32 + 3.623 \cdot 10^{-1})\langle v\partial_x g\rangle - (32^2 - 1.882)v \cdot \partial_x \rho + \cdots$ | 0.22 % |
| 1/ 64 | Yes | $\partial_t g = -(64^2 - 3.325 \cdot 10^{1})g - (64 + 1.774 \cdot 10^{-1})v \cdot \partial_x g$ $+(64 + 1.887 \cdot 10^{-1})\langle v\partial_x g\rangle - (64^2 - 3.414 \cdot 10^{1})v \cdot \partial_x \rho + \cdots$ | 0.81 % |
| 1/ 128 | Yes | $\partial_t g = -(128^2 - 0.522 \cdot 10^{3})g - (128 - 3.225)v \cdot \partial_x g$ $+(128 - 3.259)\langle v\partial_x g\rangle - (128^2 - 0.526 \cdot 10^{3})v \cdot \partial_x \rho + \cdots$ | 3.19 % |
| 1/ 256 | Yes | $\partial_t g = -(256^2 + 0.787 \cdot 10^{4})g - (256 - 2.884 \cdot 10^{1})v \cdot \partial_x g$ $+(256 - 3.067 \cdot 10^{1})\langle v\partial_x g\rangle - (256^2 + 0.787 \cdot 10^{4})v \cdot \partial_x \rho + \cdots$ | 12.01 % |
| 1/ 512 | Yes | $\partial_t g = -(512^2 - 0.996 \cdot 10^{5})g - (512 - 1.936 \cdot 10^{2})v \cdot \partial_x g$ $+(512 - 1.946 \cdot 10^{2})\langle v\partial_x g\rangle - (512^2 - 0.997 \cdot 10^{5})v \cdot \partial_x \rho + \cdots$ | 38.03 % |

Table 3
*Learned g-equation using the DC-RNN algorithm based on Forward-Euler schemes.*

| $\varepsilon$ | Multiscale | learned $g$-equation using IMEX1 Scheme | Error |
|---|---|---|---|
| 1/ 16 | Yes | $\partial_t g = -(16^2 - -0.617 \cdot 10^{-2})g - (16 + 2.033 \cdot 10^{-1})v \cdot \partial_x g$ $+(16 + 2.094 \cdot 10^{-1})\langle v\partial_x g\rangle - (16^2 - 1.282 \cdot 10^{-1})v \cdot \partial_x \rho + \cdots$ | 0.10 % |
| 1/ 32 | Yes | $\partial_t g = -(32^2 - 1.863)g - (32 + 3.303 \cdot 10^{-1})v \cdot \partial_x g$ $+(32 + 3.466 \cdot 10^{-1})\langle v\partial_x g\rangle - (32^2 - 1.826)v \cdot \partial_x \rho + \cdots$ | 0.20 % |
| 1/ 64 | Yes | $\partial_t g = -(64^2 - 2.672 \cdot 10^{1})g - (64 + 2.679 \cdot 10^{-1})v \cdot \partial_x g$ $+(64 + 2.887 \cdot 10^{-1})\langle v\partial_x g\rangle - (64^2 - 2.758 \cdot 10^{1})v \cdot \partial_x \rho + \cdots$ | 0.65 % |
| 1/ 128 | Yes | $\partial_t g = -(128^2 - 2.488 \cdot 10^{2})g - (128 - 1.139)v \cdot \partial_x g$ $+(128 - 1.011)\langle v\partial_x g\rangle - (128^2 - 2.523 \cdot 10^{2})v \cdot \partial_x \rho + \cdots$ | 1.52 % |
| 1/ 256 | Yes | $\partial_t g = -(256^2 - 0.586 \cdot 10^{4})g - (256 - 2.537 \cdot 10^{1})v \cdot \partial_x g$ $+(256 - 2.156 \cdot 10^{1})\langle v\partial_x g\rangle - (256^2 - 0.586 \cdot 10^{4})v \cdot \partial_x \rho + \cdots$ | 8.94 % |
| 1/ 512 | Yes | $\partial_t g = -(512^2 - 3.472 \cdot 10^{4})g - (512 - 0.626 \cdot 10^{2})v \cdot \partial_x g$ $+(512 - 0.692 \cdot 10^{2})\langle v\partial_x g\rangle - (512^2 - 3.472 \cdot 10^{4})v \cdot \partial_x \rho + \cdots$ | 13.24 % |

Table 4
*Learned g-equation using the DC-RNN algorithm based on IMEX1 schemes.*

Example 6.3. ***Regularity Assumptions.*** *When the number of layers is large (large dictionary), or when data are lacking or noisy, over-fitting becomes an issue. We described a few physics-aware regularization terms in Section 3. We summarize these conditions below:*

1. *Regularization via sparsity;*
2. *Regularization via $\langle g\rangle = 0$;*
3. *Regularization via the continuity of weights and biases.*

*We numerically verify that further improvements can be made by applying these physics-aware regularization terms. For data that does not satisfy a smooth PDE, the regularization via the continuity of weights and biases might not be necessary. We record our results concerning the other two regularization methods in Tables 9 and 10. We explore the regularization via the continuity of weights and biases (Equation (4.18)) in the next example.*

Example 6.4. ***Learning Space-Dependent Functions.*** *We demonstrate that functions such as $\sigma^S(x)$, $\sigma^A(x)$, or $G(x)$ can be learned using space-dependent weights and biases. In this example, we choose*

$$\sigma^S(x) = 1 + 100x^2, \tag{6.5}$$

*$\sigma^A(x) = 0$, and $G(x) = 0$. We use $\varepsilon = 1$ and use our DC-RNN based on the IMEX-BDF-2 fitting. The predicted PDE for the g-equation is:*

$$\begin{aligned} \partial_t g &= (1 - 0.011)v\partial_x g - (1 - 0.016)\langle v\partial_x g\rangle \\ &\quad + (1 + 0.005)v\partial_x \rho + [1 + 0.100, 100 - 3.757]g + \cdots, \end{aligned} \tag{6.6}$$

| Scheme | Multiscale | learned $g$-equation | Error |
|---|---|---|---|
| 1/ 16 | No | $\partial_t g = -(16^2 + 3.073 \cdot 10^{-1})g - (16 - 0.531 \cdot 10^{-1})v \cdot \partial_x g$ $+(16 - 0.715 \cdot 10^{-3})\langle v\partial_x g\rangle - (16^2 + 1.354 \cdot 10^{-2})v \cdot \partial_x \rho + \cdots$ | 0.06 % |
| 1/ 32 | No | $\partial_t g = -(32^2 + 1.736 \cdot 10^{-1})g - (32 + 0.615 \cdot 10^{-1})v \cdot \partial_x g$ $+(32 + 1.243 \cdot 10^{-2})\langle v\partial_x g\rangle - (32^2 + 2.441 \cdot 10^{-2})v \cdot \partial_x \rho + \cdots$ | 0.01 % |
| 1/ 64 | No | $\partial_t g = -(64^2 + 0.784 \cdot 10^{0})g - (64 - 0.777 \cdot 10^{-2})v \cdot \partial_x g$ $+(64 - 1.195 \cdot 10^{-2})\langle v\partial_x g\rangle - (64^2 - 2.922 \cdot 10^{-1})v \cdot \partial_x \rho + \cdots$ | 0.01 % |
| 1/ 128 | No | $\partial_t g = -(128^2 - 2.857 \cdot 10^{1})g - (128 - 0.953 \cdot 10^{-1})v \cdot \partial_x g$ $+(128 - 2.176 \cdot 10^{-1})\langle v\partial_x g\rangle - (128^2 - 2.392 \cdot 10^{1})v \cdot \partial_x \rho + \cdots$ | 0.15 % |
| 1/ 256 | No | $\partial_t g = -(256^2 - 2.105 \cdot 10^{4})g - (256 - 2.411 \cdot 10^{2})v \cdot \partial_x g$ $+(256 - 0.821 \cdot 10^{2})\langle v\partial_x g\rangle - (256^2 - 2.102 \cdot 10^{4})v \cdot \partial_x \rho + \cdots$ | 32.22 % |
| 1/ 512 | No | $\partial_t g = -(512^2 - 2.621 \cdot 10^{5})g - (512 - 0.511 \cdot 10^{3})v \cdot \partial_x g$ $+(512 - 0.511 \cdot 10^{3})\langle v\partial_x g\rangle - (512^2 - 2.621 \cdot 10^{5})v \cdot \partial_x \rho + \cdots$ | 99.99 % |

Table 5
*Learned $g$-equation using IMEX-BDF-2 scheme assuming no dependence on $\varepsilon$ (Non-multiscale).*

| $\varepsilon$ | Multiscale | learned $g$-equation | Error |
|---|---|---|---|
| 1/ 16 | No | $\partial_t g = -(16^2 + 1.707 \cdot 10^{-1})g - (16 + 2.886 \cdot 10^{-2})v \cdot \partial_x g$ $+(16 + 3.391 \cdot 10^{-3})\langle v\partial_x g\rangle - (16^2 - 1.251 \cdot 10^{-3})v \cdot \partial_x \rho + \cdots$ | 0.03 % |
| 1/ 32 | No | $\partial_t g = -(32^2 + 3.706 \cdot 10^{-1})g - (32 - 0.822 \cdot 10^{-2})v \cdot \partial_x g$ $+(32 - 4.989 \cdot 10^{-3})\langle v\partial_x g\rangle - (32^2 + 3.662 \cdot 10^{-2})v \cdot \partial_x \rho + \cdots$ | 0.01 % |
| 1/ 64 | No | $\partial_t g = -(64^2 + 0.842 \cdot 10^{3})g - (64 - 3.800 \cdot 10^{1})v \cdot \partial_x g$ $+(64 - 3.819 \cdot 10^{1})\langle v\partial_x g\rangle - (64^2 - 4.054 \cdot 10^{3})v \cdot \partial_x \rho + \cdots$ | 59.76 % |
| 1/ 128 | No | $\partial_t g = -(128^2 - 1.424 \cdot 10^{4})g - (128 - 1.044 \cdot 10^{2})v \cdot \partial_x g$ $+(128 - 1.049 \cdot 10^{2})\langle v\partial_x g\rangle - (128^2 - 1.634 \cdot 10^{4})v \cdot \partial_x \rho + \cdots$ | 93.243 % |
| 1/ 256 | No | $\partial_t g = -(256^2 - 0.655 \cdot 10^{5})g - (256 - 2.551 \cdot 10^{2})v \cdot \partial_x g$ $+(256 - 2.550 \cdot 10^{2})\langle v\partial_x g\rangle - (256^2 - 0.654 \cdot 10^{5})v \cdot \partial_x \rho + \cdots$ | 99.96 % |
| 1/ 512 | No | $\partial_t g = -(512^2 - 2.621 \cdot 10^{5})g - (512 - 0.511 \cdot 10^{3})v \cdot \partial_x g$ $+(512 - 0.511 \cdot 10^{3})\langle v\partial_x g\rangle - (512^2 - 2.621 \cdot 10^{5})v \cdot \partial_x \rho + \cdots$ | 99.98 % |

Table 6
*Learned $g$-equation using IMEX-ARS(2,2,2) scheme assuming no dependence on $\varepsilon$ (Non-multiscale).*

*where $[1 + 0.100, 100 - 3.757]$ is the minimum and maximum values of $\sigma^S(x)$. We display the predicted $\sigma^S$ on the left of Figure 5. We also impose continuity of $\sigma^S(x)$ to our loss function. Our predicted PDE with continuity is given by:*

$$\begin{aligned}\partial_t g = (1 - 0.008)v\partial_x g - (1 - 0.017)\langle v\partial_x g\rangle \\ + (1 + 0.001)v\partial_x \rho + [1 + 0.060, 100 - 3.893]g + \cdots ,\end{aligned} \tag{6.7}$$

*with predicted $\sigma^S$ plotted on the right of Figure 5. We note that the jump discontinuities the left of Figure 5 is due to over fitting of the data. As we can see from this example, utilizing the continuity condition (4.18) these jumps are removed.*

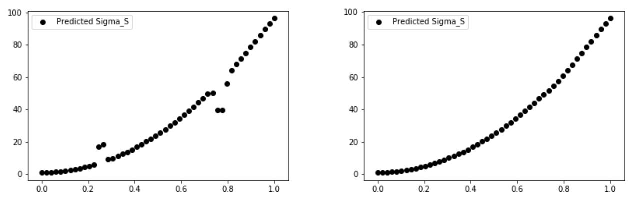

Fig. 5. *Left: Predicted $\sigma^S$ with no continuity constraints. Right: Predicted $\sigma^S$ with continuity constraints.*

| $\varepsilon$ | Multiscale | learned $g$-equation | Error |
|---|---|---|---|
| 1/ 16 | YES | $\partial_t g = -(16^2 - 0.782 \cdot 10^0)g - (16 - 1.056 \cdot 10^{-1})v \cdot \partial_x g$ <br> $+(16 - 0.737 \cdot 10^{-3})\langle v\partial_x g\rangle - (16^2 + 3.311 \cdot 10^{-2})v \cdot \partial_x \rho + \cdots$ | 0.16 % |
| 1/ 32 | YES | $\partial_t g = -(32^2 + 1.307)g - (32 + 4.107 \cdot 10^{-1})v \cdot \partial_x g$ <br> $+(32 + 1.753 \cdot 10^{-2})\langle v\partial_x g\rangle - (32^2 + 1.289 \cdot 10^{-2})v \cdot \partial_x \rho + \cdots$ | 0.08 % |
| 1/ 64 | YES | $\partial_t g = -(64^2 + 1.593)g - (64 + 0.576 \cdot 10^{-1})v \cdot \partial_x g$ <br> $+(64 + 3.793 \cdot 10^{-2})\langle v\partial_x g\rangle - (64^2 + 1.709)v \cdot \partial_x \rho + \cdots$ | 0.04% |
| 1/ 128 | YES | $\partial_t g = -(128^2 + 4.817 \cdot 10^1)g - (128 - 0.573 \cdot 10^{-1})v \cdot \partial_x g$ <br> $+(128 + 1.349 \cdot 10^{-1})\langle v\partial_x g\rangle - (128^2 + 0.506 \cdot 10^2)v \cdot \partial_x \rho + \cdots$ | 0.29 % |
| 1/ 256 | YES | $\partial_t g = -(256^2 + 4.745 \cdot 10^2)g - (256 + 1.118 \cdot 10^1)v \cdot \partial_x g$ <br> $+(256 + 1.851)\langle v\partial_x g\rangle - (256^2 + 4.688 \cdot 10^2)v \cdot \partial_x \rho + \cdots$ | 0.72 % |
| 1/ 512 | YES | $\partial_t g = -(512^2 - 2.602 \cdot 10^5)g - (512 - 0.508 \cdot 10^3)v \cdot \partial_x g$ <br> $+(512 - 0.508 \cdot 10^3)\langle v\partial_x g\rangle - (512^2 - 2.602 \cdot 10^5)v \cdot \partial_x \rho + \cdots$ | 99.27 % |

Table 7
*Learned $g$-equation using IMEX-BDF-2 scheme assuming dependence on $\varepsilon$ (Multiscale).*

| $\varepsilon$ | Multiscale | learned $g$-equation | Error |
|---|---|---|---|
| 1/ 16 | YES | $\partial_t g = -(16^2 - 2.177 \cdot 10^0)g - (16 + 3.167 \cdot 10^{-2})v \cdot \partial_x g$ <br> $+(16 - 1.269 \cdot 10^{-2})\langle v\partial_x g\rangle - (16^2 + 1.997 \cdot 10^{-2})v \cdot \partial_x \rho + \cdots$ | 0.41 % |
| 1/ 32 | YES | $\partial_t g = -(32^2 - 1.251 \cdot 10^0)g - (32 - 2.309 \cdot 10^{-2})v \cdot \partial_x g$ <br> $+(32 + 2.640 \cdot 10^{-3})\langle v\partial_x g\rangle - (32^2 - 4.427 \cdot 10^{-1})v \cdot \partial_x \rho + \cdots$ | 0.08 % |
| 1/ 64 | YES | $\partial_t g = -(64^2 - 3.075 \cdot 10^{-2})g - (64 - 0.866 \cdot 10^{-1})v \cdot \partial_x g$ <br> $+(64 - 1.748 \cdot 10^{-1})\langle v\partial_x g\rangle - (64^2 + 1.177)v \cdot \partial_x \rho + \cdots$ | 0.01 % |
| 1/ 128 | YES | $\partial_t g = -(128^2 + 1.018 \cdot 10^1)g - (128 + 1.068 \cdot 10^0)v \cdot \partial_x g$ <br> $+(128 + 2.269 \cdot 10^{-1})\langle v\partial_x g\rangle - (128^2 + 1.044 \cdot 10^1)v \cdot \partial_x \rho + \cdots$ | 0.06 % |
| 1/ 256 | YES | $\partial_t g = -(256^2 - 1.093 \cdot 10^3)g - (256 - 3.649 \cdot 10^{-1})v \cdot \partial_x g$ <br> $+(256 - 0.812)\langle v\partial_x g\rangle - (256^2 - 4.311 \cdot 10^1)v \cdot \partial_x \rho + \cdots$ | 0.86 % |
| 1/ 512 | YES | $\partial_t g = -(512^2 - -3.861 \cdot 10^4)g - (512 + 1.296 \cdot 10^3)v \cdot \partial_x g$ <br> $+(512 - 0.518 \cdot 10^2)\langle v\partial_x g\rangle - (512^2 - 1.102 \cdot 10^4)v \cdot \partial_x \rho + \cdots$ | 9.70 % |

Table 8
*Learned $g$-equation using ARS(2,2,2) scheme assuming dependence on $\varepsilon$ (Multiscale).*

Example 6.5. ***Higher-Order Methods*** *We test the performance of our algorithm using second-order and fourth-order time schemes. As expected, higher-order methods produce more accurate results as shown by Table 11.*

Example 6.6. ***Comparison with Conventional Methods. (Part 1)***
*The Lasso method [40, 15] is a popular tool for determining features involved in the dynamics of the data. This method does not assume a Chapman-Enskog like expansion as in (3.4). Since the dictionary has to be recorded in a matrix, the memory requirements for using the Lasso method are typically larger compared to our algorithm. We perform tests of our algorithm vs Lasso using $\sigma^S(x) = 1$, $\sigma^A(x) = 0$, and $G(x) = 0$. We record results for the $g$-equation in Table 12. Our ansatz assumed 18 terms, those involved in the dynamics: $g$, $v\partial_x\rho$, $v\partial_x g$, $\langle v\partial_x g\rangle$ and 14 others not involved (built by compositions of advection and projection operators (3.2) and (3.3)). We ran the Lasso method several times using several values of the regularization parameter $\alpha$. However, we only present the results associated with the best $\alpha$. The Lasso method performed fairly well but, it typically predicted more undesirable features for the dynamics and thus had a greater error.*

Example 6.7. ***Comparison with Conventional Methods. (Part 2)***
*Next, we try the STRidge method in [36]. Similar to the Lasso method, a matrix of the dictionary is formed. Unlike the Lasso method, [36] makes more efficient use of memory requirements and also features a hard threshold, i.e., large coefficients are assumed to be likely candidates for the dynamics of the PDE. Again, we use 18 terms for our dictionary as in the previous example. After running the STRidge algorithm, the predicted weights for the involved terms $g$, $v\partial_x\rho$, $v\partial_x g$, $\langle v\partial_x g\rangle$, were accurate. However, the STRidge*

| MULTISCALE | SPARSE | LEARNED $g$-EQUATION |
|---|---|---|
| NO | NO | $\partial_t g = -(256^2 - [65457.219])g - (256 - [253.605])v \cdot \partial_x g + (256 - [218.651])\langle v\partial_x g\rangle + (256^2 - [65417.618])v \cdot \partial_x \rho + \cdots$ |
| YES | NO | $\partial_t g = (-256^2 + [3151.269])g - (256 - [2.398])v \cdot \partial_x g + (256 - [20.289])\langle v\partial_x g\rangle - (256^2 - [3089.949])v \cdot \partial_x \rho + \cdots$ |
| YES | YES | $\partial_t g = (-256^2 + [1613.457])g - (256 + [4.126])v \cdot \partial_x g + (256 - [6.598])\langle v\partial_x g\rangle - (256^2 + [1600.326])v \cdot \partial_x \rho + \cdots$ |

TABLE 9
*Learned g-equation with and without sparse regularity assumptions.*

| $\varepsilon$ | $\langle g\rangle = 0$ APPLIED | LEARNED $g$-EQUATION | ERROR |
|---|---|---|---|
| 1/4 | NO | $\partial_t g = (-4^2 + [0.197])g - (4 + [0.105])v \cdot \partial_x g + (4 + [0.063])\langle v\partial_x g\rangle - (4^2 + [0.030])v \cdot \partial_x \rho + \cdots$ | 0.98% |
| 1/4 | YES | $\partial_t g = (-4^2 + [0.318])g - (4 + [0.006])v \cdot \partial_x g + (4 + [0.006])\langle v\partial_x g\rangle - (4^2 + [0.028])v \cdot \partial_x \rho + \cdots$ | 0.89% |
| 1/8 | NO | $\partial_t g = (-8^2 - [3.183])g - (8 - [0.342])v \cdot \partial_x g + (8 - [0.174])\langle v\partial_x g\rangle - (8^2 + [0.004])v \cdot \partial_x \rho + \cdots$ | 2.57% |
| 1/8 | YES | $\partial_t g = (-8^2 - [1.886])g - (8 - [0.104])v \cdot \partial_x g + (8 - [0.107])\langle v\partial_x g\rangle - (8^2 - [0.036])v \cdot \partial_x \rho + \cdots$ | 1.48% |

TABLE 10
*Learned g-equation with and without $\langle g\rangle = 0$ regularity.*

*algorithm also identified terms that are not supposed to be involved in the dynamics. The weights of the erroneous terms were so large that overall, the algorithm had a large error. For the STRidge algorithm, the main source of error is likely in the hard threshold assumption.*

EXAMPLE 6.8. ***Comparison with Conventional Methods. (Part 3)*** *Now we discuss a purely machine learning based algorithm presented in [34, 35]. In [34, 35], the authors suggest forming neural net approximations to the data which we denote by $\mathcal{N}_g$ and $\mathcal{N}_\rho$. The differential operator:*

$$F(\rho, g) = \partial_t g - (\lambda_1 v\partial_x g + \lambda_2\langle v\partial_x g\rangle + \lambda_3 v\partial_x \rho + \lambda_4 g) \tag{6.8}$$

*can be computed using backpropagation. The loss is given by:*

$$Loss = ||F(\rho, g)|| + ||g - \mathcal{N}_g|| + ||\rho - \mathcal{N}_\rho||, \tag{6.9}$$

*For small $\varepsilon$, we obtain a mediocre fit to the data using the loss given by Equation (6.9). Motivated by Equation (3.4), we redefine $F(\rho, g)$ to:*

$$\begin{aligned} F(\rho, g) =& \partial_t g - (\lambda_{1,0} + \frac{\lambda_{1,1}}{\varepsilon_{pred}} + \frac{\lambda_{1,2}}{\varepsilon_{pred}^2})v\partial_x g - (\lambda_{2,0} + \frac{\lambda_{2,1}}{\varepsilon_{pred}} + \frac{\lambda_{2,2}}{\varepsilon_{pred}^2})\langle v\partial_x g\rangle \\ &- (\lambda_{3,0} + \frac{\lambda_{3,1}}{\varepsilon_{pred}} + \frac{\lambda_{3,2}}{\varepsilon_{pred}^2})v\partial_x \rho - (\lambda_{4,0} + \frac{\lambda_{4,1}}{\varepsilon_{pred}} + \frac{\lambda_{4,2}}{\varepsilon_{pred}^2})g, \end{aligned} \tag{6.10}$$

*and use sparse $\lambda_{i,j}$ parameters. Equation (6.10) yields a much better fit to the data. The results are recorded in Table 13 where "No PT-Expansion" corresponds to fitting with Equation (6.8) and "Yes PT-Expansion" corresponds to fitting with Equation (6.10). We note that our algorithm is more adept as we do not already assume to know the terms involved in the dynamics as in [34, 35].*

| ORDER | LEARNED $g$-EQUATION | ERROR |
|---|---|---|
| 2ND | $\partial_t g = -(64^2 - 3.03)g - (64 + 0.006)v \cdot \partial_x g + (64 + 0.009)\langle v\partial_x g\rangle - (64^2 - 0.58)v \cdot \partial_x \rho$ | 0.043% |
| 4TH | $\partial_t g = -(64^2 - 1.43)g - (64 - 0.002)v \cdot \partial_x g + (64 + 0.011)\langle v\partial_x g\rangle - (64^2 + 0.12)v \cdot \partial_x \rho$ | 0.018% |

TABLE 11
*Learned g-equation using second and fourth order schemes.*

| METHOD | LEARNED $g$-EQUATION | ERROR |
|---|---|---|
| LASSO | $\partial_t g = -(128^2 - 10.04)g - (128 + 0.00)v \cdot \partial_x g + (128 + 26.35)\langle v\partial_x g\rangle - (128^2 + 23.42)v \cdot \partial_x \rho$ | 0.18% |
| PT-BASED ML | $\partial_t g = -(128^2 - 16.92)g - (128 - 1.09)v \cdot \partial_x g + (128 + 0.24)\langle v\partial_x g\rangle - (128^2 + 5.62)v \cdot \partial_x \rho$ | 0.072% |
| LASSO | $\partial_t g = -(256^2 - 1271.98)g - (256 + 4.13)v \cdot \partial_x g + (256 - 70.52)\langle v\partial_x g\rangle - (256^2 - 985.73)v \cdot \partial_x \rho$ | 1.77% |
| PT-BASED ML | $\partial_t g = -(256^2 + 72.69)g - (256 - 3.61)v \cdot \partial_x g + (256 + 0.44)\langle v\partial_x g\rangle - (256^2 - 32.29)v \cdot \partial_x \rho$ | 0.08% |

TABLE 12
*Learned g-equation. Comparison with Lasso method.*

EXAMPLE 6.9. ***Comparison with Conventional Methods. (Part 4)*** *We now compare our results with the multiscale hierarchical deep learning (MS-HDL) approach proposed in [26]. The approach in [26] is to train separate feed-forward neural networks $\boldsymbol{F}_j(\boldsymbol{x}, \Delta t_j)$ for different time scales $\Delta t_j$:*

$$\boldsymbol{x}_{t+\Delta t_j} = \boldsymbol{x}_t + \boldsymbol{F}_j(\boldsymbol{x}, \Delta t_j). \tag{6.11}$$

*For example, $\Delta t_j$ could be set to slow, medium, and fast scales by setting $\Delta t_j = \dfrac{\Delta t}{\varepsilon^j}$ for some fixed $\varepsilon$ and $j = 0, 1, 2$. Unfortunately, [26] does not provide a method for determining operators involved for each $\boldsymbol{F}_j(\boldsymbol{x}, \Delta t_j)$. Since we are interested in discovering the dynamics, we fit the $\boldsymbol{F}_j$ using the same 18 terms (denoted by $\mathcal{A}_i(v, x, t)$ for $i = 1, 2, cdots, 18$) as in example 6.6:*

$$\boldsymbol{F}_j(\boldsymbol{x}, t_n) := \sum_{i=1}^{18} \lambda_{i,j}\mathcal{A}_i(v, x, t_n). \tag{6.12}$$

*As suggested in Equation (6.11), we propagate data using the forward Euler scheme. Thus, the $\lambda_{i,j}$ are determined using the loss in Equation (4.4).*

*To be clear, Equation (6.11) is used to determine the dynamics of each map $\boldsymbol{F}_j(\boldsymbol{x}, t)$ separately. Thus, the desired equations the MS-HDL would like to uncover are:*

$$\begin{aligned} \partial_t g_{fast} &= -\frac{\sigma^A}{\varepsilon^2} g_{fast} - \frac{1}{\varepsilon^2} v\partial_x \rho, \\ \partial_t g_{medium} &= -\frac{1}{\varepsilon}(v\partial_x g_{medium} - \langle v\partial_x g_{medium}\rangle), \\ \partial_t g_{slow} &= -\sigma^A g_{slow}. \end{aligned} \tag{6.13}$$

*We use $\sigma^S = 1$, $\sigma^A = 0$, and $G(x) = 0$ to produce the data so that only fast and medium scales are present. For the MS-HDL, we choose $\Delta t_j = \frac{\Delta t}{\varepsilon^j}$, $j = 0, 1, 2$ with the correct value of $\varepsilon$. In [26], the authors*

| PT-expansion | learned $g$-equation | Error |
|---|---|---|
| No | $\partial_t g = -(64^2 + 4057.11)g - (64 + 30.64)v \cdot \partial_x g$ $+(64 + 30.91)\langle v\partial_x g\rangle - (64^2 + 4057.01)v \cdot \partial_x \rho$ | Large |
| Yes | $\partial_t g = -(64^2 - .075)g - (64 - 0.00)v \cdot \partial_x g$ $+(64 + 0.00)\langle v\partial_x g\rangle - (64^2 + 0.87)v \cdot \partial_x \rho$ | Small |

Table 13
*Learned g-equation using alternate machine learning fitting.*

| Method | Learned $g$-equation | Error |
|---|---|---|
| MS-HDL | $-(64^2 - 3.64)g + (64 + 1.99)v\partial_x g$ $-(64 + 0.87)\langle v\partial_x g\rangle + (64^2 + 1.36)v\partial_x \rho$ | 0.094% |
| DC-RNN | $-(64^2 + 0.56)g + (64 + 3.43)v\partial_x g$ $-(64 + 0.76)\langle v\partial_x g\rangle(4096 + 0.78)v\partial_x \rho$ | 0.066% |
| MS-HDL | $-(128^2 - 10.09)g + (128 + 4.84)v\partial_x g$ $-(128 + 1.87)\langle v\partial_x g\rangle + (128^2 + 6.54)v\partial_x \rho$ | 0.070% |
| DC-RNN | $-(128^2 - 15.89)g + (128 + 7.34)v\partial_x g$ $-(128 + 1.18)\langle v\partial_x g\rangle + (128^2 - 8.85)v\partial_x \rho$ | 0.010% |

Table 14
*Learned g-equation using Multiscale Deep Learning methods.*

*suggest gathering data for each time scale:*

$$\begin{aligned} g_{fast}(v,x,t_n) &= g(v,x,n\Delta t_2), && n = 0,1,2,...,N_{fast} \\ g_{medium}(v,x,t_n) &= g(v,x,n\Delta t_1), && n = 0,1,2,...,N_{medium} \\ g_{slow}(v,x,t_n) &= g(v,x,n\Delta t_0), && n = 0,1,2,...,N_{slow}, \end{aligned} \tag{6.14}$$

*i.e. the coarseness of the time grid determines the time scales. Of course, gathering data as in (6.14) can be a problem. Namely, (6.14) is only an approximation to the dynamics of (6.13). Thus, for our numerical example, we made the extra effort to perfectly split the data into different orders. In practice, it may be difficult to accurately split the data into different orders. For our DC-RNN algorithm, we do not need to split the data. The data for the DC-RNN is collected by:*

$$g(v,x,t_n) = g(v,x,n\Delta t) \qquad n = 0,1,2,...,N_t. \tag{6.15}$$

*Thus, one reason to prefer using DC-RNN over the MS-HDL is that one does not need to make the extra effort to split the data into different orders. Also, in the DC-RNN method we do not have to choose $\Delta t_j$ before hand, the DC-RNN algorithm learns appropriate time scales via Equation (5.1) in an automatic manner. We compare our DC-RNN method with the MS-HDL method in Table 14.*

Example 6.10. ***The diffusion limit.*** *As mentioned in Section 2, the equation for $\rho$ is given by equation (2.4). However, after applying the Chapman-Enskog expansion, one obtains Equation (2.7). Thus, if $\varepsilon$ is small enough, each equation is nearly equally likely to be predicted. Whether Equation (2.4) or (2.7) gets predicted likely depends on the algorithm used to minimize the loss. For our experiments, we use the Adam method followed by L-BFGS-B optimization.*

*In this example, we choose $\sigma^S = 1/3$, $\sigma^A = 0$, and $G = 0$. This means that in the limit $\varepsilon \to 0$, the dynamics of $\rho$ depends on either the terms $\langle v\partial_x g\rangle$ or $\partial_{xx}\rho$. The algorithm may deduce that each term has equal weights. However, because of the $\ell_1$ sparsity condition, the algorithm tends to place all the weights on either $\langle v\partial_x g\rangle$ or $\partial_{xx}\rho$. We summarize the numerical experiments in Table 15.*

| epsilon | Learned $\rho$-equation |
|---|---|
| 1/16 | $\partial_t \rho = (-0.000141)\partial_{xx}\rho - (0.993171)\langle v\partial_x g\rangle + \cdots$ |
| 1/256 | $\partial_t \rho = (-0.0436861)\partial_{xx}\rho - (1.042619)\langle v\partial_x g\rangle + \cdots$ |
| 1/2048 | $\partial_t \rho = (0.985353)\partial_{xx}\rho - (0.003806)\langle v\partial_x g\rangle + \cdots$ |
| 1/4096 | $\partial_t \rho = (0.985596)\partial_{xx}\rho - (0.010862)\langle v\partial_x g\rangle + \cdots$ |

Table 15
*Learned $\rho$-equation for various values of $\varepsilon$.*

## 7. Conclusion.

We propose a deep learning algorithm capable of learning time-dependent multiscale and nonlocal partial differential equations (PDEs) from data. The key to achieving our goal is to construct a Densely Connected Recurring Neural Network (DC-RNN) that accounts for potential multiscale and nonlocal structures in the data. The DC-RNN is a symbolic network with relationship among the symbols given by high-order IMEX schemes used to target dynamics of stiff PDEs describing kinetic equations. Incorporated into the training of the network are physics-aware constraints. Through various numerical experiements, we verify that our DC-RNN accurately and efficiently recovers multiscale PDEs which the data satisfies. As a byproduct, our DC-RNN determines appropriate multiscale parameters and can potentially discover lower dimensional representations for kinetic equations.

## 8. Appendix.

Here we present details on how to define a loss function which makes use of high-order IMEX schemes to fit data to Equations (2.4) and (2.5).

### 8.1. Higher-order IMEX Runge-Kutta fitting.

Higher-order fitting can be done following the high-order IMEX schemes for solving Equations (2.4) and (2.5). For the remainder of this section, we omit the spatial discritization of the spacial operators. Generally, higher-order spacial discritization should be used for higher-order IMEX schemes for numerical stability (see [4]). The time-steps are denoted by superscripts while stages are denoted by superscripts enclosed in parenthesis. The higher-order K-stage IMEX Runge-Kutta scheme is given by:

$$\begin{aligned} g^{(i)} = g^n &- \Delta t \sum_{j=1}^{i-1} \widetilde{a}_{i,j} \left( \frac{1}{\varepsilon}(I - \langle\rangle)(v\partial_x g^{(j)}) + \frac{1}{\varepsilon^2} v\partial_x \rho^{(j)} + \sigma^A g^{(j)} \right) \\ &- \Delta t \sum_{j=1}^{i} a_{i,j} \left( \frac{\sigma^S}{\varepsilon^2} g^{(j)} \right), \end{aligned} \tag{8.1}$$

$$\rho^{(i)} = \rho^n - \Delta t \sum_{j=1}^{i-1} \widetilde{a}_{i,j} (\sigma^A \rho^{(j)} - G) - \Delta t \sum_{j=1}^{i} a_{i,j} \partial_x \langle v g^{(j)} \rangle, \tag{8.2}$$

$$\begin{aligned} g^{n+1} = g^n &- \Delta t \sum_{i=1}^{K} \widetilde{w}_i \left( \frac{1}{\varepsilon}(I - \langle\rangle)(v\partial_x g^{(i)}) + \frac{1}{\varepsilon^2} v\partial_x \rho^{(i)} + \sigma^A g^{(i)} \right) \\ &- \Delta t \sum_{j=1}^{K} w_i \left( \frac{\sigma^S}{\varepsilon^2} g^{(j)} \right), \end{aligned} \tag{8.3}$$

$$\rho^{n+1} = \rho^n - \Delta t \sum_{i=1}^{K} \widetilde{w}_i (\sigma^A \rho^{(i)} - G) - \Delta t \sum_{i=1}^{K} w_i \partial_x \langle v g^{(i)} \rangle. \tag{8.4}$$

Equations (8.1) and (8.2) are intermediate stages and Equations (8.3) and (8.4) are the approximate solution at the next time step. Here $\widetilde{A} = (\widetilde{a}_{i,j})$ with $\widetilde{a}_{i,j} = 0$ for $j \geq i$ and $A = (a_{i,j})$ with $a_{i,j} = 0$ for $j > i$

are $K \times K$ matrices. Along with the coefficient vectors $\widetilde{\boldsymbol{w}} = (\widetilde{w}_1, \cdots, \widetilde{w}_K)^T$, $\boldsymbol{w} = (w_1, \cdots, w_K)^T$, they can be represented by a double Butcher tableau:

$$\begin{array}{c|c} \widetilde{\boldsymbol{c}} & \widetilde{A} \\ \hline & \widetilde{\boldsymbol{w}}^T \end{array} \qquad \text{and} \qquad \begin{array}{c|c} \boldsymbol{c} & A \\ \hline & \boldsymbol{w}^T \end{array},$$

where the vectors $\widetilde{\boldsymbol{c}} = (\widetilde{c}_1, \cdots, \widetilde{c}_K)^T$ and $\boldsymbol{c} = (c_1, \cdots, c_K)^T)$ are defined as:

$$\widetilde{c}_i = \sum_{j=1}^{i-1} \widetilde{a}_{i,j} \qquad \text{and} \qquad c_i = \sum_{j=1}^{i-1} a_{i,j}. \tag{8.5}$$

For convenience, we provide the tableau for the ARS(2,2,2) scheme:

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \gamma & \gamma & 0 & 0 \\ 1 & \delta & 1-\delta & 0 \\ \hline & \delta & 1-\delta & 0 \end{array} \qquad \text{and} \qquad \begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \gamma & 0 & \gamma & 0 \\ 1 & 0 & 1-\gamma & \gamma \\ \hline & 0 & 1-\gamma & \gamma \end{array},$$

where $\gamma = 1 - \dfrac{\sqrt{2}}{2}$ and $\delta = 1 - \dfrac{1}{2\gamma}$.

The loss function based on this fitting scheme is defined by:

$$L = \frac{1}{Nt-1} \sum_{n=1}^{Nt-1} ||\mathcal{K}_g^n|| + ||\mathcal{K}_\rho^n||, \tag{8.6}$$

with,

$$\mathcal{K}_g^n = \mathcal{K}_g^n(\{g(v,x,t_n), g(v,x,t_{n+1})\}), \tag{8.7}$$

$$\mathcal{K}_\rho^n = \mathcal{K}_\rho^n(\{\rho(x,t_n), \rho(x,t_{n+1})\}), \tag{8.8}$$

to be defined below.

$$\begin{aligned} \mathcal{K}_g^n := g(v,x,t_{n+1}) - g(v,x,t_n) + \Delta t \left( \sum_{i=1}^{K} \sigma^A(x) \widetilde{w}_i g^{(i)} + \frac{\sigma^S(x)}{\varepsilon^2} w_i g^{(i)} \right) \\ + \Delta t \sum_{i=1}^{K} \widetilde{w}_i \left( \mathcal{F}_1(g^{(i)}(v,x), \rho^{(i)}(x)) \right) \end{aligned}, \tag{8.9}$$

$$\begin{aligned} \mathcal{K}_\rho^n := \rho(x,t_{n+1}) - \rho(x,t_n) + \Delta t \sum_{i=1}^{K} \widetilde{w}_i (\sigma^A(x) \rho^{(i)} - G(x)) \\ + \Delta t \sum_{i=1}^{K} w_i \left( \mathcal{F}_2(g^{(i)}(v,x), \rho^{(i)}(x)) \right). \end{aligned} \tag{8.10}$$

The operators $\mathcal{F}_1(g,\rho)$, $\mathcal{F}_2(g,\rho)$ are given by (3.5) and are generated by the RNN in Equation (3.8). The intermediate stages are given by:

$$\begin{aligned} g^{(i)} = g(v,x,t_n) - \Delta t \sum_{j=1}^{i} a_{i,j} \frac{\sigma^S(x)}{\varepsilon^2} g^{(j)} \\ - \Delta t \sum_{j=1}^{i-1} \widetilde{a}_{i,j} \left( \mathcal{F}_1(g^{(j)}, \rho^{(j)}) \right) \end{aligned} \tag{8.11}$$

TABLE 16

| $q$ | $\alpha$ | $\gamma$ | $\beta$ |
|---|---|---|---|
| 1 | $(-1, 1)$ | 1 | 1 |
| 2 | $(\frac{1}{3}, -\frac{4}{3}, 1)$ | $(-\frac{2}{3}, \frac{4}{3})$ | $\frac{2}{3}$ |
| 3 | $(-\frac{2}{11}, \frac{9}{11}, -\frac{18}{11}, 1)$ | $(\frac{6}{11}, -\frac{18}{11}, \frac{18}{11})$ | $\frac{6}{11}$ |
| 4 | $(\frac{3}{25}, -\frac{16}{25}, \frac{36}{25}, -\frac{48}{25}, 1)$ | $(-\frac{12}{25}, \frac{48}{25}, -\frac{72}{25}, \frac{48}{25})$ | $\frac{12}{25}$ |

$$\begin{aligned}\rho^{(i)} = \rho(x,t_n) &- \Delta t \sum_{j=1}^{i-1} \widetilde{a}_{i,j}(\sigma^A(x)\rho^{(j)} - G) \\ &- \Delta t \sum_{j=1}^{i} \widetilde{a}_{i,j}\left(\mathcal{F}_2(g^{(j)}, \rho^{(j)})\right)\end{aligned} \tag{8.12}$$

We note that $\sigma^A(x)$, $\sigma^S(x)$, and $G(x)$ do not need to be assumed known. These functions can be part of the fitting process by replacing them with feed-forward neural nets, say.

**8.2. Higher-order IMEX-BDF fitting.** Another way to go higher-order in time is through the IMEX-BDF scheme [11]:

$$\begin{aligned}\sum_{i=0}^{q} \alpha_i g^{n+i} + \Delta t \sum_{i=0}^{q-1} \gamma_i \bigg(\frac{1}{\varepsilon}(I - \langle\rangle)(v\partial_x g^{n+i}) \\ + \frac{1}{\varepsilon^2} v\partial_x \rho^{n+i} + \sigma^A g^{n+i}\bigg) + \beta\Delta t \left(\frac{\sigma^S}{\varepsilon^2} g^{n+q}\right) = 0,\end{aligned} \tag{8.13}$$

and

$$\sum_{i=0}^{q} \alpha_i \rho^{n+i} + \Delta t \sum_{i=0}^{q-1} \gamma_i(\sigma^A \rho^{n+i} - G) + \beta\Delta t \partial_x \langle v g^{n+q}\rangle = 0. \tag{8.14}$$

We display some coefficients $\alpha = (\alpha_0, \cdots, \alpha_q)$, $\gamma = (\gamma_0, \cdots, \gamma_{q-1})$, and $\beta$ for the above scheme in Table 16.

The loss function for the fitting scheme based on the IMEX-BDF method, is defined by:

$$L = \frac{1}{N_t - q} \sum_{n=1}^{Nt-q} ||\mathcal{K}^n(D; \boldsymbol{\theta})||, \tag{8.15}$$

with,

$$D = \{u(x,t_n), u(x,t_{n+1}) \cdots, u(x,t_{n+q})\}. \tag{8.16}$$

For the $g$ equation $\mathcal{K}_g^n$ is given by:

$$\begin{aligned}\mathcal{K}_g^n = &\sum_{i=0}^{q} \alpha_i g^{n+i} - \beta\Delta t \frac{\sigma^S(x)}{\varepsilon^2} g^{n+q} - \Delta t \sum_{i=0}^{q-1} \sigma^A(x) g^{n+i} \\ &+ \Delta t \sum_{i=0}^{q-1} \gamma_i \left(\mathcal{F}_1(g(v,x,t_{n+i}), \rho(x,t_{n+i}))\right).\end{aligned} \tag{8.17}$$

The operator $\mathcal{F}_1(g, \rho)$ is given by (3.5) and is generated by the RNN in Equation (3.8).

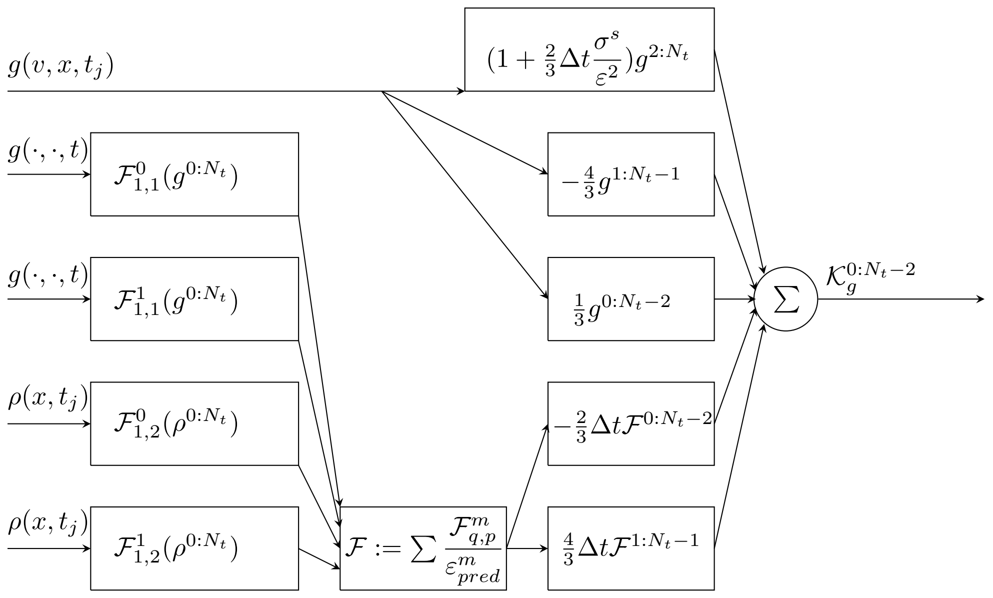
$g(v,x,t_j)$
$(1+\frac{2}{3}\Delta t\frac{\sigma^s}{\varepsilon^2})g^{2:N_t}$
$g(\cdot,\cdot,t)$
$\mathcal{F}^0_{1,1}(g^{0:N_t})$
$-\frac{4}{3}g^{1:N_t-1}$
$g(\cdot,\cdot,t)$
$\mathcal{F}^1_{1,1}(g^{0:N_t})$
$\frac{1}{3}g^{0:N_t-2}$
$\sum$
$\mathcal{K}_g^{0:N_t-2}$
$\rho(x,t_j)$
$\mathcal{F}^0_{1,2}(\rho^{0:N_t})$
$-\frac{2}{3}\Delta t\mathcal{F}^{0:N_t-2}$
$\rho(x,t_j)$
$\mathcal{F}^1_{1,2}(\rho^{0:N_t})$
$\mathcal{F}:=\sum\frac{\mathcal{F}^m_{q,p}}{\varepsilon^m_{pred}}$
$\frac{4}{3}\Delta t\mathcal{F}^{1:N_t-1}$

FIG. 6. *Example DC-RNN based on IMEX-BDF-2 scheme for predicting the g-equation. The inputs are $\rho(t)$, and $g(t)$. The dictionary contains order $\mathcal{O}(1)$ and $\mathcal{O}(\varepsilon)$ operators. These operators are generated by the RNNs of orders $\varepsilon^{-m}$ $m=0,1$. The output $\mathcal{K}_g^{0:N_t-2}$ is to be minimized with respect to a chosen norm.*

For the $\rho$ equation $\mathcal{K}_g^n$ is given by:

$$\mathcal{K}_\rho^n = \sum_{i=0}^{q}\alpha_i g^{n+i} + \Delta t\sum_{i=0}^{q-1}\gamma_i\left(\sigma^A\rho^{n+i} - G\right) - \beta\Delta t\left(\mathcal{F}_2(g(v,x,t_{n+q}),\rho(x,t_{n+q}))\right). \tag{8.18}$$

Again, $\sigma^A(x)$, $\sigma^S(x)$, and $G(x)$ can be learned by including them in the fitting process. We display in Figure 6 a DC-RNN for determining the equation satisfied by $g(v,x,t)$ based on the IMEX-BDF-2 scheme.

## REFERENCES

[1] Z. Allen-Zhu and Y. Li, *Can sgd learn recurrent neural networks with provable generalization?*, in NeurIPS, 2019.

[2] Z. Allen-Zhu, Y. Li, and Z. Song, *On the convergence rate of training recurrent neural networks*, in Advances in Neural Information Processing Systems, vol. 32, Curran Associates, Inc., 2019.

[3] U. M. Ascher, S. J. Ruuth, and R. J. Spiteri, *Implicit-explicit runge-kutta methods for time-dependent partial differential equations*, Applied Numerical Mathematics, 25 (1997), pp. 151 – 167, https://doi.org/https://doi.org/10.1016/S0168-9274(97)00056-1, http://www.sciencedirect.com/science/article/pii/S0168927497000561. Special Issue on Time Integration.

[4] S. Boscarino, L. Pareschi, and G. Russo, *Imex runge-kutta schemes and hyperbolic systems of conservation laws with stiff diffusive relaxation*, AIP Conference Proceedings, (2009), https://doi.org/10.1063/1.3241248.

[5] S. Brunton, J. Proctor, and J. Kutz, *Discovering governing equations from data: Sparse identification of nonlinear dynamical systems*, Proceedings of the National Academy of Sciences, 113 (2015), p. 39323937, https://doi.org/10.1073/pnas.1517384113.

[6] S. L. Brunton, J. L. Proctor, and J. N. Kutz, *Discovering governing equations from data by sparse identification of nonlinear dynamical systems*, Proceedings of the National Academy of Sciences, 113 (2016), pp. 3932–3937, https://doi.org/10.1073/pnas.1517384113, https://www.pnas.org/content/113/15/3932, https://arxiv.org/abs/https://www.pnas.org/content/113/15/3932.full.pdf.

[7] K. Champion, S. Brunton, and J. Kutz, *Discovery of nonlinear multiscale systems: Sampling strategies and embeddings*, SIAM Journal on Applied Dynamical Systems, 18 (2019), pp. 312–333, https://doi.org/10.1137/18M1188227.

[8] S. Chandrasekhar, *Radiative Transfer*, Dover Publications, 1960.

[9] K. A. F. Copeland, *Local polynomial modelling and its applications*, Journal of Quality Technology, 29 (1997), pp. 234–234, https://doi.org/10.1080/00224065.1997.11979762, https://doi.org/10.1080/00224065.1997.11979762, https://arxiv.org/abs/https://doi.org/10.1080/00224065.1997.11979762.

[10] B. Davison, *Neutron Transport Theory*, Oxford University Press, London, 1973.

[11] G. Dimarco and L. Pareschi, *Implicit-explicit linear multistep methods for stiff kinetic equations*, SIAM Journal on Numerical Analysis, 55 (2016), https://doi.org/10.1137/16M1063824.

[12] Q. Du, Y. Gu, H. Yang, and C. Zhou, *The discovery of dynamics via linear multistep methods and deep learning: Error estimation*, arxiv:2103.11488, (2021).

[13] J. Gergonne, *The application of the method of least squares to the interpolation of sequences*, Historia Mathematica, 1 (1974), pp. 439–447, https://doi.org/https://doi.org/10.1016/0315-0860(74)90034-2, https://www.sciencedirect.com/science/article/pii/0315086074900342.

[14] J. Harlim, S. W. Jiang, S. Liang, and H. Yang, *Machine learning for prediction with missing dynamics*, 2020, https://arxiv.org/abs/1910.05861.

[15] T. Hastie, R. Tibshirani, and M. Wainwright, *Statistical learning with sparsity: The lasso and generalizations*, Chapman and Hall/CRC, 01 2015, https://doi.org/10.1201/b18401.

[16] S. Jin, *Efficient asymptotic-preserving (ap) schemes for some multiscale kinetic equations*, SIAM Journal on Scientific Computing, 21 (1999), pp. 441–454, https://doi.org/10.1137/S1064827598334599, https://doi.org/10.1137/S1064827598334599, https://arxiv.org/abs/https://doi.org/10.1137/S1064827598334599.

[17] S. Jin and L. Pareschi, *Asymptotic-preserving (ap) schemes for multiscale kinetic equations: a unified approach*, in Hyperbolic Problems: Theory, Numerics, Applications, H. Freistühler and G. Warnecke, eds., Basel, 2001, Birkhäuser Basel, pp. 573–582.

[18] S. Jin, L. Pareschi, and G. Toscani, *Diffusive relaxation schemes for multiscale discrete-velocity kinetic equations*, SIAM Journal on Numerical Analysis, 35 (1998), pp. 2405–2439, https://doi.org/10.1137/S0036142997315962, https://doi.org/10.1137/S0036142997315962, https://arxiv.org/abs/https://doi.org/10.1137/S0036142997315962.

[19] E. Kaiser, J. N. Kutz, and S. L. Brunton, *Sparse identification of nonlinear dynamics for model predictive control in the low-data limit*, in Proceedings of the Royal Society A, 2018.

[20] R. Keller and Q. Du, *Discovery of dynamics using linear multistep methods*, 2020, https://arxiv.org/abs/1912.12728.

[21] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, CoRR, abs/1412.6980 (2014).

[22] M. Lemou and L. Mieussens, *A new asymptotic preserving scheme based on micro-macro formulation for linear kinetic equations in the diffusion limit*, SIAM Journal on Scientific Computing, 31 (2008), pp. 334–368, https://doi.org/10.1137/07069479X, https://doi.org/10.1137/07069479X, https://arxiv.org/abs/https://doi.org/10.1137/07069479X.

[23] Z. Li, J. Han, W. E, and Q. Li, *On the curse of memory in recurrent neural networks: Approximation and optimization analysis*, 2020, https://arxiv.org/abs/2009.07799.

[24] S. Liang, L. Lyu, C. Wang, and H. Yang, *Reproducing activation function for deep learning*, arXiv:2101.04844, (2021).

[25] J.-G. Liu and L. Mieussens, *Analysis of an asymptotic preserving scheme for linear kinetic equations in the diffusion limit*, SIAM Journal on Numerical Analysis, 48 (2010), pp. 1474–1491, https://doi.org/10.1137/090772770, https://doi.org/10.1137/090772770, https://arxiv.org/abs/https://doi.org/10.1137/090772770.

[26] Y. Liu, J. Kutz, and S. Brunton, *Hierarchical deep learning of multiscale differential equation time-steppers*, ArXiv, abs/2008.09768 (2020).

[27] Z. Long, Y. Lu, and B. Dong, *Pde-net 2.0: Learning pdes from data with A numeric-symbolic hybrid deep network*, CoRR, abs/1812.04426 (2018), http://arxiv.org/abs/1812.04426, https://arxiv.org/abs/1812.04426.

[28] Z. Long, Y. Lu, X. Ma, and B. Dong, *Pde-net: Learning pdes from data*, in ICML, 2018.

[29] J. Lu, Z. Shen, H. Yang, and S. Zhang, *Deep network approximation for smooth functions*, arXiv e-prints, (2020), arXiv:2001.03040, p. arXiv:2001.03040, https://arxiv.org/abs/2001.03040.

[30] N. M. Mangan, J. N. Kutz, S. L. Brunton, and J. L. Proctor, *Model selection for dynamical systems via sparse regression and information criteria*, in Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 2017.

[31] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao, *Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review*, International Journal of Automation and Computing, 14 (2017), pp. 503–519.

[32] M. Raissi and G. E. Karniadakis, *Hidden physics models: Machine learning of nonlinear partial differential equations*, Journal of Computational Physics, 357 (2018), pp. 125 – 141, https://doi.org/https://doi.org/10.1016/j.jcp.2017.11.039, http://www.sciencedirect.com/science/article/pii/S0021999117309014.

[33] M. Raissi, P. Perdikaris, and G. Karniadakis, *Multistep neural networks for data-driven discovery of nonlinear dynamical systems*, arXiv: Dynamical Systems, (2018).

[34] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics informed deep learning (part I): data-driven solutions of nonlinear partial differential equations*, CoRR, abs/1711.10561 (2017), http://arxiv.org/abs/1711.10561, https://arxiv.org/abs/1711.10561.

[35] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics informed deep learning (part II): data-driven discovery of nonlinear partial differential equations*, CoRR, abs/1711.10566 (2017), http://arxiv.org/abs/1711.10566, https://arxiv.org/abs/1711.10566.

[36] S. Rudy, S. Brunton, J. Proctor, and J. Kutz, *Data-driven discovery of partial differential equations*, Science Advances, 3 (2016), https://doi.org/10.1126/sciadv.1602614.

[37] H. Schaeffer, G. Tran, and R. Ward, *Extracting sparse high-dimensional dynamics from limited data*, SIAM Journal of Applied Mathematics, 78 (2017), pp. 3279–3295.

[38] Z. Shen, H. Yang, and S. Zhang, *Deep network approximation characterized by number of neurons*, Communications in Computational Physics, 28 (2020), pp. 1768–1811, https://doi.org/https://doi.org/10.4208/cicp.OA-2020-0149, http://global-sci.org/intro/article_detail/cicp/18396.html.

[39] T. Swart and V. Rousse, *A mathematical justification for the herman-kluk propagator*, Communications in Mathematical Physics - COMMUN MATH PHYS, 286 (2007), https://doi.org/10.1007/s00220-008-0681-4.

[40] R. Tibshirani, *Regression shrinkage and selection via the lasso*, Journal of the Royal Statistical Society: Series B (Methodological), 58 (1996), pp. 267–288, https://doi.org/https://doi.org/10.1111/j.2517-6161.1996.tb02080.x, https://rss.

onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1996.tb02080.x, https://arxiv.org/abs/https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.2517-6161.1996.tb02080.x.

[41] C. Villani, *A review of mathematical topics in collisional kinetic theory*, in Handbook of Mathematical Fluid Mechanics, S. Friedlander and D. Serre, eds., vol. I, North-Holland, 2002, pp. 71–305.

[42] Y. Wang, S. W. Cheung, E. T. Chung, Y. Efendiev, and M. Wang, *Deep multiscale model learning*, 2018, https://arxiv.org/abs/1806.04830.

[43] K. Wu and D. Xiu, *Data-driven deep learning of partial differential equations in modal space*, Journal of Computational Physics, 408 (2020), p. 109307, https://doi.org/https://doi.org/10.1016/j.jcp.2020.109307, https://www.sciencedirect.com/science/article/pii/S0021999120300814.

[44] D. Yarotsky, *Error bounds for approximations with deep ReLU networks*, Neural Networks, 94 (2017), pp. 103 – 114, https://doi.org/https://doi.org/10.1016/j.neunet.2017.07.002, http://www.sciencedirect.com/science/article/pii/S0893608017301545.

[45] H. Yu, X. Tian, W. E, and Q. Li, *Onsagernet: Learning stable and interpretable dynamics using a generalized onsager principle*, 2020, https://arxiv.org/abs/2009.02327.

[46] S. Zhang and G. Lin, *Robust data-driven discovery of governing physical laws with error bars*, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science, 474 (2018), p. 20180305, https://doi.org/10.1098/rspa.2018.0305.

[47] P. Zheng, T. Askham, S. L. Brunton, J. N. Kutz, and A. Y. Aravkin, *A unified framework for sparse relaxed regularized regression: Sr3*, IEEE Access, 7 (2019), pp. 1404–1423, https://doi.org/10.1109/ACCESS.2018.2886528.