Journal of Computational Physics 408 (2020) 109307

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

# Data-driven deep learning of partial differential equations in modal space ☆

Kailiang Wu, Dongbin Xiu *

*Department of Mathematics, The Ohio State University, Columbus, OH 43210, USA*

ARTICLE INFO

*Article history:*
Received 15 October 2019
Received in revised form 27 January 2020
Accepted 31 January 2020
Available online 6 February 2020

*Keywords:*
Deep neural network
Residual network
Governing equation discovery
Modal space

ABSTRACT

We present a framework for recovering/approximating unknown time-dependent partial differential equation (PDE) using its solution data. Instead of identifying the terms in the underlying PDE, we seek to approximate the evolution operator of the underlying PDE numerically. The evolution operator of the PDE, defined in infinite-dimensional space, maps the solution from a current time to a future time and completely characterizes the solution evolution of the underlying unknown PDE. Our recovery strategy relies on approximation of the evolution operator in a properly defined modal space, i.e., generalized Fourier space, in order to reduce the problem to finite dimensions. The finite dimensional approximation is then accomplished by training a deep neural network structure, which is based on residual network (ResNet), using the given data. Error analysis is provided to illustrate the predictive accuracy of the proposed method. A set of examples of different types of PDEs, including inviscid Burgers' equation that develops discontinuity in its solution, are presented to demonstrate the effectiveness of the proposed method.



## 1. Introduction

Recently there has been an ongoing research effort to develop data-driven methods for discovering unknown physical laws. Earlier attempts such as [2,31] used symbolic regression to select the proper physical laws and determine the underlying dynamical systems. More recent efforts tend to cast the problem as an approximation problem. In this approach, the sought-after governing equation is treated as an unknown target function relating the data of the state variables to their temporal derivatives. Methods along this line of approach usually seek exact recovery of the equations by using certain sparse approximation techniques (e.g., [33]) from a large set of dictionaries; see, for example, [4]. Studies have been conducted to deal with noises in data [4,29,10], corruptions in data [34], limited data [30], partial differential equations [26,28], etc. Variations of the approaches have been developed in conjunction with other methods such as model selection approach [15], Koopman theory [3], Gaussian process regression [21,20], and expectation-maximization approach [16], to name a few. Methods using standard basis functions and without requiring exact recovery were also developed for dynamical systems [39] and Hamiltonian systems [38].

There is a recent surge of interest in developing methods using modern machine learning techniques, particularly deep neural networks. The studies include recovery of ordinary differential equations (ODEs) [24,18,27] and partial differential equations (PDEs) [14,22,23,19,13,32]. It was shown that residual network (ResNet) is particularly suitable for equation re-

☆ Funding: This work was partially supported by AFOSR FA9550-18-1-0102.
* Corresponding author.
*E-mail addresses:* wu.3423@osu.edu (K. Wu), xiu.16@osu.edu (D. Xiu).

https://doi.org/10.1016/j.jcp.2020.109307
0021-9991/

covery, in the sense that it can be an exact integrator [18]. Neural networks have also been explored for other aspects of scientific computing, including reduced order modeling [9,17], solution of conservation laws [25,37], multiphase flow simulation [36], high-dimensional PDEs [6,12], uncertainty quantification [5,35,40,11], etc.

The focus of this paper is on the development of a general numerical framework for approximating/learning unknown time-dependent PDE. Even though the topic has been explored in several recent articles, cf., [14,22,23,19,13,32], the existing studies are relatively limited, as they mostly focus on learning certain types of PDE or identifying the exact terms in the PDE from a (large) dictionary of possible terms. The specific novelty of this paper is that the proposed method seeks to recover/approximate the evolution operator of the underlying unknown PDE and is applicable for general class of PDEs. The evolution operator completely characterizes the time evolution of the solution. Its recovery allows one to conduct prediction of the underlying PDE and is effectively equivalent to the recovery of the equation. This is an extension of the equation recovery work from [18], where the flow map of the underlying unknown dynamical system is the goal of recovery. Unlike the ODE systems considered in [18], PDE systems, which is the focus of this paper, are of infinite dimension. In order to cope with infinite dimension, our method first reduces the problem into finite dimensions by utilizing a properly chosen modal space, i.e., generalized Fourier space. The equation recovery task is then transformed into recovery of the generalized Fourier coefficients, which follow a finite dimensional dynamical system. The approximation of the finite dimensional evolution operator of the reduced system is then carried out by using deep neural network, particularly the residual network (ResNet) which has been shown to be particularly suitable for this task [18]. One of the advantages of the proposed method is that, by focusing on evolution operator, it eliminates the need for time derivatives data of the state variables. Time derivative data, often required by many existing methods, are difficult to acquire in practice and susceptible to (additional) errors when computed numerically. Moreover, the proposed method can cope with solution data that are more sparsely or unevenly distributed in time. Since the proposed framework is rather general, we present several examples of recovering different types of PDEs. These include linear advection, linear diffusion, viscous and inviscid nonlinear Burgers' equations. The inviscid Burgers' equation represents a relatively challenging problem, as it develops shock over time. Our results show that the proposed method is able to accurately capture the evolution operator using only smooth data during training. The reconstructed evolution operator is then able to produce shock structure developed over time during prediction. Most of our examples are in one dimensional physical space, as this allows us to easily and thoroughly examine the solutions and their numerical errors. Our last example is the recovery of a two-dimensional advection-diffusion equation. It demonstrates the applicability of the method to multiple dimensional PDEs.

This paper is organized as follows. After the problem setup in Section 2, we discuss evolution operator and its finite dimensional representation in Section 3. The numerical approach for learning the evolution operator is then presented in Section 4, along with an error analysis for the predictive accuracy. Numerical examples are then presented in Section 5 to demonstrate the properties of the proposed approach.

## 2. Problem setup

Let us consider a state variable $u(x,t)$, which is governed by an unknown autonomous time-dependent PDE system

$$\begin{cases} u_t = \mathcal{L}(u), & (x,t) \in \Omega \times \mathbb{R}^+, \\ \mathcal{B}(u) = 0, & (x,t) \in \partial\Omega \times \mathbb{R}^+, \\ u(x,0) = u_0(x), & x \in \bar{\Omega}, \end{cases} \tag{2.1}$$

where $t$ denotes the time, $x$ is the spatial variable, $\Omega$ is the physical domain, and $\mathcal{L}$ and $\mathcal{B}$ stand for the operators in the equations and boundary conditions, respectively. Our basic assumption is that the operator $\mathcal{L}$ is unknown. In this paper, we assume the boundary conditions are known and focus on learning the PDE in the interior of the domain.

We assume data about the solution $u(x,t)$ are available at certain time instances, loosely called "snapshots" hereafter. That is, we have data

$$w(x,t_j) = u(x,t_j) + \epsilon(x,t_j), \qquad j = 1,\dots,S, \tag{2.2}$$

where $S \geq 1$ is the total number of snapshots of the solution field and $\epsilon(x,t_j)$ stands for the noises/errors when the data are acquired. (Certain reconstruction procedure may be involved in order to have the snapshot data in the form of (2.2). This, however, is not the focus of this paper.)

Our goal is to accurately reconstruct the evolution/dynamics of the unknown governing equation (2.1) via the snapshot data (2.2). Once an accurate reconstruction is achieved, it can be used to provide predictions of the solution.

## 3. Finite dimensional approximation

While many of the existing equation learning methods seek to directly approximate or learn the specific form of the governing equations, we adopt a different framework, which seeks to approximate evolution operator of the underlying equations. Such an approach was presented and analyzed in [18] for recovery of ODEs. For the PDE recovery problem considered in this paper, our first task is to reduce the problem from infinite dimension to finite dimension.

### 3.1. Evolution operator

Without specifying the form of the governing equation, we loosely assume that for any fixed $t \geq 0$, the solution $u(x,t)$ belongs to a Hilbert space $\mathbb{V}$, with the space norm denoted by $\|\cdot\|_{\mathbb{V}}$. Moreover, we assume the known boundary conditions are linear, i.e., the boundary operator $\mathcal{B}$ in (2.1) is linear. For many commonly-used boundary conditions, e.g., Dirichlet, Neumann, or periodic boundary conditions, etc., this assumption holds true.

We restrict our attention to autonomous PDEs. Consequently, there exists an evolution operator

$$\mathcal{E}_\Delta : \mathbb{V} \to \mathbb{V}, \qquad \mathcal{E}_\Delta u(\cdot,t) = u(\cdot, t+\Delta). \tag{3.1}$$

Note that only the time difference, or time lag, $\Delta$ is relevant, as the time variable $t$ can be arbitrarily shifted. The evolution operator completely determines the solution over time. Once it is accurately approximated, one can iteratively apply the approximate evolution operator to conduct prediction of the system.

### 3.2. Finite dimensional evolution operator

To make the PDE learning problem tractable, we consider a finite dimensional space $\mathbb{V}_n \subset \mathbb{V}$. Let

$$\mathbf{\Phi}(x) = (\phi_1(x), \ldots, \phi_n(x))^\dagger, \qquad n \geq 1, \tag{3.2}$$

be a basis of $\mathbb{V}_n$ and satisfy $\mathcal{B}(\phi_j) = 0$, $1 \leq j \leq n$, i.e.,

$$\mathbb{V}_n = \text{span}\{\phi_j : \mathcal{B}(\phi_j) = 0,\ j = 1, \ldots, n\}. \tag{3.3}$$

An approximation of $u(x,t)$ from $\mathbb{V}_n$ can be written as

$$u_n(x,t) = \sum_{j=1}^{n} v_j(t)\phi_j(x) = \langle \mathbf{v}(t), \mathbf{\Phi}(x) \rangle, \tag{3.4}$$

where the last equality is written in vector notation after defining $\mathbf{v} = (v_1, \ldots, v_n)^\dagger$. Let $\mathcal{P}_n : \mathbb{V} \to \mathbb{V}_n$ be a projection operator. For any $t$, we define the projection of the exact solution as

$$\widehat{u}_n(x,t) := \mathcal{P}_n u(x,t) = \langle \widehat{\mathbf{v}}(t), \mathbf{\Phi}(x) \rangle. \tag{3.5}$$

To approximate the (unknown) infinite dimensional evolution operator $\mathcal{E}_\Delta$ in the finite dimensional space $\mathbb{V}_n$, we consider a finite dimensional evolution operator $\mathcal{E}_{\Delta,n}$, which evolves an approximate solution $u_n \in \mathbb{V}_n$, i.e.

$$\mathcal{E}_{\Delta,n} : \mathbb{V}_n \to \mathbb{V}_n, \qquad \mathcal{E}_{\Delta,n} u_n(\cdot,t) = u_n(\cdot, t+\Delta). \tag{3.6}$$

In practice, one may choose any suitable finite dimensional operator $\mathcal{E}_{\Delta,n}$, as long as it provides a good approximation to $\mathcal{E}_\Delta$, i.e., $\mathcal{E}_{\Delta,n} \approx \mathcal{E}_\Delta$. Note that this effectively assumes an autonomous PDE can be approximated by an autonomous discrete system via a proper spatial discretization. This is a very mild assumption used in virtually any numerical methods. In this paper, we mostly employ the following finite dimensional evolution operator

$$\widetilde{\mathcal{E}}_{\Delta,n} := \mathcal{P}_n \mathcal{E}_\Delta, \tag{3.7}$$

such that

$$\widetilde{\mathcal{E}}_{\Delta,n} v = \mathcal{P}_n \mathcal{E}_\Delta v, \qquad \forall v \in \mathbb{V}_n. \tag{3.8}$$

(Note that if one chooses $v = \mathcal{P}_n u$, then this operator closely resembles the evolution operator of spectral Galerkin method for solving an *known* PDE.) For this specific choice of $\mathcal{E}_{\Delta,n}$, we have the following error bound.

**Proposition 3.1.** *Assume the evolution operator $\mathcal{E}_\Delta$ (3.1) of the underlying PDE is bounded. Let $t_k = k\Delta$, $k = 0, 1, \ldots,$ and let $\varepsilon^{\text{proj}}(t_k) := \|u(\cdot,t_k) - \mathcal{P}_n u(\cdot,t_k)\|_{\mathbb{V}}$ be the projection error of the exact solution at $t_k$. Consider the approximate evolution operator $\widetilde{\mathcal{E}}_{\Delta,n}$ defined in (3.7) and its corresponding approximate solution:*

$$u_n(\cdot,t_k) = \widetilde{\mathcal{E}}_{\Delta,n}(\cdot,t_{k-1}), \quad k = 1, 2, \ldots, \qquad u_n(\cdot,0) = \widehat{u}_n(\cdot,0).$$

*Then, the error in the approximate solution satisfies*

$$\|u_n(\cdot,t_k) - u(\cdot,t_k)\|_{\mathbb{V}} \leq \sum_{j=0}^{k} \|\mathcal{P}_n \mathcal{E}_\Delta\|^{k-j} \varepsilon^{\text{proj}}(t_j). \tag{3.9}$$

**Proof.** Let $e(t_k) := \|u_n(\cdot, t_k) - \widehat{u}(\cdot, t_k)\|_{\mathbb{V}}$. For any $k \geq 1$, we have

$$\begin{aligned} e(t_k) &= \|\mathcal{E}_{\Delta,n} u_n(\cdot, t_{k-1}) - \mathcal{P}_n u(\cdot, t_k)\|_{\mathbb{V}} \\ &= \|\mathcal{P}_n \mathcal{E}_\Delta u_n(\cdot, t_{k-1}) - \mathcal{P}_n \mathcal{E}_\Delta u(\cdot, t_{k-1})\|_{\mathbb{V}} \\ &\leq \|\mathcal{P}_n \mathcal{E}_\Delta u_n(\cdot, t_{k-1}) - \mathcal{P}_n \mathcal{E}_\Delta \widehat{u}_n(\cdot, t_{k-1})\|_{\mathbb{V}} \\ &\quad + \|\mathcal{P}_n \mathcal{E}_\Delta \widehat{u}_n(\cdot, t_{k-1}) - \mathcal{P}_n \mathcal{E}_\Delta u(\cdot, t_{k-1})\|_{\mathbb{V}} \\ &\leq \|\mathcal{P}_n \mathcal{E}_\Delta\| e(t_{k-1}) + \|\mathcal{P}_n \mathcal{E}_\Delta\| \varepsilon^{\mathtt{proj}}(t_{k-1}). \end{aligned}$$

By recursively applying the above inequality and using $e(t_0) = 0$, we obtain

$$\|u_n(\cdot, t_k) - \widehat{u}(\cdot, t_k)\|_{\mathbb{V}} \leq \sum_{j=0}^{k-1} \|\mathcal{P}_n \mathcal{E}_\Delta\|^{k-j} \varepsilon^{\mathtt{proj}}(t_j).$$

The estimate (3.9) is further obtained by using

$$\|u_n(\cdot, t_k) - u(\cdot, t_k)\|_{\mathbb{V}} \leq \|u_n(\cdot, t_k) - \widehat{u}(\cdot, t_k)\|_{\mathbb{V}} + \|\widehat{u}(\cdot, t_k) - u(\cdot, t_k)\|_{\mathbb{V}} = e(t_k) + \varepsilon^{\mathtt{proj}}(t_k). \quad \square$$

We now define a linear mapping

$$\Pi : \mathbb{R}^n \to \mathbb{V}_n, \qquad \Pi \mathbf{v} = \langle \mathbf{v}, \mathbf{\Phi}(x) \rangle, \qquad \mathbf{v} \in \mathbb{R}^n, \tag{3.10}$$

which is a bijective mapping whose inverse exists. Subsequently, $\Pi : \mathbb{R}^n \to \mathbb{V}_n$ is an isomorphism. This mapping defines a unique correspondence between a solution in $\mathbb{V}_n$ and its modal expansion coefficients in $\mathbb{R}^n$.

**Proposition 3.2.** *Let $\mathcal{E}_{\Delta,n}$ be a finite dimensional evolution operator for $u_n \in \mathbb{V}_n$, as defined in (3.6), and $\mathbf{v} \in \mathbb{R}^n$ be its coefficient vector as in (3.4), then $\mathbf{v}$ follows an evolution operator*

$$\mathcal{M}_{\Delta,n} : \mathbb{R}^n \to \mathbb{R}^n, \qquad \mathcal{M}_{\Delta,n} \mathbf{v}(t) = \mathbf{v}(t + \Delta), \tag{3.11}$$

*and*

$$\mathcal{M}_{\Delta,n} = \Pi^{-1} \mathcal{E}_{\Delta,n} \Pi, \tag{3.12}$$

*where $\Pi$ is the linear mapping defined in (3.10). Furthermore, if $\mathcal{E}_{\Delta,n}$ is defined as in (3.7), then*

$$\mathcal{M}_{\Delta,n} = \Pi^{-1} \mathcal{P}_n \mathcal{E}_\Delta \Pi =: \widetilde{\mathcal{M}}_{\Delta,n}. \tag{3.13}$$

The proof is a trivial exercise of substituting (3.10) into (3.6).

Therefore, we have transformed the learning of the infinite dimensional evolution operator $\mathcal{E}_\Delta$ (3.1) for the true solution $u \in \mathbb{V}$ to the learning of its finite dimensional approximation $\mathcal{E}_{\Delta,n}$ (3.6) for the approximate solution $u_n \in \mathbb{V}_n$, which is equivalent to the learning of the evolution operator $\mathcal{M}_{\Delta,n}$ (3.11) for its expansion coefficient $\mathbf{v} \in \mathbb{R}^n$.

## 4. Numerical approach

In this section, we discuss the detail of our PDE learning algorithm. The general procedure consists of the following steps:

- Choose a basis for the finite dimensional space $\mathbb{V}_n$ (3.3) and a corresponding projection operator (3.5).
- Apply the projection operation and project the snapshot data (2.2) to $\mathbb{V}_n$ (3.3) to obtain training data in modal space.
- Choose an appropriate deep neural network structure to approximate the finite dimensional evolution operator $\mathcal{M}_{\Delta,n}$ (3.11) and conduct the network training.
- Conduct numerical prediction of the system by advancing the learned neural network model for the evolution operator.

We remark that this is a fairly general procedure. One is certainly not confined to using neural network. Other approximation methods can also be applied to model the evolution operator. However, modern neural networks, along with their advanced training algorithms, are able to handle relatively high dimensional inputs. This feature makes them more suitable to PDE learning.

### 4.1. Basis selection and data set construction

The choice of basis functions is fairly straightforward – any basis suitable for spatial approximation of the solution data can be used. These include piecewise polynomials, typically used in finite difference or finite elements methods, or orthogonal polynomials used in spectral methods, etc. The basis should also be sufficiently fine to resolve the structure of the true solution.

Once the basis functions are selected, one proceeds to employ a suitable projection operator $\mathcal{P}_n : \mathbb{V} \to \mathbb{V}_n$ to represent the solution in the finite dimensional form (3.5). This can be accomplished via piecewise interpolation, as commonly used in finite elements and finite difference methods, or orthogonal projection, which is often used in spectral methods.

One of the key ingredients for learning the finite dimensional evolution operator $\mathcal{M}_{\Delta,n}$ (3.11) is to acquire modal vector data in pairs, whose components are separated by a time lag. That is, let $J \geq 1$ be the total number of solution data pairs. Then, we define the $j$-th data pair as

$$(\mathbf{v}_j(0), \mathbf{v}_j(\Delta_j)), \qquad j = 1, \ldots, J, \tag{4.1}$$

where $\Delta_j > 0$ is the time lag. Note again that for the autonomous systems considered in this paper, the time difference $\Delta_j$ is the only relevant variable. Hereafter, we will assume, without loss of generality, $\Delta_j = \Delta$ is a constant.

#### 4.1.1. Data pairing via snapshot data projection

When solution snapshots are available in the form of (2.2), we first identify and create pairs of snapshots that are separated by the time lag $\Delta$. That is, we seek to have the data arranged in the following form

$$(u^{(j)}(\cdot, t_{k_j}), u^{(j)}(\cdot, t_{k_j} + \Delta)), \qquad j = 1, \ldots, J, \tag{4.2}$$

where $J \geq 1$ is the total number of pairs. Note that some (or, even all) of the pairs may be originated from the same "trajectory". That is, they are obtained from the original data snapshots (2.2) with the same initial condition. For more effective equation recovery, it is strongly preferred that the data pairs are originated from different trajectories with a large number of different initial conditions [39].

Once the snapshot data pairs are constructed, we proceed to project them onto the finite dimensional space $\mathbb{V}_n$ by applying the projection operator (3.5). This then produces data pairs for the modal expansion coefficients (4.1), where

$$\mathbf{v}_j(0) = \Pi^{-1}\mathcal{P}_n u^{(j)}(\cdot, t_{k_j}), \qquad \mathbf{v}_j(\Delta) = \Pi^{-1}\mathcal{P}_n u^{(j)}(\cdot, t_{k_j} + \Delta).$$

Learning the finite dimensional evolution operator is then conducted in the modal space in search for $\mathcal{M}_{\Delta,n}$ (3.11). This is accomplished by formally solving the following minimization problem:

$$\mathcal{M}_{\Delta,n} = \underset{\mathcal{N}_\Delta : \mathbb{R}^n \to \mathbb{R}^n}{\operatorname{argmin}} \frac{1}{J} \sum_{j=1}^{J} \| \mathcal{N}_\Delta \mathbf{v}_j(0) - \mathbf{v}_j(\Delta) \|_2^2. \tag{4.3}$$

This corresponds to finding the operator $\mathcal{E}_{\Delta,n}$ (3.6) by formally solving

$$\mathcal{E}_{\Delta,n} := \underset{E_\Delta : \mathbb{V}_n \to \mathbb{V}_n}{\operatorname{argmin}} \frac{1}{J} \sum_{j=1}^{J} \| E_\Delta \widehat{u}^{(j)}(\cdot, t_{k_j}) - \widehat{u}^{(j)}(\cdot, t_{k_j} + \Delta) \|_{\mathbb{V}}^2, \tag{4.4}$$

where $\widehat{u}^{(j)} = \mathcal{P}_n u^{(j)}$.

#### 4.1.2. Data pairing via sampling in modal space

When data collection procedure is in a controlled environment, it is then possible to directly sample in the modal space to generate the training data pairs. One example of such a case is when the unknown PDE is controlled by a black-box simulation software or a device. This would allow one to possibly generate arbitrary "initial" conditions and then collect their states at a later time. In this case, the training data pairs (4.1) can be generated as follows.

- Sample $J$ points $\{\mathbf{v}_j(0)\}_{j=1}^J$ over a domain $D \subset \mathbb{R}^n$, where $D$ is a region in which one is interested in the solution behavior. The sampling can be conducted randomly or by using other proper sampling techniques.
- For each $j = 1, \ldots, J$, construct initial solution

$$u^{(j)}(x, 0) = \Pi \mathbf{v}_j(0) = \langle \mathbf{v}_j(0), \boldsymbol{\Phi}(x) \rangle. \tag{4.5}$$

  Then, march forward in time for the time lag $\Delta$, by using the underlying black-box simulation code or device, and obtain the solution snapshots $u^{(j)}(x, \Delta)$. Conduct the projection operation (3.5) to obtain

$$\mathbf{v}_j(\Delta) = \Pi^{-1}\mathcal{P}_n u^{(j)}(x, \Delta). \tag{4.6}$$

The modal expansion coefficients data generated in this manner then satisfy, for each $j = 1, \ldots, J$,

$$\mathbf{v}_j(\Delta) = \Pi^{-1}\mathcal{P}_n\mathcal{E}_\Delta u^{(j)}(x,0) = \Pi^{-1}\mathcal{P}_n\mathcal{E}_\Delta \Pi \mathbf{v}_j(0) = \widetilde{\mathcal{M}}_{\Delta,n}\mathbf{v}_j(0),$$

where $\widetilde{\mathcal{M}}_{\Delta,n}$ is defined in (3.13) and corresponds to the finite dimensional evolution operator defined in (3.7).

We remark that this procedure also generates solution pairs along the way, i.e,

$$(u^{(j)}(\cdot,0), u^{(j)}(\cdot,\Delta)), \qquad u^{(j)}(\cdot,0) \in \mathbb{V}_n, \qquad j = 1, \ldots, J.$$

Note that here $u^{(j)}(\cdot,0) \in \mathbb{V}_n$ due to (4.5) via the direct sampling in the modal space. These pairs are then different from those in (4.2), whose components are not necessarily in $\mathbb{V}_n$.

### 4.2. Neural network modeling of evolution operator

Once the training data set (4.1) becomes available, one can proceed to learn the unknown governing equation. This is achieved by learning the finite dimensional evolution operator relating the solution coefficients in the data pairs. As shown in Proposition 3.2, finding the finite dimensional evolution operator of the unknown PDE (3.6) is equivalent to find the evolution operator (3.12) for the modal expansion coefficient vectors. Suppose the solution coefficient vectors formally follow an unknown autonomous system, $d\mathbf{v}/dt = \mathbf{f}(\mathbf{v})$, we then have, by using mean-value theorem,

$$\begin{aligned} \mathbf{v}(\Delta) &= \mathcal{M}_{\Delta,n}(\mathbf{v}(0)) = \mathbf{v}(0) + \int_0^\Delta \mathbf{f}(\mathbf{v}(t))dt \\ &= \mathbf{v}(0) + \Delta \cdot \mathbf{f}(\mathcal{M}_{\tau,n}\mathbf{v}(0)), \quad 0 \le \tau \le \Delta, \end{aligned} \tag{4.7}$$

where $\mathcal{M}_{\Delta,n}$ is the evolution operator defined in (3.12) and $\tau$ depends on $\Delta$ and the form of the operator $\mathcal{L}$ in (2.1). Furthermore, if $\mathcal{M}_{\Delta,n}$ takes the form of (3.13), we then have

$$\begin{aligned} \mathbf{v}(\Delta) &= \widetilde{\mathcal{M}}_{\Delta,n}(\mathbf{v}(0)) = \Pi^{-1}\mathcal{P}_n\mathcal{E}_\Delta\Pi\mathbf{v}(0) \\ &= \Pi^{-1}\mathcal{P}_n\mathcal{E}_\Delta\langle \mathbf{v}(0), \mathbf{\Phi}(x)\rangle \\ &= \Pi^{-1}\mathcal{P}_n\left(\langle \mathbf{v}(0), \mathbf{\Phi}(x)\rangle + \int_0^\Delta \mathcal{L}(\mathcal{E}_t\Pi\mathbf{v}(0))dt\right) \\ &= \Pi^{-1}\mathcal{P}_n\langle \mathbf{v}(0), \mathbf{\Phi}(x)\rangle + \Delta\cdot\Pi^{-1}\mathcal{P}_n\mathcal{L}(\mathcal{E}_\tau\Pi\mathbf{v}(0)) \\ &= \mathbf{v}(0) + \Delta\cdot\Pi^{-1}\mathcal{P}_n\mathcal{L}(\mathcal{E}_\tau\Pi\mathbf{v}(0)), \quad 0 \le \tau \le \Delta. \end{aligned} \tag{4.8}$$

These relations suggest that, when the time lag $\Delta$ is small, it is natural to adopt the residue network (ResNet) [8] to model the evolution operator. This approach was presented and systematically studied in [18], for learning of unknown dynamical systems from data. The block ResNet structure for equation learning from [18] takes the following form

$$\mathbf{v}(\Delta) = \boldsymbol{\mathcal{N}}(\mathbf{v}(0);\Theta), \tag{4.9}$$

where $\boldsymbol{\mathcal{N}}$ denotes the nonlinear operator defined by the underlying neural network with parameter set $\Theta$. For block ResNet with $K \ge 1$ ResNet blocks, the operator

$$\boldsymbol{\mathcal{N}} = \big(\mathbf{I} + \mathbf{N}(\bullet;\Theta_{K-1})\big)\circ\cdots\circ\big(\mathbf{I} + \mathbf{N}(\bullet;\Theta_0)\big),$$

with $\mathbf{N}$ is standard fully connected feedforward neural network, and $\Theta = \{\Theta_i\}_{0\le i\le K-1}$ are the parameters (weights and biases) in each ResNet block. (See [18] for more details.) The parameters are determined by minimizing the loss function (4.3), i.e.,

$$L(\Theta) = \frac{1}{J}\sum_{j=1}^{J}\big\|\boldsymbol{\mathcal{N}}(\mathbf{v}_j(0);\Theta) - \mathbf{v}_j(\Delta)\big\|_2^2, \tag{4.10}$$

where $\|\cdot\|_2$ denotes vector 2-norm. Let $\Theta_*$ be the trained parameters, after satisfactory minimization of the loss function. A learned model is then constructed in the form of $\boldsymbol{\mathcal{N}}(\cdot;\Theta_*)$, which provides an approximation to the evolution operator (3.11),

$$\boldsymbol{\mathcal{N}}(\cdot,\Theta_*) \approx \mathcal{M}_{\Delta,n}. \tag{4.11}$$

The trained network model can then be used to provide prediction of the system (2.1). For an arbitrary initial condition $u(x,0) \in \mathbb{V}$, we first conduct its projection (3.5) and obtain

$$\widehat{\mathbf{v}}(0) = \Pi^{-1}\mathcal{P}_n u(\cdot, 0), \tag{4.12}$$

where $\Pi$ is the bijective mapping defined in (3.10). We then iteratively apply the neural network model (4.11) to obtain approximate solutions for the modal expansion coefficients at time instances $t_k = k\Delta$,

$$\begin{cases} \widetilde{\mathbf{v}}(0) = \widehat{\mathbf{v}}(0), \\ \widetilde{\mathbf{v}}(t_{k+1}) = \boldsymbol{\mathcal{N}}(\widetilde{\mathbf{v}}(t_k); \Theta_*), \qquad k = 0, \ldots. \end{cases} \tag{4.13}$$

The predicted solution fields $\widetilde{u}_n(x, t_k)$ are then obtained by

$$\widetilde{u}_n(x, t_k) = \Pi\widetilde{\mathbf{v}}(t_k) = \langle \widetilde{\mathbf{v}}(t_k), \boldsymbol{\Phi}(x) \rangle, \qquad k = 1, \ldots. \tag{4.14}$$

### 4.3. Error analysis

We now derive an error bound for the predicted solution (4.14) of our neural network model, when the finite dimensional evolution operator (3.13) is the case, i.e., $\mathcal{M}_{\Delta,n} = \widetilde{\mathcal{M}}_{\Delta,n} = \Pi^{-1}\mathcal{P}_n\mathcal{E}_\Delta\Pi$. For notational convenience, we assume that the basis functions $\{\phi_j(x)\}_{j=1}^n$ of $\mathbb{V}_n$ are orthonormal. (Note that non-orthogonal basis can always be orthogonalized via Gram-Schmidt procedure.) Then, the bijective mapping $\Pi : \mathbb{R}^n \to \mathbb{V}_n$ defined in (3.10) is an isometric isomorphism and satisfies

$$\|\Pi\mathbf{v}\|_{\mathbb{V}} = \|\mathbf{v}\|_2, \qquad \mathbf{v} \in \mathbb{R}^n.$$

Also, since it is well known that neural networks are universal approximator for a general class of functions, we assume that the training error in (4.11) is bounded. We then state the following result.

**Theorem 4.1.** *Assume the evolution operator $\mathcal{E}_\Delta$ (3.1) of the underlying PDE is bounded. Also, assume the trained neural network model $\boldsymbol{\mathcal{N}} = \boldsymbol{\mathcal{N}}(\cdot, \Theta_*)$ is bounded, and its approximation error (4.11) is bounded and denote $\varepsilon^{\text{DNN}} := \|\boldsymbol{\mathcal{N}} - \Pi^{-1}\mathcal{P}_n\mathcal{E}_\Delta\Pi\| < +\infty$. Let $t_k = k\Delta$, $k = 0, 1, \ldots$, and let $\varepsilon^{\text{proj}}(t_k) := \|u(x, t_k) - \mathcal{P}_n u(x, t_k)\|_{\mathbb{V}}$ be the projection error of the exact solution at $t_k$. Then the following error bounds hold:*

$$\|\widetilde{\mathbf{v}}(t_k) - \widehat{\mathbf{v}}(t_k)\|_2 \le \sum_{j=0}^{k-1} \|\boldsymbol{\mathcal{N}}\|^{k-1-j} \Big( \varepsilon^{\text{DNN}} \|\widehat{\mathbf{v}}(t_j)\|_2 + \varepsilon^{\text{proj}}(t_j) \|\mathcal{P}_n\mathcal{E}_\Delta\| \Big), \tag{4.15}$$

*where $\widehat{\mathbf{v}} = \Pi^{-1}\widehat{u}$ is the modal expansion coefficient vector of the projected exact solution (3.5) and $\widetilde{\mathbf{v}}$ is the coefficient vector predicted by the neural network model (4.13), and*

$$\|\widetilde{u}_n(\cdot, t_k) - u(\cdot, t_k)\|_{\mathbb{V}} \le \varepsilon^{\text{proj}}(t_k) + \sum_{j=0}^{k-1} \|\boldsymbol{\mathcal{N}}\|^{k-1-j} \Big( \varepsilon^{\text{DNN}} \|\widehat{\mathbf{v}}(t_j)\|_2 + \varepsilon^{\text{proj}}(t_j) \|\mathcal{P}_n\mathcal{E}_\Delta\| \Big), \tag{4.16}$$

*where $\widetilde{u}$ is the solution state predicted by the trained neural network model (4.14).*

**Proof.** Let $e(t_k) := \|\widetilde{\mathbf{v}}(t_k) - \widehat{\mathbf{v}}(t_k)\|_2$. Note that, for $k = 1, \ldots,$

$$\widehat{\mathbf{v}}(t_k) = \Pi^{-1}\widehat{u}_n(\cdot, t_k) = \Pi^{-1}\mathcal{P}_n u(\cdot, t_k) = \Pi^{-1}\mathcal{P}_n\mathcal{E}_\Delta u(\cdot, t_{k-1}).$$

We then have

$$\begin{aligned} e(t_k) &= \|\widetilde{\mathbf{v}}(t_k) - \Pi^{-1}\mathcal{P}_n\mathcal{E}_\Delta u(\cdot, t_{k-1})\|_2 \\ &\le \|\widetilde{\mathbf{v}}(t_k) - \Pi^{-1}\mathcal{P}_n\mathcal{E}_\Delta\widehat{u}_n(\cdot, t_{k-1})\|_2 \\ &\quad + \|\Pi^{-1}\mathcal{P}_n\mathcal{E}_\Delta\widehat{u}_n(\cdot, t_{k-1}) - \Pi^{-1}\mathcal{P}_n\mathcal{E}_\Delta u(\cdot, t_{k-1})\|_2 \\ &= \|\boldsymbol{\mathcal{N}}(\widetilde{\mathbf{v}}(t_{k-1}); \Theta_*) - \Pi^{-1}\mathcal{P}_n\mathcal{E}_\Delta\Pi\mathbf{v}(t_{k-1})\|_2 \\ &\quad + \|\mathcal{P}_n\mathcal{E}_\Delta\widehat{u}_n(\cdot, t_{k-1}) - \mathcal{P}_n\mathcal{E}_\Delta u(\cdot, t_{k-1})\|_{\mathbb{V}} \\ &\le \|\boldsymbol{\mathcal{N}}(\widetilde{\mathbf{v}}(t_{k-1}); \Theta_*) - \boldsymbol{\mathcal{N}}(\widehat{\mathbf{v}}(t_{k-1}); \Theta_*)\|_2 \\ &\quad + \|\boldsymbol{\mathcal{N}}(\widehat{\mathbf{v}}(t_{k-1}); \Theta_*) - \Pi^{-1}\mathcal{P}_n\mathcal{E}_\Delta\Pi\widehat{\mathbf{v}}(t_{k-1})\|_2 \\ &\quad + \|\mathcal{P}_n\mathcal{E}_\Delta\| \|\widehat{u}_n(\cdot, t_{k-1}) - u(\cdot, t_{k-1})\|_{\mathbb{V}} \\ &\le \|\boldsymbol{\mathcal{N}}\| e(t_{k-1}) + \varepsilon^{\text{DNN}} \|\widehat{\mathbf{v}}(t_{k-1})\|_2 + \|\mathcal{P}_n\mathcal{E}_\Delta\| \varepsilon^{\text{proj}}(t_{k-1}). \end{aligned}$$

By recursively applying the above inequality and using $e(t_0) = 0$, we obtain (4.15). The proof is then complete by using

$$\begin{aligned}
&\|\widetilde{u}_n(\cdot, t_k) - u(\cdot, t_k)\|_{\mathbb{V}} \\
&\quad \le \|\widetilde{u}_n(\cdot, t_k) - \mathcal{P}_n u(\cdot, t_k)\|_{\mathbb{V}} + \|\mathcal{P}_n u(\cdot, t_k) - u(\cdot, t_k)\|_{\mathbb{V}} \\
&\quad = \|\widetilde{\mathbf{v}}(t_k) - \widehat{\mathbf{v}}(t_k)\|_2 + \varepsilon^{\mathtt{proj}}(t_k) = e(t_k) + \varepsilon^{\mathtt{proj}}(t_k). \quad \square
\end{aligned}$$

Theorem 4.1 indicates that the prediction error of the network model is affected by the approximation error of the neural network and the projection error, which is determined by the approximation space $\mathbb{V}_n$ and the regularity of the solution.

## 5. Numerical examples

In this section, we present numerical examples to verify the properties of the proposed method. For benchmarking purpose, the true governing PDEs are known in all of them examples. However, we use the true governing equations only to generate training data, particularly by following the procedure in Section 4.1.2. Our proposed learning method is then applied to these synthetic data and produces a trained neural network model for the underlying PDE. We will then use the neural network model to conduct numerical predictions of the solution and compare them against the reference solution produced by the governing equations. Numerical errors will be reported, in term of relative errors between the neural network prediction solutions and the reference solutions.

The governing equations considered in this section include: linear advection equation, linear diffusion equation, nonlinear viscous Burgers' equation, nonlinear inviscid Burgers' equation, the last of which produces shocks. We primarily focus on one dimension in physical space with noiseless data, in order to conduct detailed and thorough examination of the solution behavior. Data noises are introduced in one example to demonstrate the applicability of the method. A two-dimensional advection-diffusion problem is also presented to demonstrate the applicability of the method to multiple dimensions.

In all examples, we employ global orthogonal polynomials as the basis functions to define the finite dimensional space for training. For benchmarking purpose, all training data are generated in modal space, as described in Section 4.1.2. We also rather arbitrarily impose a decay condition in certain examples to ensure the higher modes are smaller, compared to the lower modes. This effectively poses a smoothness condition on the training data.

All of our network models are trained via minimizing the loss function (4.3) and by using the open-source Tensorflow library [1]. The training data set is divided into mini-batches of size 10, and the learning rate is taken as 0.001. The network weights are initialized randomly from Gaussian distributions and biases are initialized to zeros. Upon successful training, the neural network models are then marched forward in time with the $\Delta$ time step, as in (4.13). The results, labeled as "prediction", are compared against the reference solutions, labeled as "exact", of the true underlying governing equations.

### 5.1. Example 1: advection equation

We first consider a one-dimensional advection equation with periodic boundary condition:

$$\begin{cases} u_t + u_x = 0, \quad (x, t) \in (0, 2\pi) \in \mathbb{R}^+, \\ u(0, t) = u(2\pi, t), \quad t \in \mathbb{R}^+. \end{cases} \tag{5.1}$$

The finite dimensional approximation space is set as $\mathbb{V}_n = \text{span}\{e^{ikx}, k \le 3\}$, which implies $n = 7$. The time lag $\Delta$ is taken as 0.1. The domain $D$ in the modal space is fixed as $[-0.8, 0.8]$ for $k \le 1$ and $[-0.2, 0.2]$ for $k = 2$, and $[-0.03, 0.03]$ for $k = 3$. By using uniform distribution, we generate 80,000 training data in the modal space. For neural network modeling, we employ the block ResNet structure with two blocks ($K = 2$), each of which contains 3 hidden layers of equal width of 30

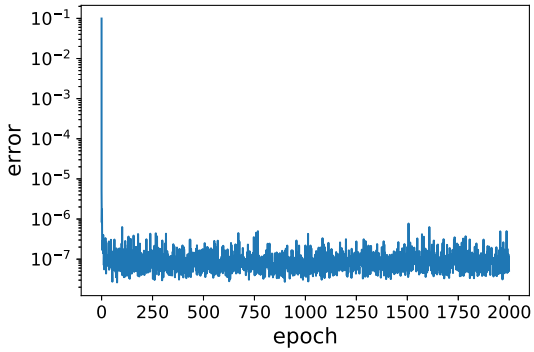


**Fig. 5.1.** Example 1: Training loss history.

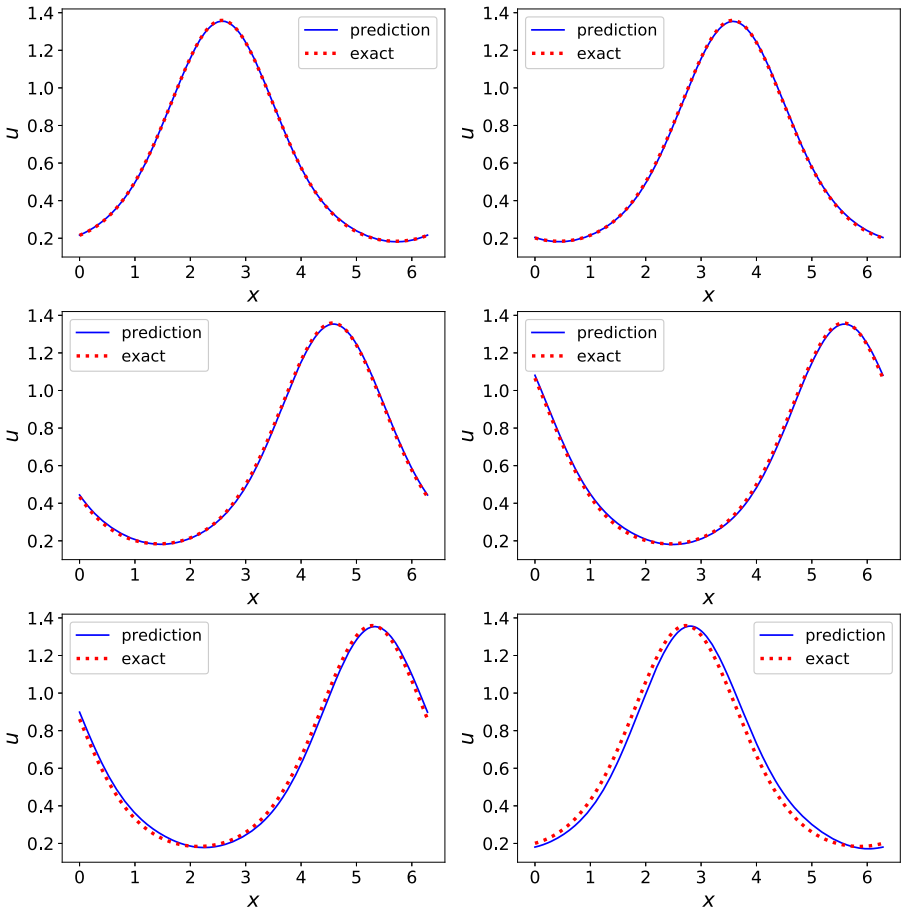


**Fig. 5.2.** Example 1: Comparison of the true solution and the learned model solution at different time. Top-left: $t = 1$; top-right: $t = 2$; middle-left: $t = 3$; middle-right: $t = 4$; bottom-left: $t = 10$; bottom-right: $t = 20$.

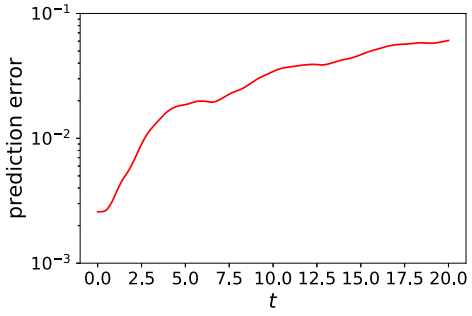


**Fig. 5.3.** Example 1: The evolution of the relative error in the prediction in $l^2$-norm.

neurons. The loss function training history is shown in Fig. 5.1, where the network is trained for up to 2,000 epochs. Convergence can be achieved after about 200 epochs. To validate the model, we employ an initial condition $u_0(x) = \frac{1}{2}\exp(\sin(x))$ and conduct simulations using the trained model, in the form of (4.12)–(4.14), for up to $t = 20$. Note that this particular initial condition, albeit smooth, is fairly representative, as it is not in the approximation space $\mathbb{V}_n$. In Fig. 5.2, the solution prediction of the trained model is plotted at different time instances, along with the true solution for comparison. The relative error in the predicted solution is shown in Fig. 5.3. We observe that the network model produces accurate prediction results for time up to $t = 20$. The error grows over time. This is consistent with the error estimate from Theorem 4.1 and is expected from any numerical time integrator. To further examine the solution property, we also plot in Fig. 5.4 the evolution of the learned expansion coefficients $\hat{v}_j$, $1 \le j \le 7$. For reference purpose, the optimal coefficients obtained by the $L^2_{\mathcal{B}}(\Omega)$ orthogonal projection of the true solution onto $\mathbb{V}_n$ are also plotted. We observe good agreement between the two solutions.

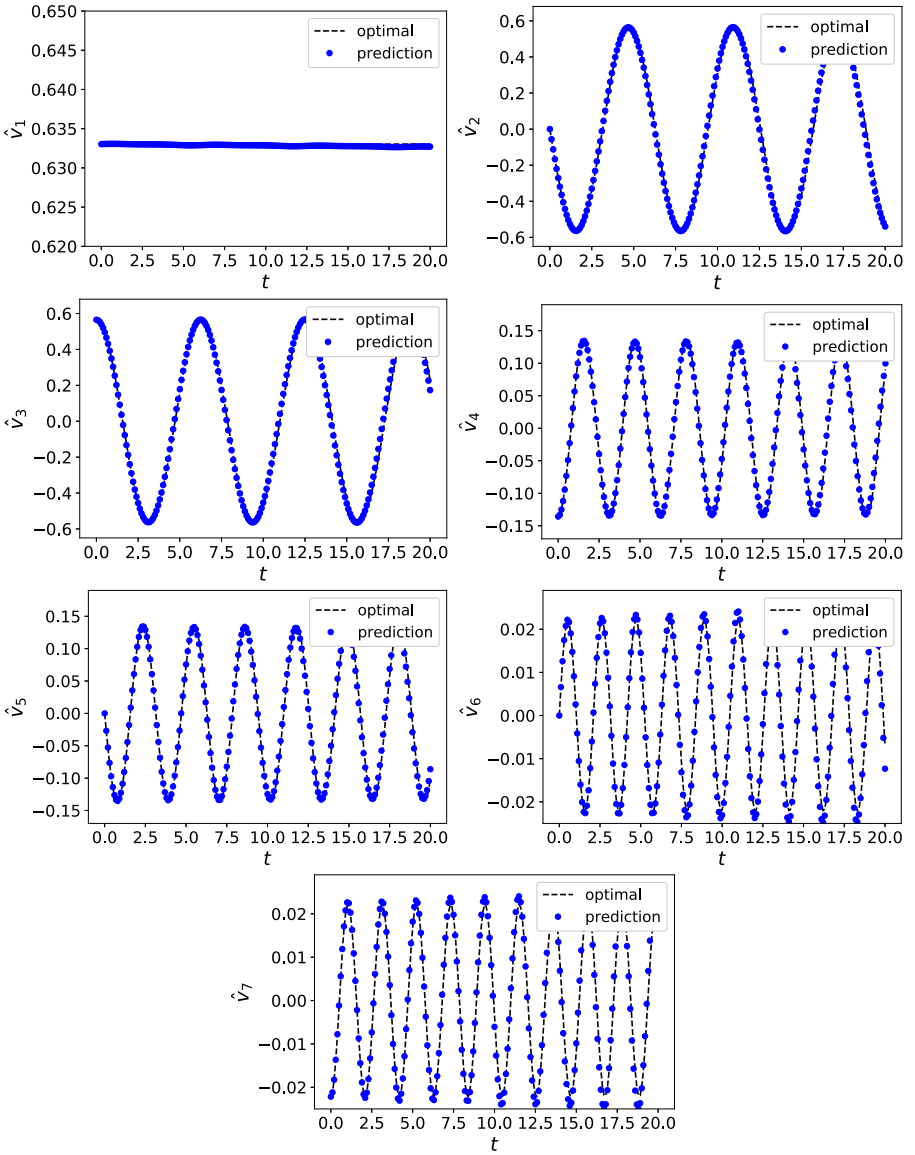


**Fig. 5.4.** Example 1: Evolution of the expansion coefficients for the learned model solution and the projection of the true solution.

### 5.2. Example 2: diffusion equation

We now consider the following diffusion equation with Dirichlet boundary condition:

$$\begin{cases} u_t = \sigma u_{xx}, & (x,t) \in (0,\pi) \times \mathbb{R}^+, \\ u(0,t) = u(\pi,t) = 0, & t \in \mathbb{R}^+. \end{cases} \tag{5.2}$$

The finite dimensional approximation space is chosen as $\mathbb{V}_n = \text{span}\{\sin(jx), 1 \le j \le 5\}$, where $n = 5$. (The symmetry of the problem allows this simplified choice to facilitate the computation and comparison.) The time lag $\Delta$ is taken as 0.1. The domain $D$ in the modal space is taken as $[-1,1] \times [-0.5,0.5] \times [-0.2,0.2] \times [-0.05,0.05] \times [-0.01,0.01]$, from which we sample 30,000 training data. In this example, we employ a single-block ResNet method ($K = 1$) containing 3 hidden layers of equal width of 30 neurons. The training of the network model is conducted for up to 500 epochs, where satisfactory convergence is established, as shown in Fig. 5.5. For validation and accuracy test, we conduct numerical predictions of the trained network model using initial condition

$$u_0(x) = \frac{x}{\pi}\left[5 - \frac{4x}{\pi} - 7\left(\frac{x}{\pi}\right)^2 + 6\left(\frac{x}{\pi}\right)^3\right],$$

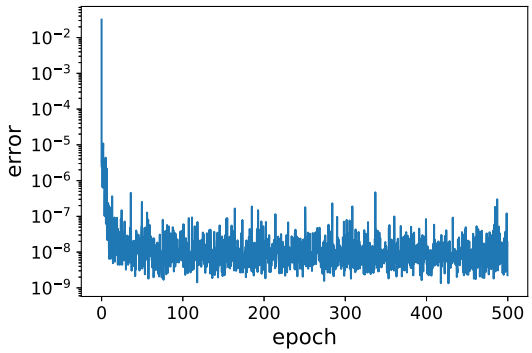


**Fig. 5.5.** Example 2: Training loss history.

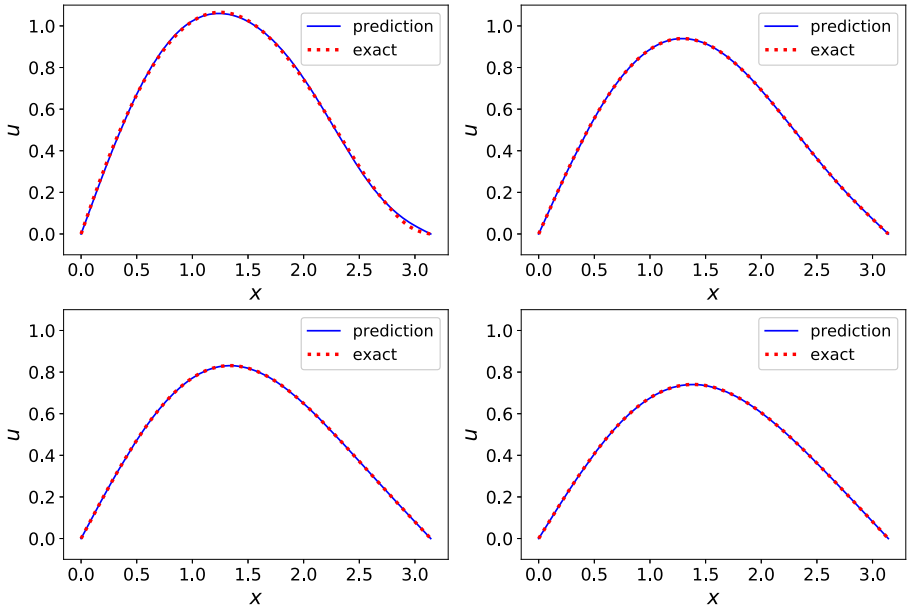


**Fig. 5.6.** Example 2: Comparison of the true solution and the learned model solution at different time. Top-left: $t = 0$; top-right: $t = 1$; bottom-left: $t = 2$; bottom-right: $t = 3$.

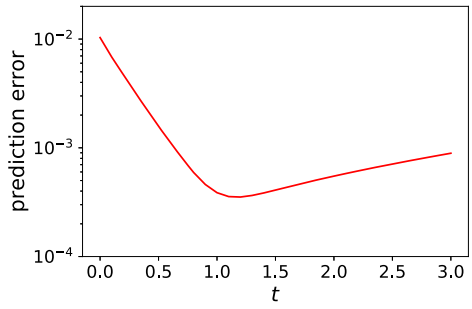


**Fig. 5.7.** Example 2: The evolution of the relative error in the prediction in $l^2$-norm.

for up to time $t = 3$. Fig. 5.6 shows the solution predicted by the trained network model, along with the exact solution of the true equation (5.2). It can be seen that the predicted solution agrees well with the exact solution. The relative error in the numerical prediction is shown in Fig. 5.7, in term of $l^2$-norm. Finally, the evolution of the expansion coefficients is also shown given in Fig. 5.8. We observe that they agree well with the optimal coefficients obtained by projecting the exact solution onto the linear space $\mathbb{V}_n$.

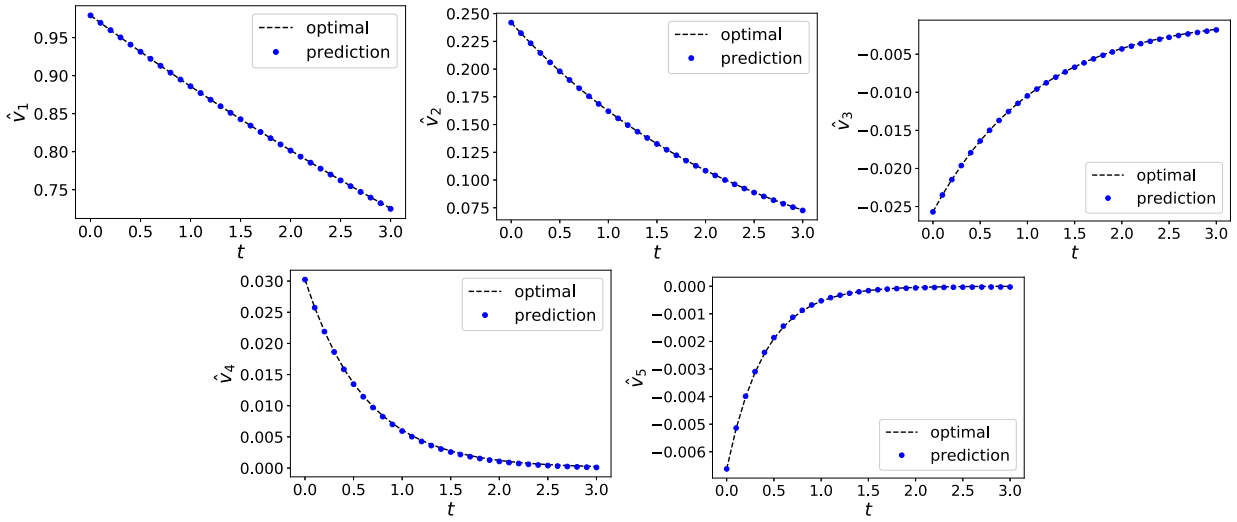


**Fig. 5.8.** Example 2: Evolution of the expansion coefficients for the learn model and the projection of the true solution.

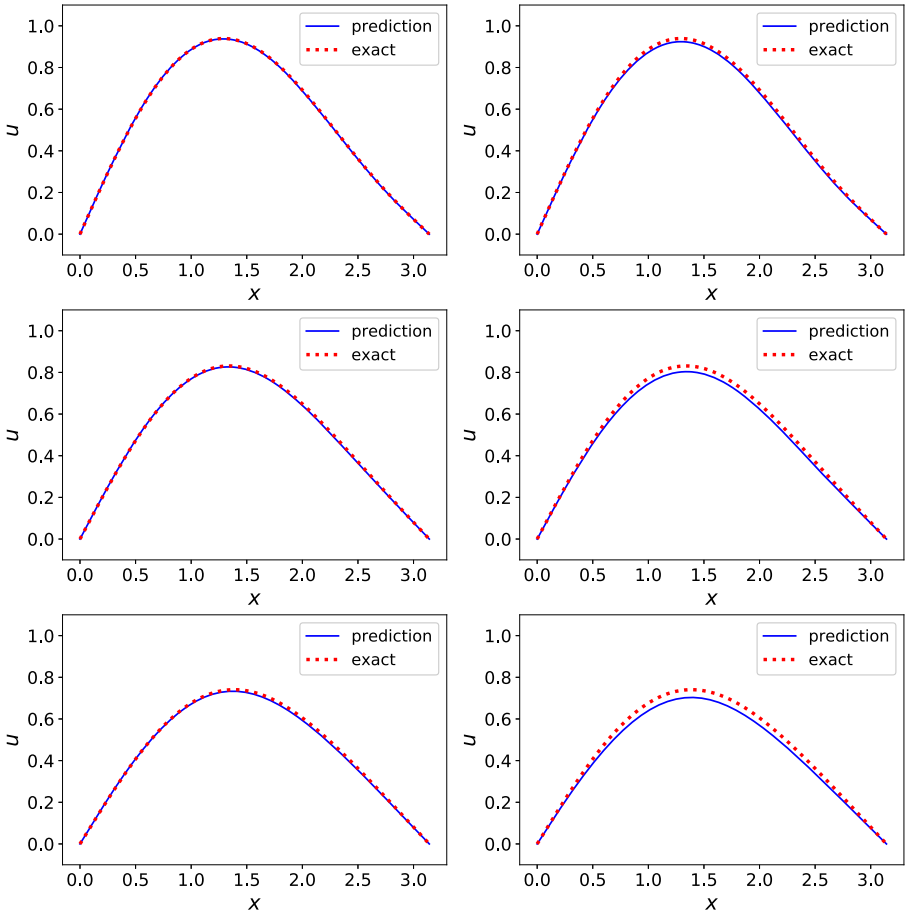


**Fig. 5.9.** Example 2: The solution at different time predicted by the neural network model trained with noisy data. Left: 2% noise; right: 5% noise. From top to bottom: solutions at $t = 1$, $t = 2$ and $t = 3$, respectively.

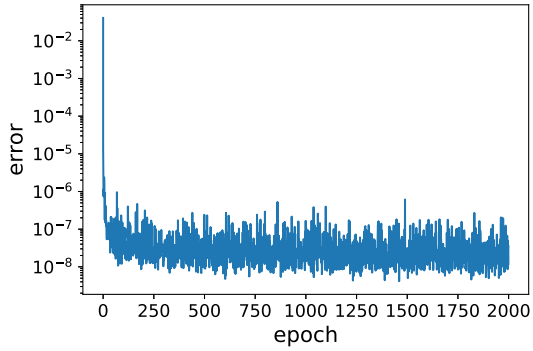


**Fig. 5.10.** Example 3: Training loss history.

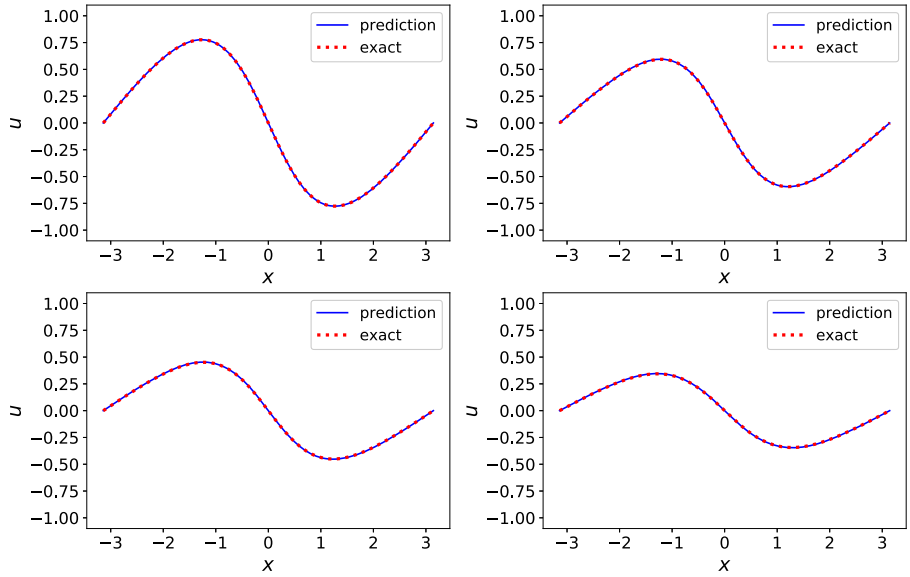


**Fig. 5.11.** Example 3: Comparison of the true solution and the learned model solution at different time. Top-left: $t = 0.5$; top-right: $t = 1$; bottom-left: $t = 1.5$; bottom-right: $t = 2$.

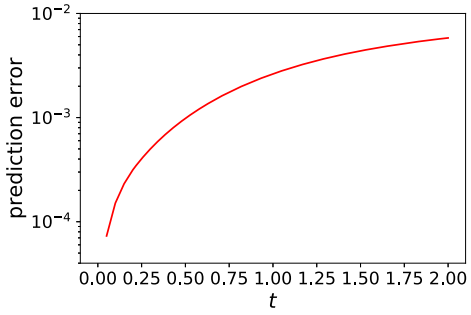


**Fig. 5.12.** Example 3: The evolution of the relative errors in the prediction in $l^2$-norm.

We now consider the case of noisy data. All training data are then perturbed by a multiplicative factor $(1 + \epsilon)$, where $\epsilon \sim [-\eta, \eta]$ follows uniform distribution. We consider two cases of $\eta = 0.02$ and $\eta = 0.05$, which respectively correspond to $\pm 2\%$ and $\pm 5\%$ relative noises in all data. In Fig. 5.9, the numerical solutions produced by the neural network models are presented, after network training of 500 epochs, along with the exact solution. We observe that the predictions of the network model are fairly robust against data noise. At higher level noises in data, the predictive results contain relatively larger numerical errors, as expected.

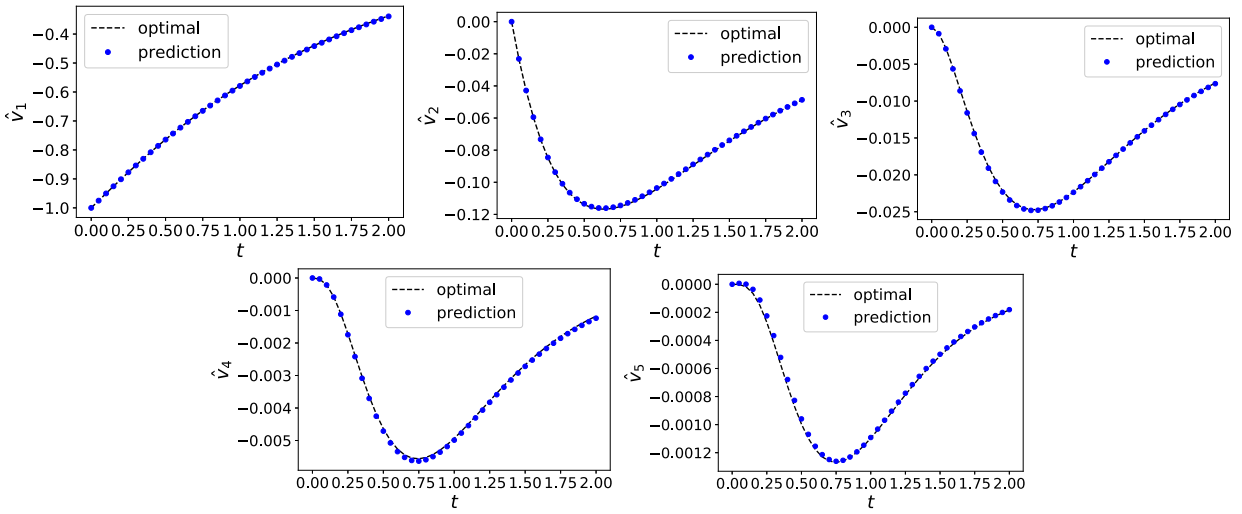


**Fig. 5.13.** Example 3: Evolution of the expansion coefficients for the learned model and the projection of the true solution.

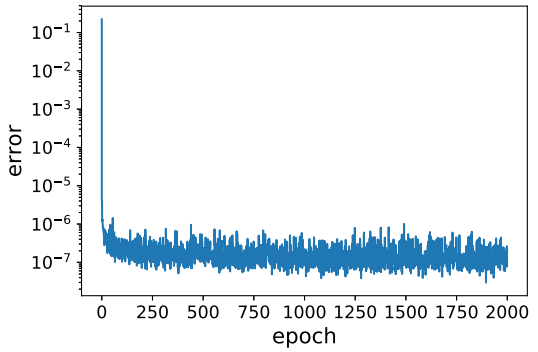


**Fig. 5.14.** Example 3: Training loss history.

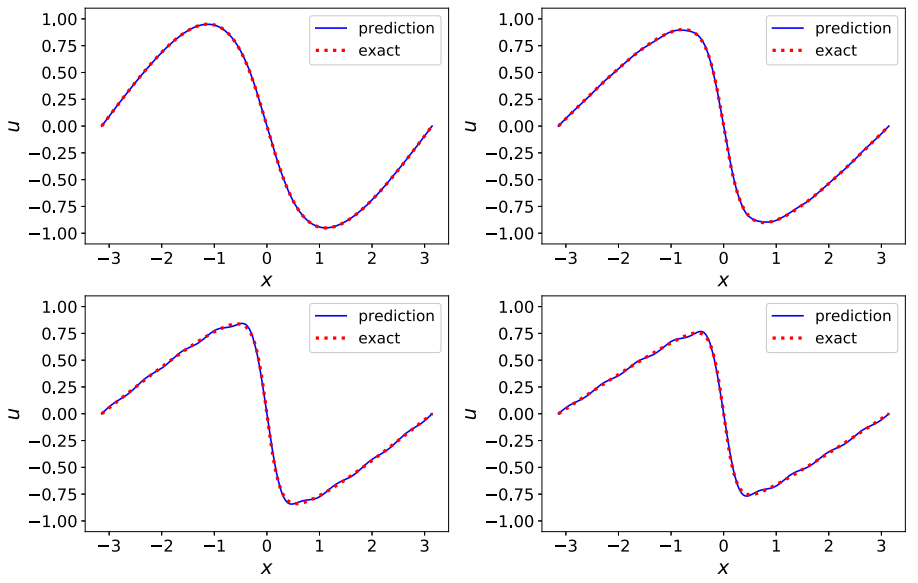


**Fig. 5.15.** Example 3: Comparison of the true solution and the learned model solution at different time. Top-left: $t = 0.5$; top-right: $t = 1$; bottom-left: $t = 1.5$; bottom-right: $t = 2$.

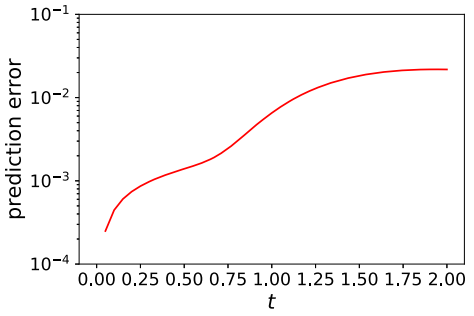


**Fig. 5.16.** Example 3: The evolution of the relative error in prediction in $l^2$-norm.

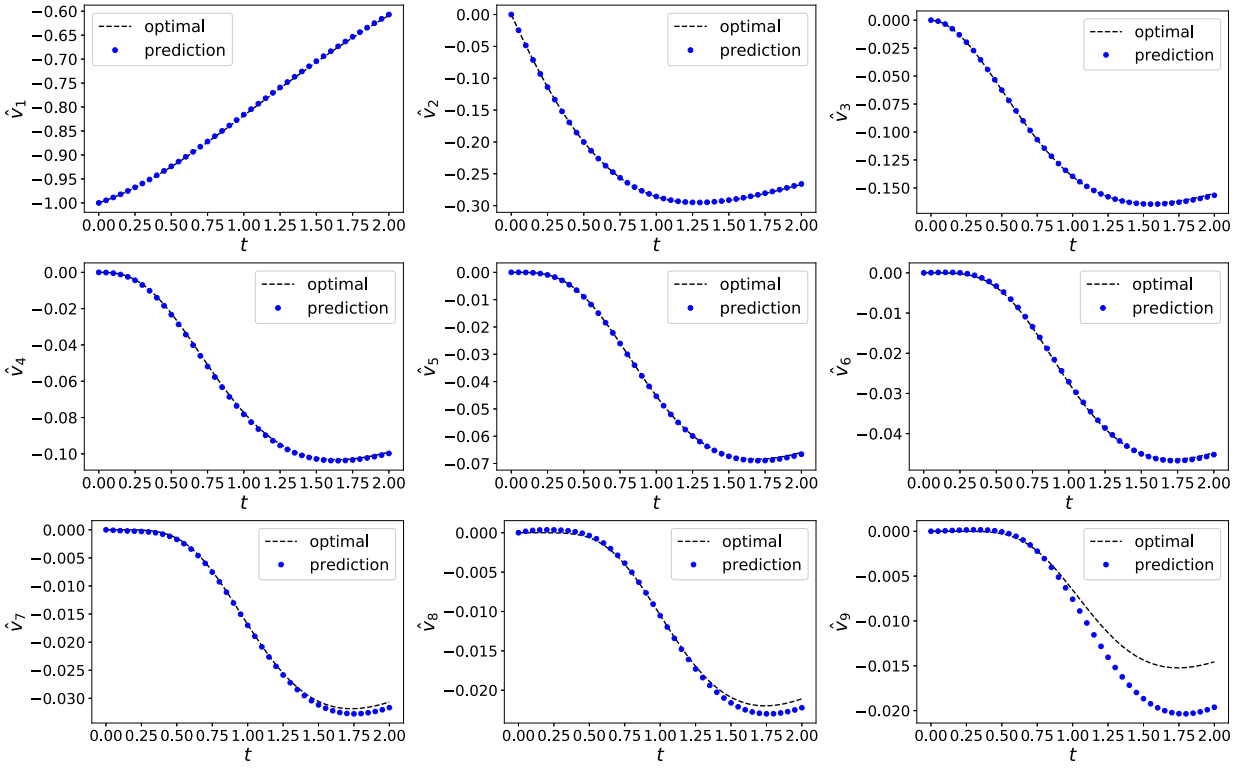


**Fig. 5.17.** Example 3: Evolution of the expansion coefficients for the learned model solution and the projection of the true solution.

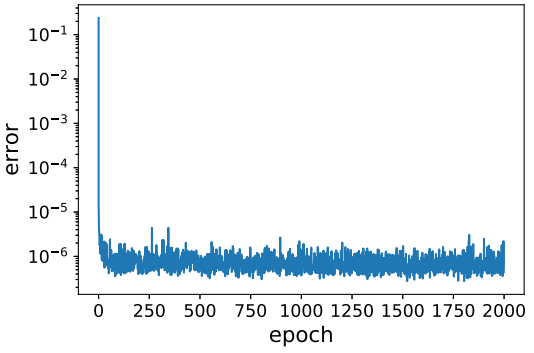


**Fig. 5.18.** Example 4: Training loss history.

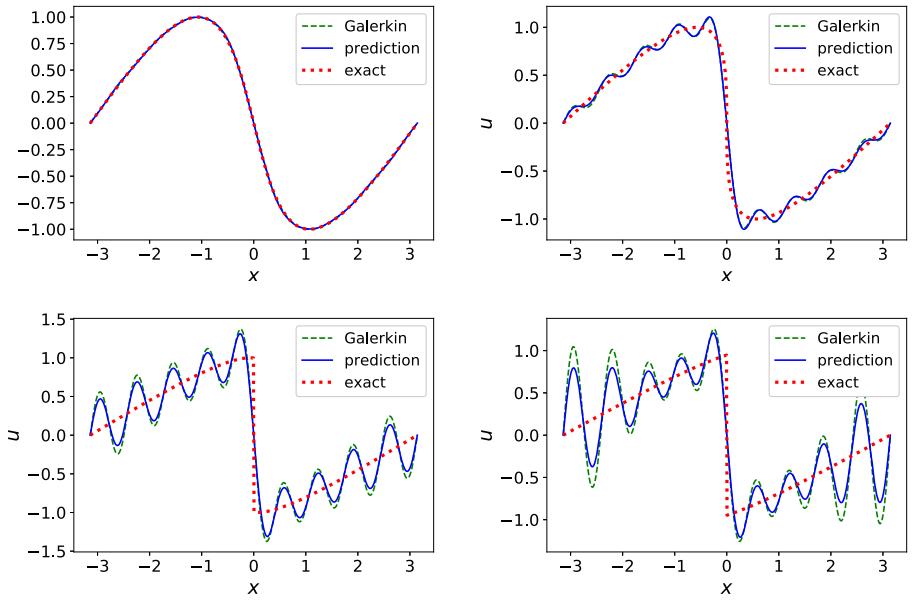


**Fig. 5.19.** Example 4: Comparison of the true solution, the learned model solution and the solution by Galerkin method at different time. Top-left: $t = 0.5$; top-right: $t = 1$; bottom-left: $t = 1.5$; bottom-right: $t = 2$.

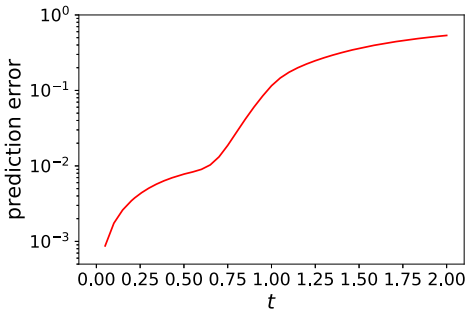


**Fig. 5.20.** Example 4: The evolution of the relative error in the prediction in $l^2$-norm.

### 5.3. Example 3: viscous Burgers' equation

We now consider the viscous Burgers' equation with Dirichlet boundary condition:

$$\begin{cases} u_t + \left(\frac{u^2}{2}\right)_x = \sigma u_{xx}, \quad (x,t) \in (-\pi, \pi) \times \mathbb{R}^+, \\ u(-\pi, t) = u(\pi, t) = 0, \quad t \in \mathbb{R}^+. \end{cases} \tag{5.3}$$

We first consider a modestly large viscosity $\sigma = 0.5$. The approximation space is chosen as $\mathbb{V}_n = \text{span}\{\sin(jx), 1 \le j \le 5\}$ with $n = 5$. The time lag $\Delta$ is fixed at $\Delta = 0.05$. The domain $D$ in the modal space is chosen as $[-1.5, 1.5] \times [-0.2, 0.2] \times [-0.05, 0.05] \times [-0.01, 0.01] \times [-0.002, 0.002]$, from which 100,000 training data are generated. The block ResNet method with two blocks ($K = 2$) is used, where each block contains 3 hidden layers of equal width of 30 neurons. Upon training the network model satisfactorily (see Fig. 5.10 for the training loss history), we validate the trained model for the initial condition

$$u_0(x) = -\sin(x),$$

for time up to $t = 2$. In Fig. 5.11, we compare the predicted solution against the exact solution at different time. The error of the prediction is computed and displayed in Fig. 5.12. We observe that the network model produces accurate prediction results. The learned expansion coefficients are shown in Fig. 5.13 and agree well with the optimal coefficients given by orthogonal projection of the exact solution.

We then consider a smaller viscosity $\sigma = 0.1$. The approximation space is chosen to be relatively larger as $\mathbb{V}_n = \text{span}\{\sin(jx), 1 \le j \le 9\}$ with $n = 9$. The time lag $\Delta$ is taken as 0.05. The domain $D$ in the modal space is taken as

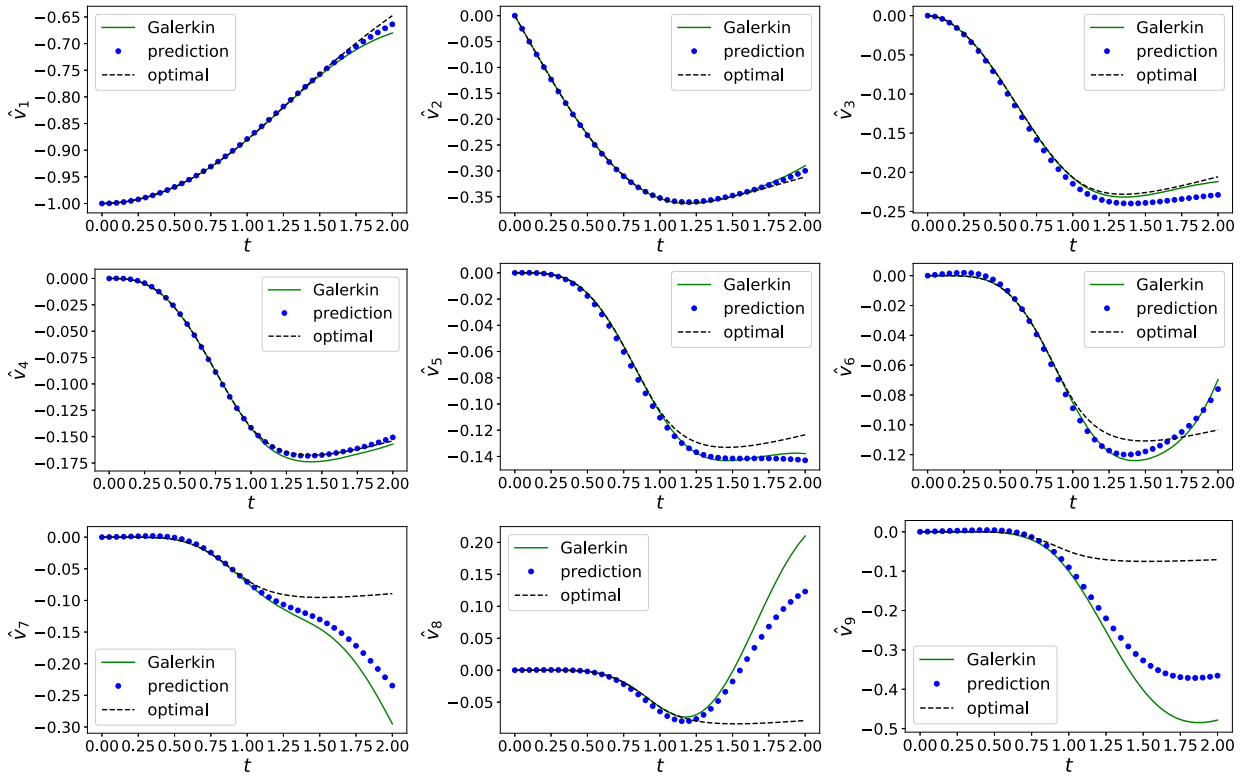


**Fig. 5.21.** Example 4: Evolution of the expansion coefficients for the learned model solution and the projection of the true solution.

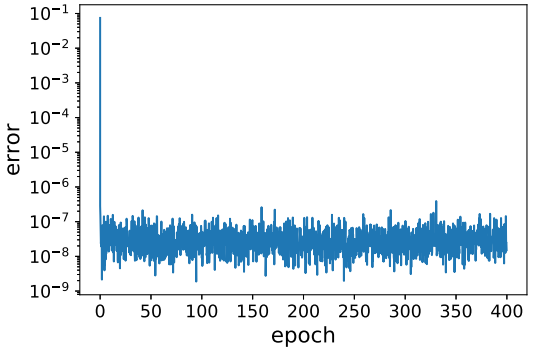


**Fig. 5.22.** Example 5: Training loss history.

$[-1.5, 1.5] \times [-0.5, 0.5] \times [-0.2, 0.2]^2 \times [-0.1, 0.1]^2 \times [-0.05, 0.05]^2 \times [-0.02, 0.02]$, from which we sample 500, 000 training data. In this example, we use the four-block ResNet method ($K = 4$) with each block containing 3 hidden layers of equal width of 30 neurons. The network model is trained for up to 2,000 epochs, and training loss history is shown in Fig. 5.14. Then we validate the trained model for the initial condition

$$u_0(x) = -\sin(x).$$

In Fig. 5.15, we present the prediction results generated by the trained network, for time up to $t = 2$. One can see that the predicted solutions are very close to the exact ones, This can also be seen in the relative error of the prediction from Fig. 5.16. We note that at time $t = 2$ the exact solution develops a relatively sharp (albeit still smooth) transition layer at this relatively low viscosity. The numerical solution starts to exhibit Gibbs' type small oscillations. This is a rather common feature for global type approximation and is not unexpected. Comparison between the learned and optimal expansion coefficients is also shown in Fig. 5.17.

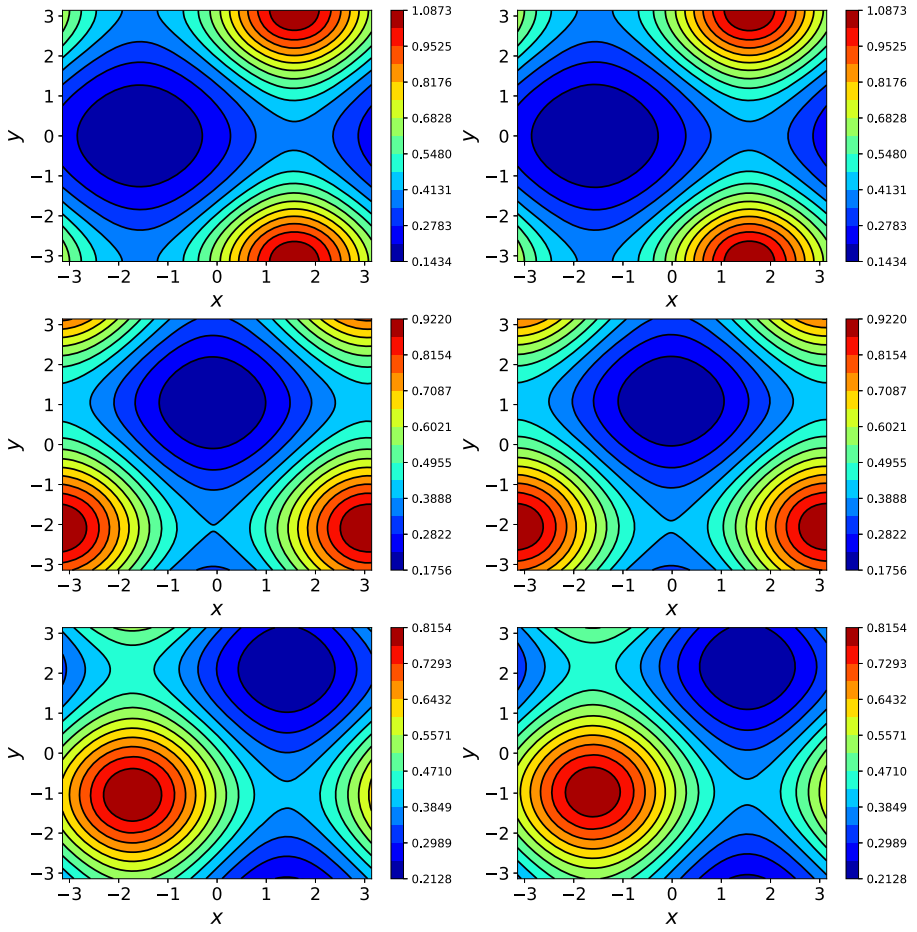


**Fig. 5.23.** Example 5: Contour plots of the solutions at different time. From top to bottom $t = 0$, $t = 1.5$ and $t = 3$. Left: learned model solution; right: true solution. 15 equally spaced contour lines are shown at the same levels for the learned model solution and true solution. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

### *5.4. Example 4: inviscid Burgers' equation*

We now consider the inviscid Burgers' equation with Dirichlet boundary condition:

$$\begin{cases} u_t + \left(\frac{u^2}{2}\right)_x = 0, \quad (x,t) \in (-\pi,\pi) \times \mathbb{R}^+, \\ u(-\pi,t) = u(\pi,t) = 0, \quad t \in \mathbb{R}^+. \end{cases} \tag{5.4}$$

This represents a challenging problem, as the nonlinear hyperbolic nature of this equation can produce shocks over time even if the initial solution is smooth.

The approximation space is chosen as $\mathbb{V}_n = \text{span}\{\sin(jx), 1 \le j \le 9\}$ with $n = 9$. The time lag $\Delta$ is taken as 0.05. The domain $D$ in the modal space is taken as $[-1.1, 1.1] \times [-0.5, 0.5] \times [-0.3 \times 0.3]^7$, from which we sample $1,000,000$ training data. We remark that by sampling in this manner, all of our training data are smooth. In this example, we use the block ResNet method with $K = 4$ blocks, each of which contains 3 hidden layers of equal width of 30. The network training is conducted for up to $2,000$ epochs, when it was deemed satisfactory, as shown in the loss history in Fig. 5.18. For validation, we conduct system prediction using initial condition

$$u_0(x) = -\sin(x),$$

for time up to $t = 2$. Although the initial condition is smooth, the exact solution will start to develop shock at $t = 1$.

The prediction results generated by our network model are shown in Fig. 5.19, along with the exact solution, and the Galerkin solution of the Burgers' equation using the same linear space $\mathbb{V}_n$. The evolution of numerical errors in the predictions by our network model are shown in Fig. 5.20. Due to the discontinuity in the solution, Gibbs type oscillations appear in the predicted solutions. This is not unexpected, as the best representation of a discontinuous function using the

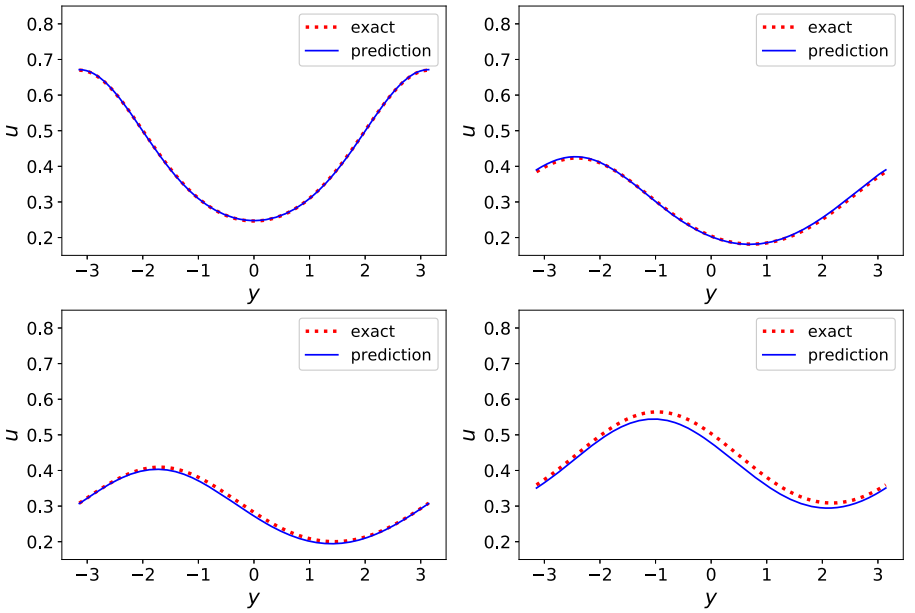


**Fig. 5.24.** Example 5: Comparison of the true solution and the learned model solution at different time along the line $x = 0$. Top-left: $t = 0$; top-right: $t = 1$; bottom-left: $t = 2$; bottom-right: $t = 3$.

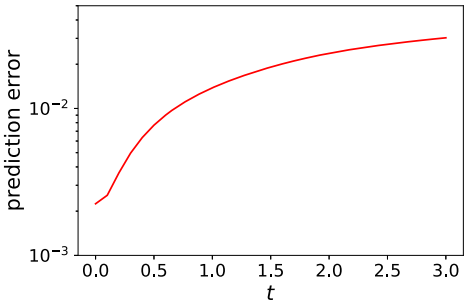


**Fig. 5.25.** Example 5: The evolution of the relative error in the prediction in $l^2$-norm.

linear subspace $\mathbb{V}_n$ will naturally produce oscillations, unless special treatment such as filtering is utilized (which is not pursued in this work). The Galerkin solution of the equation (5.4) exhibits the similar Gibbs' oscillations for precisely the same reason. We remark that it can be seen that our neural network prediction is visibly better than the Galerkin solution. While the network prediction is entirely data driven and does not require knowledge of the equation, the Galerkin solution is attainable only after knowing the precise form of the governing equation. In Fig. 5.21, we plot the evolution of the expansion coefficients in the modal space obtained by the neural network model prediction, Galerkin solution of the Burgers' equation, and orthogonal projection of the exact solution (denoted as "optimal"), the last of which serves as the reference solution. It is clearly seen that the neural network model produces more accurate results than the Galerkin solver. The accuracy improvement is especially visible at higher modes such as $\hat{v}_7$, $\hat{v}_8$, and $\hat{v}_9$. The cause of the accuracy improvement over Galerkin method will be pursued in a separate work.

### 5.5. Example 5: two-dimensional convection-diffusion equation

In our last example, we consider a two-dimensional convection-diffusion equation to demonstrate the applicability of the proposed algorithm for multiple dimensions. The equation set up is as follows.

$$\begin{cases} u_t + \alpha_1 u_x + \alpha_2 u_y = \sigma_1 u_{xx} + \sigma_2 u_{yy}, \quad (x, y, t) \in (-\pi, \pi)^2 \in \mathbb{R}^+, \\ u(-\pi, y, t) = u(\pi, y, t), \ u_x(-\pi, y, t) = u_x(\pi, y, t), \ \ (y, t) \in (-\pi, \pi) \times \mathbb{R}^+, \\ u(x, -\pi, t) = u(x, \pi, t), \ u_y(x, -\pi, t) = u_y(x, \pi, t), \ \ (x, t) \in (-\pi, \pi) \times \mathbb{R}^+, \end{cases} \tag{5.5}$$

where the parameters are set as $\alpha_1 = 1$, $\alpha_2 = 0.7$, $\sigma_1 = 0.1$, and $\sigma_2 = 0.16$ in the test.

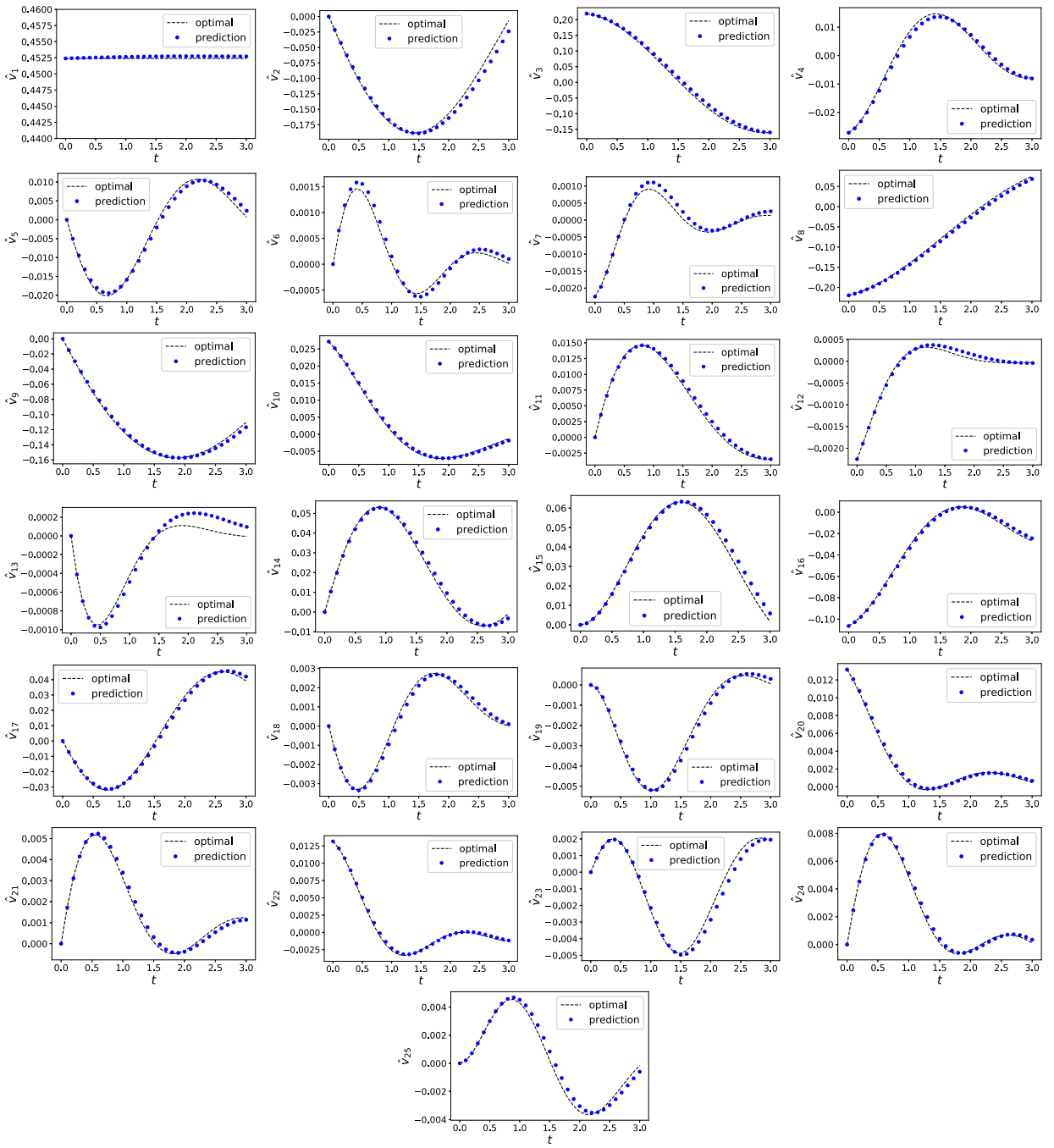


**Fig. 5.26.** Example 5: Evolution of the expansion coefficients for the learned model solution and the projection of the true solution.

We chose the finite dimensional approximation space $\mathbb{V}_n$ as a span by $n = 25$ basis functions:

$$\phi_1(x, y) = 1, \quad \phi_2(x, y) = \cos(x), \quad \phi_3(x, y) = \sin(x),$$
$$\phi_4(x, y) = \cos(2x), \quad \phi_5(x, y) = \sin(2x), \quad \phi_6(x, y) = \cos(3x), \quad \phi_7(x, y) = \sin(3x),$$
$$\phi_8(x, y) = \cos(y), \quad \phi_9(x, y) = \sin(y), \quad \phi_{10}(x, y) = \cos(2y), \quad \phi_{11}(x, y) = \sin(2y),$$
$$\phi_{12}(x, y) = \cos(3y), \quad \phi_{13}(x, y) = \sin(3y), \quad \phi_{14}(x, y) = \cos(x)\cos(y),$$
$$\phi_{15}(x, y) = \cos(x)\sin(y), \quad \phi_{16}(x, y) = \sin(x)\cos(y), \quad \phi_{17}(x, y) = \sin(x)\sin(y),$$

$$\phi_{18}(x, y) = \cos(x)\cos(2y), \quad \phi_{19}(x, y) = \cos(x)\sin(2y), \quad \phi_{20}(x, y) = \sin(x)\cos(2y),$$
$$\phi_{21}(x, y) = \sin(x)\sin(2y), \quad \phi_{22}(x, y) = \cos(2x)\cos(y), \quad \phi_{23}(x, y) = \cos(2x)\sin(y),$$
$$\phi_{24}(x, y) = \sin(2x)\cos(y), \quad \phi_{25}(x, y) = \sin(2x)\sin(y).$$

The time lag $\Delta$ is taken as 0.1 and $1,000,000$ training data are generated. The neural network model is a five-block ResNet method ($K = 5$) with each block containing 3 hidden layers of equal width of 40 neurons. Upon training the network model satisfactorily for 400 epochs (see Fig. 5.22 for the training loss history), we validate the trained models by using the initial condition

$$u_0(x) = \frac{2}{5}\exp\left(\frac{\sin(x) - \cos(y)}{2}\right),$$

for time up to $t = 3$.

The comparison between the network model prediction and the exact solution is shown in Figs. 5.23 as solution contours, and in Fig. 5.24 as solution slice profiles at certain locations. The relative error in the network prediction is shown in Fig. 5.25. The time evolution of the modal expansion coefficients is shown in Fig. 5.26, along with the evolution of the orthogonal projection coefficients of the exact solution. Again, we observe good accuracy in the network prediction.

## 6. Conclusion

In this paper, we presented a data-driven framework for learning unknown time-dependent autonomous PDEs, based on training of deep neural networks, particularly, those based on residual networks. Instead of identifying the exact terms in the underlying PDEs forms of the unknown PDEs, we proposed to approximately recover the evolution operator of the underlying PDEs. Since the evolution operator completely characterizes the solution evolution, its recovery allows us to conduct accurate system prediction by recursive use of the operator. The key to the successful learning of the operator is to reduce the problem into finite dimension. To this end, we proposed an approach in modal space, i.e., generalized Fourier space. Error analysis was conducted to quantify the prediction accuracy of the proposed data-driven approach. We presented a variety of test problems to demonstrate the applicability and potential of the method. More detailed study of its properties, especially to more complex problems in high dimensions such as electronic structure problem [7], will be pursued in future study.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: large-scale machine learning on heterogeneous systems, https://www.tensorflow.org/, 2015. Software available from tensorflow.org.
[2] J. Bongard, H. Lipson, Automated reverse engineering of nonlinear dynamical systems, Proc. Natl. Acad. Sci. USA 104 (2007) 9943–9948.
[3] S.L. Brunton, B.W. Brunton, J.L. Proctor, E. Kaiser, J.N. Kutz, Chaos as an intermittently forced linear system, Nat. Commun. 8 (2017).
[4] S.L. Brunton, J.L. Proctor, J.N. Kutz, Discovering governing equations from data by sparse identification of nonlinear dynamical systems, Proc. Natl. Acad. Sci. USA 113 (2016) 3932–3937.
[5] S. Chan, A. Elsheikh, A machine learning approach for efficient uncertainty quantification using multiscale methods, J. Comput. Phys. 354 (2018) 494–511.
[6] J. Han, A. Jentzen, W. E, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci. 115 (2018) 8505–8510.
[7] J. Han, L. Zhang, W. E, Solving many-electron Schrödinger equation using deep neural networks, J. Comput. Phys. 399 (2019) 108929.
[8] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[9] J. Hesthaven, S. Ubbiali, Non-intrusive reduced order modeling of nonlinear problems using neural networks, J. Comput. Phys. 363 (2018) 55–78.
[10] S.H. Kang, W. Liao, Y. Liu, IDENT: identifying differential equations with numerical time evolution, arXiv preprint arXiv:1904.03538, 2019.
[11] S. Karumuri, R. Tripathy, I. Bilionis, J. Panchal, Simulator-free solution of high-dimensional stochastic elliptic partial differential equations using deep neural networks, arXiv preprint arXiv:1902.05200, 2019.
[12] Y. Khoo, J. Lu, L. Ying, Solving parametric PDE problems with artificial neural networks, arXiv preprint arXiv:1707.03351, 2018.
[13] Z. Long, Y. Lu, B. Dong, PDE-Net 2.0: learning PDEs from data with a numeric-symbolic hybrid deep network, arXiv preprint arXiv:1812.04426, 2018.
[14] Z. Long, Y. Lu, X. Ma, B. Dong, PDE-net: learning PDEs from data, in: J. Dy, , A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, Stockholmsmässan, Stockholm Sweden, 10–15 Jul, in: Proceedings of Machine Learning Research, vol. 80, 2018, pp. 3208–3216.
[15] N.M. Mangan, J.N. Kutz, S.L. Brunton, J.L. Proctor, Model selection for dynamical systems via sparse regression and information criteria, Proc. R. Soc., Math. Phys. Eng. Sci. 473 (2017).
[16] D. Nguyen, S. Ouala, L. Drumetz, R. Fablet, EM-like learning chaotic dynamics from noisy and partial observations, arXiv preprint arXiv:1903.10335, 2019.
[17] S. Pawar, S.M. Rahman, H. Vaddireddy, O. San, A. Rasheed, P. Vedula, A deep learning enabler for nonintrusive reduced order modeling of fluid flows, Phys. Fluids 31 (2019) 085101.

[18] T. Qin, K. Wu, D. Xiu, Data driven governing equations approximation using deep neural networks, J. Comput. Phys. 395 (2019) 620–635.
[19] M. Raissi, Deep hidden physics models: deep learning of nonlinear partial differential equations, J. Mach. Learn. Res. 19 (2018) 1–24.
[20] M. Raissi, G.E. Karniadakis, Hidden physics models: machine learning of nonlinear partial differential equations, J. Comput. Phys. 357 (2018) 125–141.
[21] M. Raissi, P. Perdikaris, G.E. Karniadakis, Machine learning of linear differential equations using gaussian processes, J. Comput. Phys. 348 (2017) 683–693.
[22] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (part i): data-driven solutions of nonlinear partial differential equations, arXiv preprint arXiv:1711.10561, 2017.
[23] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (part ii): data-driven discovery of nonlinear partial differential equations, arXiv preprint arXiv:1711.10566, 2017.
[24] M. Raissi, P. Perdikaris, G.E. Karniadakis, Multistep neural networks for data-driven discovery of nonlinear dynamical systems, arXiv preprint arXiv: 1801.01236, 2018.
[25] D. Ray, J. Hesthaven, An artificial neural network as a troubled-cell indicator, J. Comput. Phys. 367 (2018) 166–191.
[26] S.H. Rudy, S.L. Brunton, J.L. Proctor, J.N. Kutz, Data-driven discovery of partial differential equations, Sci. Adv. 3 (2017) e1602614.
[27] S.H. Rudy, J.N. Kutz, S.L. Brunton, Deep learning of dynamics and signal-noise decomposition with time-stepping constraints, J. Comput. Phys. 396 (2019) 483–506.
[28] H. Schaeffer, Learning partial differential equations via data discovery and sparse optimization, Proc. R. Soc., Math. Phys. Eng. Sci. 473 (2017).
[29] H. Schaeffer, S.G. McCalla, Sparse model selection via integral terms, Phys. Rev. E 96 (2017) 023302.
[30] H. Schaeffer, G. Tran, R. Ward, Extracting sparse high-dimensional dynamics from limited data, SIAM J. Appl. Math. 78 (2018) 3279–3295.
[31] M. Schmidt, H. Lipson, Distilling free-form natural laws from experimental data, Science 324 (2009) 81–85.
[32] Y. Sun, L. Zhang, H. Schaeffer, NeuPDE: neural network based ordinary and partial differential equations for modeling time-dependent data, arXiv preprint arXiv:1908.03190, 2019.
[33] R. Tibshirani, Regression shrinkage and selection via the lasso, J. R. Stat. Soc. B (1996) 267–288.
[34] G. Tran, R. Ward, Exact recovery of chaotic systems from highly corrupted data, Multiscale Model. Simul. 15 (2017) 1108–1129.
[35] R. Tripathy, I. Bilionis, Deep UQ: learning deep neural network surrogate model for high dimensional uncertainty quantification, J. Comput. Phys. 375 (2018) 565–588.
[36] Y. Wang, G. Lin, Efficient deep learning techniques for multiphase flow simulation in heterogeneous porous media, arXiv preprint arXiv:1907.09571, 2019.
[37] Y. Wang, Z. Shen, Z. Long, B. Dong, Learning to discretize: solving 1D scalar conservation laws via deep reinforcement learning, arXiv preprint arXiv: 1905.11079, 2019.
[38] K. Wu, T. Qin, D. Xiu, Structure-preserving method for reconstructing unknown hamiltonian systems from trajectory data, arXiv preprint arXiv:1905. 10396, 2019.
[39] K. Wu, D. Xiu, Numerical aspects for approximating governing equations using data, J. Comput. Phys. 384 (2019) 200–221.
[40] Y. Zhu, N. Zabaras, Bayesian deep convolutional encoder-decoder networks for surrogate modeling and uncertainty quantification, J. Comput. Phys. 366 (2018) 415–447.