Journal of Computational Physics 395 (2019) 620–635

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Data driven governing equations approximation using deep neural networks ☆

Tong Qin, Kailiang Wu, Dongbin Xiu *

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

ARTICLE INFO

Article history:
Received 13 November 2018
Received in revised form 22 February 2019
Accepted 15 June 2019
Available online 19 June 2019

Keywords:
Deep neural network
Residual network
Recurrent neural network
Governing equation discovery

ABSTRACT

We present a numerical framework for approximating unknown governing equations using observation data and deep neural networks (DNN). In particular, we propose to use residual network (ResNet) as the basic building block for equation approximation. We demonstrate that the ResNet block can be considered as a one-step method that is exact in temporal integration. We then present two multi-step methods, recurrent ResNet (RT-ResNet) method and recursive ReNet (RS-ResNet) method. The RT-ResNet is a multi-step method on uniform time steps, whereas the RS-ResNet is an adaptive multi-step method using variable time steps. All three methods presented here are based on integral form of the underlying dynamical system. As a result, they do not require time derivative data for equation recovery and can cope with relatively coarsely distributed trajectory data. Several numerical examples are presented to demonstrate the performance of the methods.

© 2019 Elsevier Inc. All rights reserved.

1. Introduction

Recently there has been a growing interest in discovering governing equations numerically using observational data. Earlier efforts include methods using symbolic regression ([5,43]), equation-free modeling [24], heterogeneous multi-scale method (HMM) ([15]), artificial neural networks ([19]), nonlinear regression ([50]), empirical dynamic modeling ([46,53]), nonlinear Laplacian spectral analysis ([18]), automated inference of dynamics ([44,12,13]), etc. More recent efforts start to cast the problem into a function approximation problem, where the unknown governing equations are treated as target functions relating the data for the state variables and their time derivatives. The majority of the methods employ certain sparsity-promoting algorithms to create parsimonious models from a large set of dictionary for all possible models, so that the true dynamics could be recovered exactly ([47]). Many studies have been conducted to effectively deal with noises in data ([7,41]), corruptions in data ([48]), partial differential equations [38,40], etc. Methods have also been developed in conjunction with model selection approach ([28]), Koopman theory ([6]), and Gaussian process regression ([35]), to name a few. A more recent work resorts to the more traditional means of approximation by using orthogonal polynomials ([52]). The approach seeks accurate numerical approximation to the underlying governing equations, instead of their exact recovery. By doing so, many existing results in polynomial approximation theory can be applied, particularly those on sampling strategies. It was shown in [52] that data from a large number of short bursts of trajectories are more effective for equation recovery than those from a single long trajectory.

☆ This work was partially supported by AFOSR FA9550-18-1-0102.
* Corresponding author.
E-mail addresses: qin.428@osu.edu (T. Qin), wu.3423@osu.edu (K. Wu), xiu.16@osu.edu (D. Xiu).

https://doi.org/10.1016/j.jcp.2019.06.042
0021-9991/© 2019 Elsevier Inc. All rights reserved.

On the other hand, artificial neural network (ANN), and particularly deep neural network (DNN), has seen tremendous successes in many different disciplines. The number of publications is too large to mention. Here we cite only a few relatively more recent review/summary type publications ([30,4,16,32,14,20,42]). Efforts have been devoted to the use of ANN for various aspects of scientific computing, including construction of reduced order model ([22]), aiding solution of conservation laws ([37]), multiscale problems ([8,51]), solving and learning systems involving ODEs and PDEs ([29,11,27,25]), uncertainty quantification ([49,54]), etc.

The focus of this paper is on the approximation/learning of dynamical systems using deep neural networks (DNN). The topic has been explored in a series of recent articles, in the context of ODEs ([36,39]) and PDEs ([34,33,27]). The new contributions of this paper include the following. First, we introduce new constructions of deep neural network (DNN), specifically suited for learning dynamical systems. In particular, our new network structures employ residual network (ResNet), which was first proposed in [21] for image analysis and has become very popular due to its effectiveness. In our construction, we employ a ResNet block, which consists of multiple fully connected hidden layers, as the fundamental building block of our DNN structures. We show that the ResNet block can be considered as a one-step numerical integrator in time. This integrator is "exact" in time, i.e., no temporal error, in the sense that the only error stems from the neural network approximation of the evolution operators defining the governing equation. This is different from a few existing work where ResNet is viewed as the Euler forward scheme ([9]). Secondly, we introduce two variations of the ResNet structure to serve as multi-step learning of the underlying governing equations. The first one employs recurrent use of the ResNet block. This is inspired by the well known recurrent neural network (RNN), whose connection with dynamical systems has long been recognized, cf. [20]. Our recurrent network, termed RT-ResNet hereafter, is different in the sense that the recurrence is enforced blockwise on the ResNet block, which by itself is a DNN. (Note that in the traditional RNN, the recurrence is enforced on the hidden layers.) We show that the RT-ResNet is a multi-step integrator that is exact in time, with the only error stemming from the ResNet approximation of the evolution operator of the underlying equation. The other variation of the ResNet approximator employs recursive use of the ResNet block, termed RS-ResNet. Again, the recursion is enforced blockwise on the ResNet block (which is a DNN). We show that the RS-ResNet is also an exact multi-step integrator. The difference between RT-ResNet and RS-ResNet is that the former is equivalent to a multi-step integrator using an uniform time step, whereas the latter is an "adaptive" method with variable time steps depending on the particular problem and data. Thirdly, the derivations in this paper utilize integral form of the underlying dynamical system. By doing so, the proposed methods do not require knowledge or data of the time derivatives of the equation states. This is different from most of the existing studies (cf. [5,7,41,52]), which deal with the equations directly and thus require time derivative data. Acquiring time derivatives introduces an additional source for noises and errors, particularly when one has to conduct numerical differentiation of noisy trajectory data. Consequently, the proposed three new DNN structures, the one-step ResNet and multi-step RT-ResNet and RS-ResNet, are capable of approximating unknown dynamical systems using only state variable data, which could be relatively coarsely distributed in time. In this case, most of the existing methods become less effective, as accurate extraction of time derivatives is difficult.

This paper is organized as follows. After the basic problem setup in Section 2, we present the main methods in Section 3 and some theoretical properties in Section 4. We then present, in Section 5, a set of numerical examples, covering both linear and nonlinear differential equations, to demonstrate the effectiveness of the proposed algorithms.

2. Setup

Let us consider an autonomous system

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}), \qquad \mathbf{x}(t_0) = \mathbf{x}_0, \tag{2.1}$$

where $\mathbf{x} \in \mathbb{R}^n$ are the state variables. Let $\mathbf{\Phi}_s : \mathbb{R}^n \to \mathbb{R}^n$ be the flow map, which maps the state from $t = 0$ to the state at $t = s$. Note that for autonomous systems the time variable t can be arbitrarily shifted and only the time difference, or time lag, $t - t_0$ is relevant. The solution can be written as

$$\mathbf{x}(t; \mathbf{x}_0, t_0) = \mathbf{\Phi}_{t-t_0}(\mathbf{x}_0). \tag{2.2}$$

Hereafter we will omit t in the exposition, unless confusion arises.

In this paper, we assume the form of the governing equations $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ is unknown. Our goal is to create an accurate model for the governing equation using data of the solution trajectories. In particular, we assume data are collected in the form of pairs, each of which corresponds to the solution states along one trajectory at two different time instances. That is, we consider the set

$$\mathcal{S} = \{(\mathbf{z}_j^{(1)}, \mathbf{z}_j^{(2)}) : j = 1, \ldots, J\}, \tag{2.3}$$

where J is the total number of data pairs, and for each pair $j = 1, \ldots, J$,

$$\mathbf{z}_j^{(1)} = \mathbf{x}_j + \epsilon_j^{(1)}, \qquad \mathbf{z}_j^{(2)} = \mathbf{\Phi}_{\Delta_j}(\mathbf{x}_j) + \epsilon_j^{(2)}. \tag{2.4}$$

Here the terms $\epsilon_j^{(1)}$ and $\epsilon_j^{(2)}$ stand for the potential noises in the data, and Δ_j is the time lag between the two states. For notational convenience, we assume $\Delta_j = \Delta$ to be a constant for all j throughout this paper. Consequently, the data set becomes input-output measurements of the Δ-lag flow map,

$$\mathbf{x} \to \mathbf{\Phi}_\Delta(\mathbf{x}). \tag{2.5}$$

3. Deep neural network approximation

The core building block of our methods is a standard fully connected feedforward neural network (FNN) with $M \geq 3$ layers, of which $(M-2)$ are hidden layers. It has been established that fully connected FNN can approximate arbitrarily well a large class of input-output maps, i.e., they are universal approximators, cf. [31,2,23]. Since the right-hand-side $\mathbf{f}$ of (2.1) is our approximation goal, we will consider $\mathbb{R}^n \to \mathbb{R}^n$ map. Let n_j, $j = 1, \ldots, M$, be the number of neurons in each layer, we then have $n_1 = n_M = n$.

Let $\mathbf{N} : \mathbb{R}^n \to \mathbb{R}^n$ be the operator of this network. For any input $\mathbf{y}^{in} \in \mathbb{R}^n$, the output of the network is

$$\mathbf{y}^{out} = \mathbf{N}(\mathbf{y}^{in}; \Theta), \tag{3.1}$$

where Θ is the parameter set including all the parameters in the network. The operator $\mathbf{N}$ is a composition of the following operators

$$\mathbf{N}(\cdot; \Theta) = (\sigma_M \circ \mathbf{W}_{M-1}) \circ \cdots \circ (\sigma_2 \circ \mathbf{W}_1), \tag{3.2}$$

where $\circ$ stands for operator composition, $\mathbf{W}_j$ is the weight matrix containing the weight parameters connecting the neurons from j-th layer to $(j+1)$-th layer, after using the standard approach of augmenting the biases into the weights, and $\sigma_j : \mathbb{R} \to \mathbb{R}$ is the activation function, which is applied component-wise to the j-th layer. There exist many choices for the activation functions, e.g., sigmoid functions, ReLU (rectified linear unit), etc. In this paper we use a sigmoid function, in particular, the $\sigma_i(x) = \tanh(x)$ function, in all layers, except at the output layer $\sigma_M(x) = x$. This is one of the common choices for DNN.

Using the data set (2.3), we can directly train (3.1) to approximate the Δ-lag flow map (2.5). This can be done by applying (3.1) with $\mathbf{y}_j^{in} = \mathbf{z}_j^{(1)}$ to obtain $\mathbf{y}_j^{out}$ for each $j = 1, \ldots, J$, and then minimizing following mean squared loss function

$$L(\Theta) = \frac{1}{J} \sum_{j=1}^{J} \left\| \mathbf{y}_j^{out} - \mathbf{z}_j^{(2)} \right\|^2, \tag{3.3}$$

where $\|\cdot\|$ denotes vector 2-norm hereafter. With a slight abuse of notation, hereafter we will write $\mathbf{y}^{in} = \mathbf{z}^{(1)}$ to stand for $\mathbf{y}_j^{in} = \mathbf{z}_j^{(1)}$ for all sample data $j = 1, \ldots, J$, unless confusion arises otherwise.

3.1. One-step ResNet approximation

We now present the idea of using residual neural network (ResNet) as a one-step approximation method. The idea of ResNet is to explicitly introduce the identity operator in the network and force the network to effectively approximate the "residue" of the input-output map. Although mathematically equivalent, this simple transformation has been shown to be highly advantageous in practice and become increasingly popular, after its formal introduction in [21].

The structure of the ResNet is illustrated in Fig. 3.1. The ResNet block consists of N fully connected hidden layers and an identity operator to re-introduce the input $\mathbf{y}^{in}$ back into the output of the hidden layers. The introduction of the identity operator effectively produces the following mapping

$$\mathbf{y}^{out} = \mathbf{y}^{in} + \mathbf{N}(\mathbf{y}^{in}; \Theta), \quad \mathbf{y}^{in} = \mathbf{z}^{(1)}, \tag{3.4}$$

where Θ are the weight and bias parameters in the network. The parameters are determined by minimizing the same loss function (3.3). This effectively accomplishes the training of the operator $\mathbf{N}(\cdot; \Theta)$.

The connection between dynamical systems and ResNet has been recognized. In fact, ResNet has been viewed as the Euler forward time integrator ([9]). To further examine its property, let us consider the exact Δ-lag flow map,

$$\begin{aligned} \mathbf{x}(\Delta) &= \mathbf{\Phi}_\Delta(\mathbf{x}(0)) \\ &= \mathbf{x}(0) + \int_0^\Delta \mathbf{f}(\mathbf{x}(t))dt \\ &= \mathbf{x}(0) + \Delta \cdot \mathbf{f}(\mathbf{x}(\tau)) \\ &= \mathbf{x}(0) + \Delta \cdot \mathbf{f}(\mathbf{\Phi}_\tau(\mathbf{x}(0))), \quad 0 \leq \tau \leq \Delta. \end{aligned} \tag{3.5}$$

This is a trivial derivation using the mean value theorem. For notational convenience, we now define "effective increment".

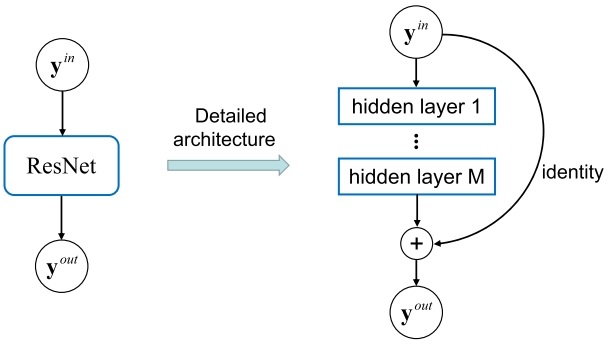


Fig. 3.1. Schematic of the ResNet structure for one-step approximation.

Definition 3.1. For a given autonomous system (2.1), given an initial state $\mathbf{x}$ and an increment $\Delta \geq 0$, then its effective increment of size Δ is defined as

$$\boldsymbol{\phi}_{\Delta}(\mathbf{x}; \mathbf{f}) = \Delta \cdot \mathbf{f}(\boldsymbol{\Phi}_{\tau}(\mathbf{x})), \tag{3.6}$$

for some $0 \leq \tau \leq \Delta$ such that

$$\mathbf{x}(\Delta) = \mathbf{x} + \boldsymbol{\phi}_{\Delta}(\mathbf{x}; \mathbf{f}). \tag{3.7}$$

Note that the effective increment $\boldsymbol{\phi}_{\Delta}$ depends only on its initial state $\mathbf{x}$, once the governing equation $\mathbf{f}$ and the increment Δ are fixed.

Upon comparing the exact state (3.7) and the one-step ResNet method (3.4), it is thus easy to see that a successfully trained network operator $\mathbf{N}$ is an approximation to the effective increment $\boldsymbol{\phi}_{\Delta}$, i.e.,

$$\mathbf{N}(\mathbf{x}; \Theta) \approx \boldsymbol{\phi}_{\Delta}(\mathbf{x}; \mathbf{f}). \tag{3.8}$$

Since the effective increment completely determines the true solution states on a Δ interval, we can then use the ResNet operator to approximate the solution trajectory. That is, starting with a given initial state $\mathbf{y}^{(0)} = \mathbf{x}$, we can time march the state

$$\mathbf{y}^{(k+1)} = \mathbf{y}^{(k)} + \mathbf{N}(\mathbf{y}^{(k)}; \Theta), \qquad k = 0, \ldots. \tag{3.9}$$

This discrete dynamical system serves as our approximation to the true dynamical system (2.1). It gives us an approximation to the true states on a uniform time grids with stepsize Δ.

Remark 3.1. Even though the approximate system (3.9) resembles the well known Euler forward time stepping scheme, it is not a first-order method in time. In fact, upon comparing (3.9) and the true state (3.7), it is easy to see that (3.9) is "exact" in term of temporal integration. The only source of error in the system (3.9) is the approximation error of the effective increment in (3.8). The size of this error is determined by the quality of the data and the network training algorithm.

Remark 3.2. The derivation here is based on (3.7), which is from the integral form of the governing equation. As a result, training of the ResNet method does not require data on the time derivatives of the true states. Moreover, Δ does not need to be exceedingly small (to enable accurate numerical differentiation in time). This makes the ResNet method suitable for problems with relatively coarsely distributed data.

3.2. Multi-step recurrent ResNet (RT-ResNet) approximation

We now combine the idea of recurrent neural network (RNN) and the ResNet method from the previous section. The distinct feature of our construction is that the recurrence is applied to the entire ResNet block, rather than to the individual hidden layers, as is done for the standard RNNs. (For an overview of RNN, interested readers are referred to [20], Ch. 10.)

The structure of the resulting Recurrent ResNet (RT-ResNet) is shown in Fig. 3.2. The ResNet block, as presented in Fig. 3.1, is "repeated" $(K-1)$ times, for an integer $K \geq 1$, before producing the output $\mathbf{y}^{out}$. This makes the occurrence of the ResNet block a total of K times. The unfolded structure is shown on the right of Fig. 3.2. The RT-ResNet then produces the following scheme, for $K \geq 1$,

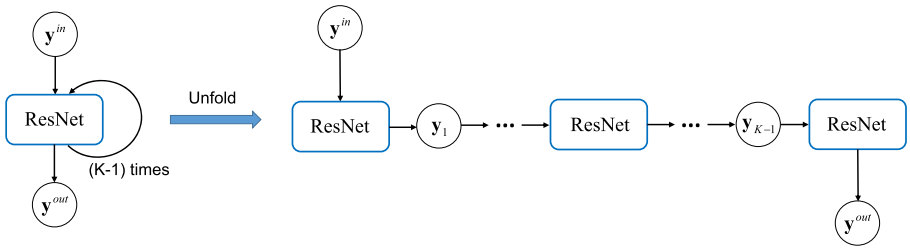


Fig. 3.2. Schematic of the recurrent ResNet (RT-ResNet) structure for multi-step approximation ($K \geq 1$).

$$\begin{cases} \mathbf{y}_0 = \mathbf{z}^{(1)}, \\ \mathbf{y}_{k+1} = \mathbf{y}_k + \mathbf{N}(\mathbf{y}_k; \Theta), \quad k = 0, \ldots, K-1, \\ \mathbf{y}^{out} = \mathbf{y}_K. \end{cases} \tag{3.10}$$

The network is then trained by using the data set (2.3) and minimizing the same loss function (3.3). For $K = 1$, this reduces to the one-step ResNet method (3.4).

To examine the properties of the RT-ResNet, let us consider a unform discretization of the time lag Δ. That is, let $\delta = \Delta/K$, and consider, $t_k = k\delta$, $k = 0, \ldots, K$. The exact solution state $\mathbf{x}$ satisfies the following relation

$$\begin{cases} \mathbf{x}(t_0) = \mathbf{x}(0), \\ \mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + \boldsymbol{\phi}_\delta(\mathbf{x}(t_k); \mathbf{f}), \quad k = 0, \ldots, K-1, \\ \mathbf{x}(\Delta) = \mathbf{x}(t_K), \end{cases} \tag{3.11}$$

where $\boldsymbol{\phi}_\delta(\mathbf{x}; \mathbf{f})$ is the effective increment defined of size δ, as defined in Definition 3.1.

Upon comparing this with the RT-ResNet scheme (3.10), it is easy to see that training the RT-ResNet is equivalent to finding the operator $\mathbf{N}$ to approximate the δ-effective increment,

$$\mathbf{N}(\mathbf{x}; \Theta) \approx \boldsymbol{\phi}_\delta(\mathbf{x}; \mathbf{f}). \tag{3.12}$$

Similar to the one-step ResNet method, the multi-step RT-ResNet is also exact in time, as it contains no temporal discretization error. The only error stems from the approximation of the δ-effective increment.

Once the RT-ResNet is successfully trained, it gives us a discrete dynamical system (3.10) that can be further marched in time using any initial state. This is an approximation to the true dynamical system on uniformly distributed time instances with an interval $\delta = \Delta/K$. Therefore, even though the training data are given over Δ time interval, the RT-ResNet system can produce solution states on finer time grids with a step size $\delta \leq \Delta$ ($K \geq 1$).

3.3. Multi-step recursive ResNet approximation

We now present another multi-step approximation method based on the ResNet block in Fig. 3.1. The structure of the network is shown in Fig. 3.3. From the input $\mathbf{y}^{in}$, ResNet blocks are recursively used a total of $K \geq 1$ times, before producing the output $\mathbf{y}^{out}$. The network, referred to as recursive ResNet (RS-ResNet) hereafter, thus produces the following scheme, for any $K \geq 1$,

$$\begin{cases} \mathbf{y}_0 = \mathbf{z}^{(1)}, \\ \mathbf{y}_{k+1} = \mathbf{y}_k + \mathbf{N}(\mathbf{y}_k; \Theta_k), \quad k = 0, \ldots, K-1, \\ \mathbf{y}^{out} = \mathbf{y}_K. \end{cases} \tag{3.13}$$

Compared to the recurrent RT-ResNet method (3.10) from the previous section, the major difference in RS-ResNet is that each ResNet block inside the network has its own parameter sets Θ_k and thus are different from each other. Since each ResNet is a DNN by itself, the RS-ResNet can be a very deep network when $K > 1$. When $K = 1$, it also reduces back to the one-step ResNet network.

Let $0 = t_0 < t_1 < \cdots < t_K = \Delta$ be an arbitrarily distributed time instances in $[0, \Delta]$ and $\delta_k = t_{k+1} - t_k$, $k = 0, \ldots, K-1$, be the (non-uniform) increments. It is then straightforward to see that the exact state satisfies

$$\begin{cases} \mathbf{x}(t_0) = \mathbf{x}(0), \\ \mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + \boldsymbol{\phi}_{\delta_k}(\mathbf{x}(t_k); \mathbf{f}), \quad k = 0, \ldots, K-1, \\ \mathbf{x}(\Delta) = \mathbf{x}(t_K), \end{cases} \tag{3.14}$$

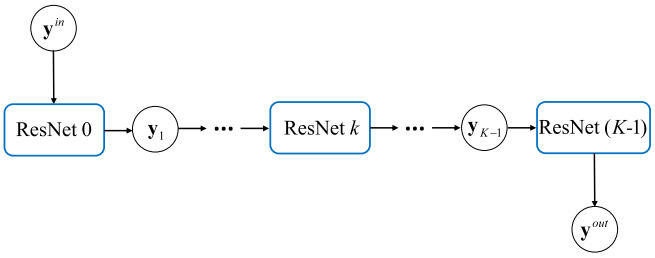


Fig. 3.3. Schematic of the recursive ResNet (RS-ResNet) structure for multi-step approximation ($K \geq 1$).

where $\boldsymbol{\phi}_{\delta_k}(\mathbf{x}; \mathbf{f})$ is the δ_k effective increment defined in Definition 3.1.

Upon comparing with the RS-ResNet scheme (3.13), one can see that the training of the RS-ResNet produces the following approximation

$$\mathbf{N}(\mathbf{x}; \Theta_k) \approx \boldsymbol{\phi}_{\delta_k}(\mathbf{x}; \mathbf{f}), \qquad k = 0, \ldots, K-1. \tag{3.15}$$

That is, each ResNet operator $\mathbf{N}(\mathbf{x}; \Theta_k)$ is an approximation of an effective increment of size δ_k, for $k = 0, \ldots, K-1$, under the condition $\sum_{k=0}^{K-1} \delta_k = \Delta$. Training the network using the data (2.3) and loss function (3.3) will determine the parameter sets Θ_k, and subsequently the effective increments with size δ_k, for $k = 0, \ldots, K-1$, From this perspective, one may view RS-ResNet as an "adaptive" method, as it adjusts its parameter sets to approximate K smaller effective increments whose increments are determined by the data. Since RS-ResNet is a very deep network with a large number of parameters, it is, in principle, capable of producing more accurate results than ResNet and RT-ResNet, assuming cautions have been exercised to prevent overfitting.

A successfully trained RS-ResNet also gives us a discrete dynamical system that approximates the true governing equation (2.1). Due to its "adaptive" nature, the intermediate time intervals δ_k are variables and not known explicitly. Therefore, the discrete RS-ResNet needs to be applied K times to produce the solution states over the time interval Δ, which is the same interval given by the training data. This is different from the RT-ResNet, which can produce solutions over a smaller and uniform time interval $\delta = \Delta/K$.

4. Theoretical properties

In this section we present a few straightforward analysis to demonstrate certain theoretical aspects of the proposed DNN for equation approximation.

4.1. Continuity of flow map

Under certain conditions on $\mathbf{f}$, one can show that the flow map of the dynamical system (2.1) is locally Lipschitz continuous.

Lemma 4.1. *Assume* $\mathbf{f}$ *is Lipschitz continuous with Lipschitz constant* L *on a set* $D \subseteq \mathbb{R}^n$. *For any* $\tau > 0$, *define*

$$D_\tau := \left\{ \mathbf{x}_0 \in D : \boldsymbol{\Phi}_t(\mathbf{x}_0) \in D, \ \forall t \in [0, \tau] \right\}.$$

Then, for any $t \in [0, \tau]$, *the flow map* $\boldsymbol{\Phi}_t$ *is Lipschitz continuous on* D_τ. *Specifically, for any* $\mathbf{x}_0, \widetilde{\mathbf{x}}_0 \in D_\tau$,

$$\|\boldsymbol{\Phi}_t(\mathbf{x}_0) - \boldsymbol{\Phi}_t(\widetilde{\mathbf{x}}_0)\| \leq \mathrm{e}^{Lt} \|\mathbf{x}_0 - \widetilde{\mathbf{x}}_0\|, \quad \forall t \in [0, \tau]. \tag{4.1}$$

Proof. The proof directly follows from the classical result on the continuity of the dynamical system (2.1) with respect to initial data; see [45, p. 109]. □

The above continuity ensures that the flow map can be approximated by neural networks to any desired degree of accuracy by increasing the number of hidden layers and neurons; see, for example, [26,31]. The Lipschitz continuity will also play an important role in the error analysis in Theorem 4.3.

4.2. Compositions of flow maps

It was shown in [3] that any smooth bi-Lipschitz function can be represented as compositions of functions, each of which is near-identity in Lipschitz semi-norm. For the flow map of the autonomous system (2.1), we can prove a stronger result by using the following property

$$\boldsymbol{\Phi}_{t_1} \circ \boldsymbol{\Phi}_{t_2} = \boldsymbol{\Phi}_{t_1 + t_2}, \quad \forall t_1, t_2. \tag{4.2}$$

Theorem 4.2. *For any positive integer $K \geq 1$, the flow map $\mathbf{\Phi}_\Delta$ can be expressed as a K-fold composition of $\mathbf{\Phi}_\delta$, namely,*

$$\mathbf{\Phi}_\Delta = \underbrace{\mathbf{\Phi}_\delta \circ \cdots \circ \mathbf{\Phi}_\delta}_{K-\text{fold}}, \tag{4.3}$$

where $\delta = \Delta/K$, and $\mathbf{\Phi}_\delta$ satisfies

$$\|\mathbf{\Phi}_\delta(\mathbf{x}_0) - \mathbf{x}_0\| \leq \frac{\Delta}{K} \sup_{t\in[0,\delta]} \|\mathbf{f}(\mathbf{\Phi}_t(\mathbf{x}_0))\|, \quad \forall \mathbf{x}_0. \tag{4.4}$$

Suppose that $\mathbf{f}$ is bounded on $D \subseteq \mathbb{R}^n$, then

$$\|\mathbf{\Phi}_\delta - \mathbf{I}\|_{L^\infty(D_\delta)} \leq \frac{\Delta}{K} \|\mathbf{f}\|_{L^\infty(D)} = \mathcal{O}\left(\frac{\Delta}{K}\right), \tag{4.5}$$

where $\mathbf{I} : \mathbb{R}^n \to \mathbb{R}^n$ is the identity map, and $\|\cdot\|_{L^\infty} := \operatorname{ess\,sup} \|\cdot\|$.

Proof. The representation (4.3) is a direct consequence of the property (4.2). For any $\mathbf{x}_0$, we have

$$\|\mathbf{\Phi}_\delta(\mathbf{x}_0) - \mathbf{x}_0\| = \left\| \int_0^\delta \mathbf{f}(\mathbf{\Phi}_t(\mathbf{x}_0))dt \right\| = \delta\eta\left(\frac{1}{\delta}\int_0^\delta \mathbf{f}(\mathbf{\Phi}_t(\mathbf{x}_0))dt\right),$$

where $\eta(\mathbf{x}) = \|\mathbf{x}\|$. Since η is a convex function, it satisfies the Jensen's inequality

$$\eta\left(\frac{1}{\delta}\int_0^\delta \mathbf{f}(\mathbf{\Phi}_t(\mathbf{x}_0))dt\right) \leq \frac{1}{\delta}\int_0^\delta \eta\big(\mathbf{f}(\mathbf{\Phi}_t(\mathbf{x}_0))\big)dt.$$

Thus we obtain

$$\|\mathbf{\Phi}_\delta(\mathbf{x}_0) - \mathbf{x}_0\| \leq \int_0^\delta \|\mathbf{f}(\mathbf{\Phi}_t(\mathbf{x}_0))\|\, dt \leq \delta \sup_{t\in[0,\delta]} \|\mathbf{f}(\mathbf{\Phi}_t(\mathbf{x}_0))\|,$$

which implies (4.4). For any $\mathbf{x}_0 \in D_\delta$, we have $\mathbf{\Phi}_t(\mathbf{x}_0) \in D$ for $0 \leq t \leq \delta$. Hence

$$\|\mathbf{\Phi}_\delta(\mathbf{x}_0) - \mathbf{x}_0\| \leq \frac{\Delta}{K}\|\mathbf{f}\|_{L^\infty(D)}, \quad \forall \mathbf{x}_0 \in D_\delta.$$

This yields (4.5), and the proof is complete. □

This estimate can serve as a theoretical justification of the ResNet method ($K = 1$) and RT-ResNet method ($K \geq 1$). As long as Δ is reasonably small, the flow map of the underlying dynamical system is close to identity. Therefore, it is natural to use ResNet, which explicitly introduces the identity operator, to approximate the "residue" of the flow map. The norm of the DNN operator $\mathbf{N}$, which approximates the residual flow map, $\mathbf{\Phi}_\delta - \mathbf{I}$, becomes small at $O(\Delta)$. For RT-ResNet with $K > 1$, its norm becomes even smaller at $O(\Delta/K)$. We remark that it was pointed out empirically in [9] that using multiple ResNet blocks can result in networks with smaller norm.

4.3. Error bound

Let $\mathcal{N}$ denote the neural network approximation operator to the Δ-lag flow map $\mathbf{\Phi}_\Delta$. For the proposed ResNet (3.4), RT-ResNet (3.10), and RS-ResNet (3.13), the operators can be written as

$$\begin{aligned}
&\mathcal{N} = \mathbf{I} + \mathbf{N}(\bullet;\Theta), \quad \text{ResNet};\\
&\mathcal{N} = \underbrace{\big(\mathbf{I} + \mathbf{N}(\bullet;\Theta)\big) \circ \cdots \circ \big(\mathbf{I} + \mathbf{N}(\bullet;\Theta)\big)}_{K-\text{fold}}, \quad \text{RT-ResNet};\\
&\mathcal{N} = \big(\mathbf{I} + \mathbf{N}(\bullet;\Theta_{K-1})\big) \circ \cdots \circ \big(\mathbf{I} + \mathbf{N}(\bullet;\Theta_0)\big), \quad \text{RS-ResNet}.
\end{aligned} \tag{4.6}$$

We now derive a general error bound for the solution approximation using the DNN operator $\mathcal{N}$. This bound serves a general guideline for the error growth. More specific error bounds for each different network structure are more involved and will be pursued in a future work.

Let $\mathbf{y}^{(m)}$ denote the solution of the approximate model at time $t^{(m)} := t_0 + m\Delta$. Let $\mathcal{E}^{(m)} := \|\mathbf{y}^{(m)} - \mathbf{x}(t^{(m)})\|$ denote the error, $j = 0, 1, \ldots, m$.

Theorem 4.3. *Assume that the same assumptions in Lemma 4.1 hold, and let us further assume*

1. $\|\mathcal{N} - \mathbf{\Phi}_\Delta\|_{L^\infty(D_\Delta)} < +\infty$,
2. $\mathbf{y}^{(i)}, \mathbf{x}(t^{(i)}) \in D_\Delta$ *for* $0 \le i \le m-1$,

then we have

$$\mathcal{E}^{(m)} \le \left(1 + \mathrm{e}^{L\Delta}\right)^m \mathcal{E}^{(0)} + \|\mathcal{N} - \mathbf{\Phi}_\Delta\|_{L^\infty(D_\Delta)} \frac{\left(1 + \mathrm{e}^{L\Delta}\right)^m - 1}{\mathrm{e}^{L\Delta}}. \tag{4.7}$$

Proof. The triangle inequality implies that

$$\begin{aligned}
\mathcal{E}^{(m)} &= \left\|\mathbf{y}^{(m-1)} + \mathcal{N}(\mathbf{y}^{(m-1)}) - \mathbf{x}(t^{(m-1)}) - \mathbf{\Phi}_\Delta(\mathbf{x}(t^{(m-1)}))\right\| \\
&\le \left\|\mathbf{y}^{(m-1)} - \mathbf{x}(t^{(m-1)})\right\| + \left\|\mathcal{N}(\mathbf{y}^{(m-1)}) - \mathbf{\Phi}_\Delta(\mathbf{x}(t^{(m-1)}))\right\| \\
&\le \left\|\mathbf{y}^{(m-1)} - \mathbf{x}(t^{(m-1)})\right\| + \left\|\mathcal{N}(\mathbf{y}^{(m-1)}) - \mathbf{\Phi}_\Delta(\mathbf{y}^{(m-1)})\right\| \\
&\quad + \left\|\mathbf{\Phi}_\Delta(\mathbf{y}^{(m-1)}) - \mathbf{\Phi}_\Delta(\mathbf{x}(t^{(m-1)}))\right\| \\
&\le \left\|\mathbf{y}^{(m-1)} - \mathbf{x}(t^{(m-1)})\right\| + \|\mathcal{N} - \mathbf{\Phi}_\Delta\|_{L^\infty(D_\Delta)} \\
&\quad + \mathrm{e}^{L\Delta}\left\|\mathbf{y}^{(m-1)} - \mathbf{x}(t^{(m-1)})\right\| \\
&= \left(1 + \mathrm{e}^{L\Delta}\right)\mathcal{E}^{(m-1)} + \|\mathcal{N} - \mathbf{\Phi}_\Delta\|_{L^\infty(D_\Delta)},
\end{aligned}$$

where the Lipschitz continuity of the flow map, shown in (4.1), has been used in the last inequality. Recursively using the above estimate gives

$$\begin{aligned}
\mathcal{E}^{(m)} &\le \left(1 + \mathrm{e}^{L\Delta}\right)\mathcal{E}^{(m-1)} + \|\mathcal{N} - \mathbf{\Phi}_\Delta\|_{L^\infty(D_\Delta)} \\
&\le \left(1 + \mathrm{e}^{L\Delta}\right)^2 \mathcal{E}^{(m-2)} + \|\mathcal{N} - \mathbf{\Phi}_\Delta\|_{L^\infty(D_\Delta)}\left(1 + \left(1 + \mathrm{e}^{L\Delta}\right)\right) \\
&\le \cdots \\
&\le \left(1 + \mathrm{e}^{L\Delta}\right)^m \mathcal{E}^{(0)} + \|\mathcal{N} - \mathbf{\Phi}_\Delta\|_{L^\infty(D_\Delta)} \sum_{i=0}^{m-1} \left(1 + \mathrm{e}^{L\Delta}\right)^i.
\end{aligned}$$

The proof is complete. □

5. Numerical examples

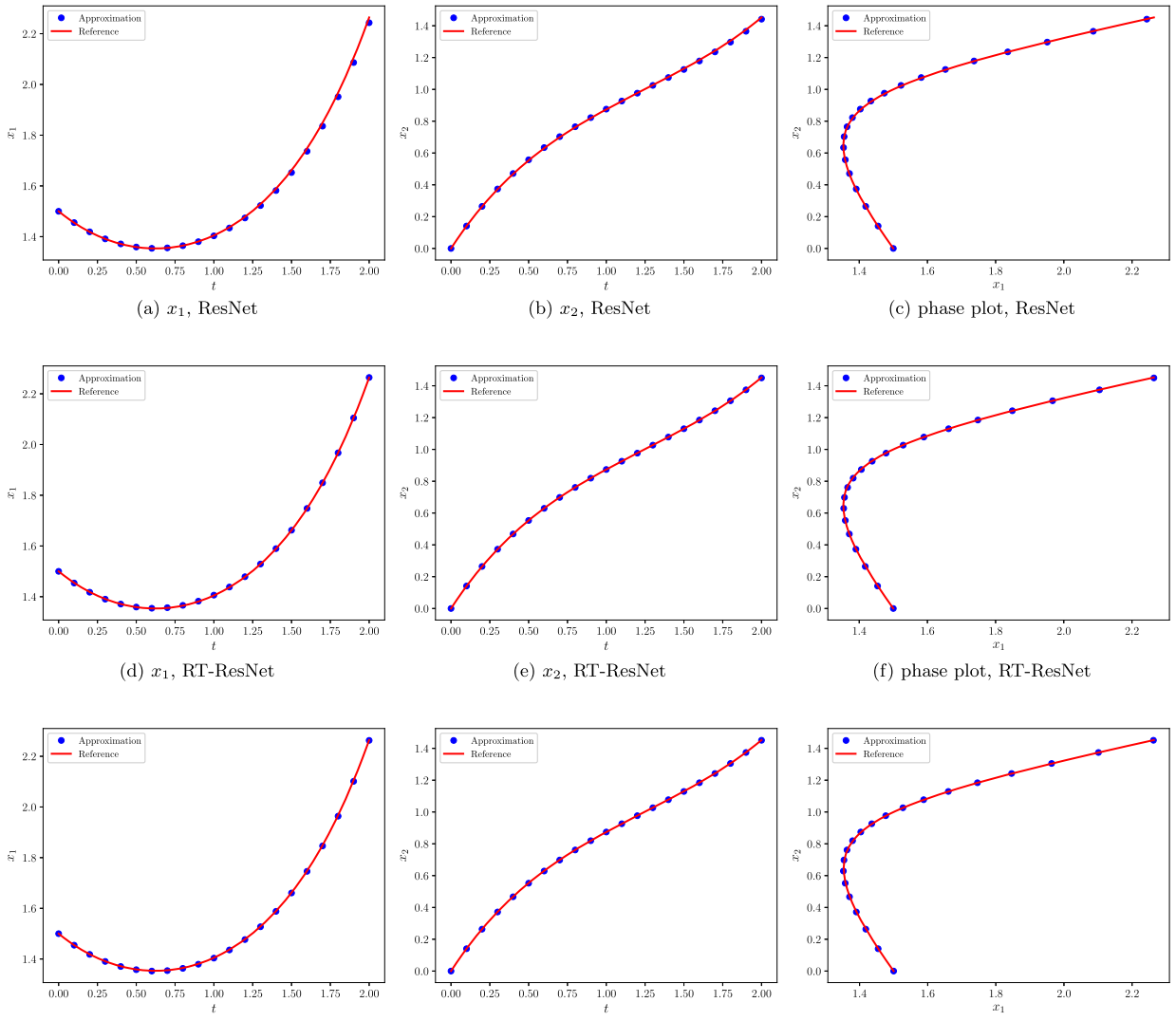
In this section we present numerical examples to verify the properties of the proposed methods. In all the examples, we generate the training data pairs $\{(\mathbf{z}_j^{(1)}, \mathbf{z}_j^{(2)}\}_{j=1}^J$ in the following way:

- Generate J points $\{\mathbf{z}_j^{(1)}\}_{j=1}^J$ from uniform distribution over a computational domain D. The domain D is a region in which we are interested in the solution behavior. It is typically chosen to be a hypercube prior to the computation.
- For each j, starting from $\mathbf{z}_j^{(1)}$, we march forward for a time lag Δ the underlying governing equation, using a highly accurate standard ODE solver, to generate $\mathbf{z}_j^{(2)}$. In our examples we set $\Delta = 0.1$.

In each example, we take 20 times as many data pairs as the number of model parameters. We remark that the time lag $\Delta = 0.1$ is relatively coarse and prevents accurate estimate of time derivatives via numerical differentiation. Since our proposed methods employ the integral form of the underlying equation, this difficulty is circumvented. The random sampling of the solution trajectories of length Δ follows from the work of [52], where it was established that such kind of dense sampling of short trajectories is highly effective for equation recovery.

All of our network models, ResNet, RT-ResNet, and RS-ResNet, are trained via the loss function (3.3) and by using the open-source Tensorflow library [1]. The training data set is divided into mini-batches of size 10. And we typically train the model for 500 epochs and reshuffle the training data in each epoch. All the weights are initialized randomly from Gaussian distributions and all the biases are initialized to be zeros.

After training the network models satisfactorily, using the data of $\Delta = 0.1$ time lag, we march the trained network models further forward in time and compare the results against the reference states, which are produced by high-order numerical solvers of the true underlying governing equations. We march the trained network systems up to $t \gg \Delta$ to examine their (relatively) long-term behaviors. For the two linear examples, we set $t = 2$; and for the two nonlinear examples, we set $t = 20$.



(a) x_1, ResNet (b) x_2, ResNet (c) phase plot, ResNet

(d) x_1, RT-ResNet (e) x_2, RT-ResNet (f) phase plot, RT-ResNet

(g) x_1, RS-ResNet (h) x_2, RS-ResNet (i) phase plot, RS-ResNet

Fig. 5.1. Trajectory and phase plots for the Example 1 with $\mathbf{x}_0 = (1.5, 0)$ for $t \in [0, 2]$. Top row: one-step ResNet model; Middle row: Multi-step RT-ResNet model; Bottom row: Multi-step RS-ResNet model.

5.1. Linear ODEs

We first study two linear ODE systems, as textbook examples. In both examples, our one-step ResNet method has 3 hidden layers, each of which has 30 neurons. For the multi-step RT-ResNet and RS-ResNet methods, they both have 3 ResNet blocks ($K = 3$), each of which contains 3 hidden layers with 20 neurons in each layer.

Example 1

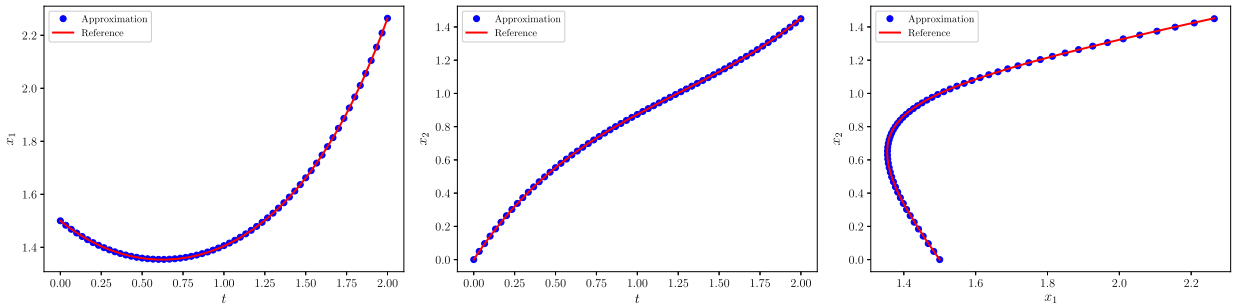
We first consider the following two-dimensional linear ODE with $\mathbf{x} = (x_1, x_2)$

$$\begin{cases} \dot{x}_1 = x_1 + x_2 - 2, \\ \dot{x}_2 = x_1 - x_2, \end{cases} \tag{5.1}$$

where $\dot{x}$ represents the time derivative $\frac{d}{dt}x$. The computational domain D is taken to be $D = [0, 2]^2$.

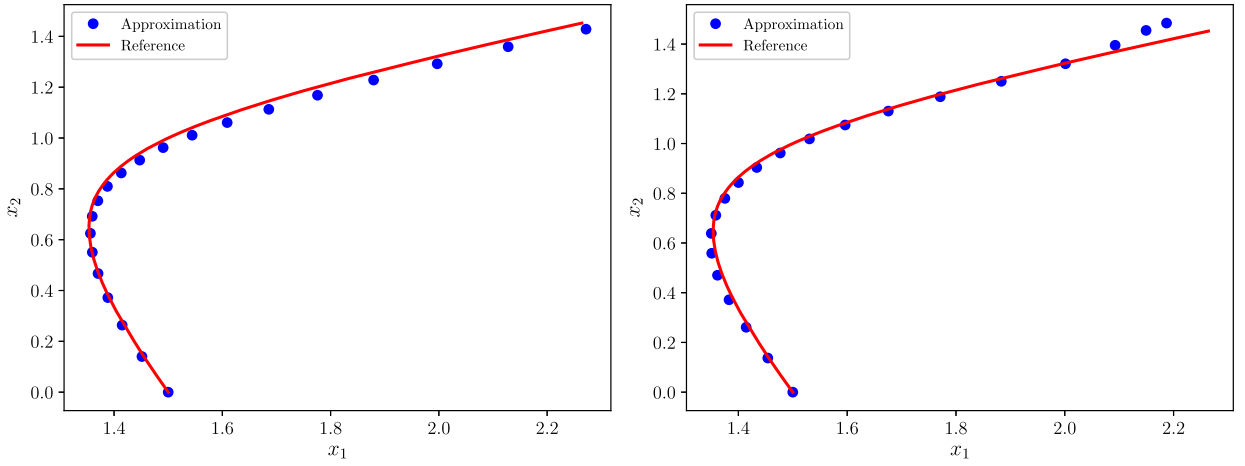
Upon training the three network models satisfactorily, using the $\Delta = 0.1$ data pairs, we march the trained models further in time up to $t = 2$. In Fig. 5.1, we show the plots for the trajectories of both x_1 and x_2 as well as the portrait on the (x_1, x_2) phase plane. We observe that all three network models produce accurate prediction results for time up to $t = 2$.

As discussed in Section 3.2, the multi-step RT-ResNet method is able to produce an approximation over a smaller time step $\delta = \Delta/K$, which in this case is $\delta = 1/30$ (with $K = 3$). The trained RT-ResNet model then allows us to produce predic-



(a) x_1 (b) x_2 (c) phase plot

Fig. 5.2. Trajectory and phase plots for Example 1 with $\mathbf{x}_0 = (1.5, 0)$ using RT-ResNet model with $K = 3$. The solutions are marched into over time step $\delta = \Delta/K = 1/30$.



(a) with 2% noise (b) with 5% noise

Fig. 5.3. Phase plots for Example 1 with $\mathbf{x}_0 = (1.5, 0)$. One-step ResNets. The training data contains 2% (left) and 5% (right) noise, respectively.

tions over the finer time step δ. In Fig. 5.2, we show the time marching of the trained RT-ResNet model for up to $t = 2$ using the smaller time step δ. The results again agree very well with the reference solution. This demonstrates the capability of RT-ResNet – it allows us to produce accurate predictions with a resolution higher than that of the given data, i.e., $\delta < \Delta$. On the other hand, our numerical tests also reveal that the training of RT-ResNet with $K > 1$ becomes more involving – more training data are typically required and convergence can be slower, compared to the training of the one-step ResNet method. Similar behavior is also observed in multi-step RS-ResNet method with $K > 1$. The development of efficient training procedures for multi-step RT-ResNet and RS-ResNet methods is necessary and will be pursued in a future work.

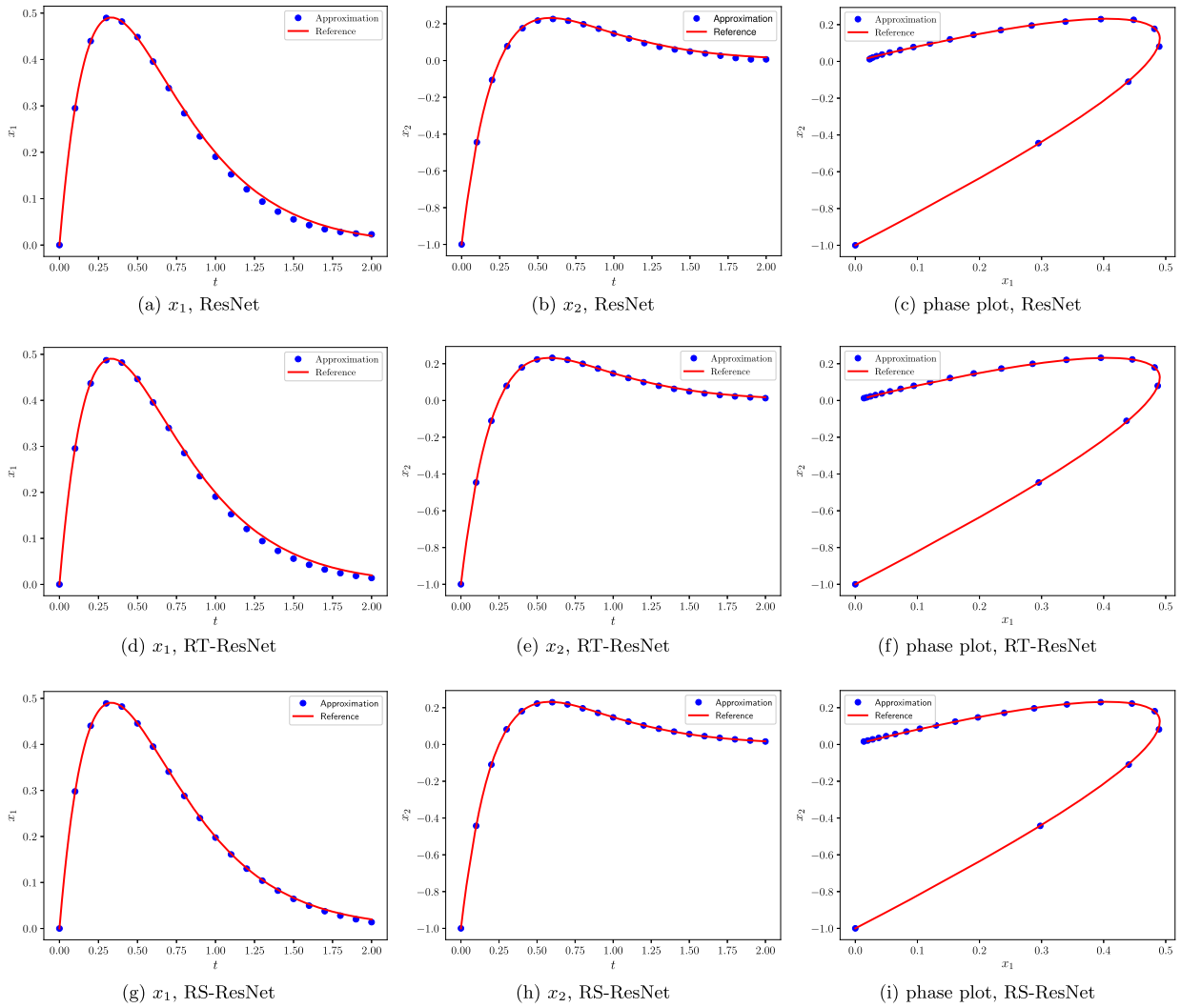
We then consider the case of noisy data. The data pairs are set as $\{\mathbf{z}_j^{(1)}(1+\epsilon_j^{(1)}), \mathbf{z}_j^{(2)}(1+\epsilon_j^{(2)})\}_{j=1}^J$, where the relatively noise levels $\epsilon_j^{(1)}$ and $\epsilon_j^{(2)}$ are drawn from uniform distribution over $[0, \eta]$. In the following experiments we set $\eta = 0.02$ and 0.05 for demonstrative purpose. In Fig. 5.3, we show the phase plots generated by the one-step ResNet. Since our NN models are based on the integral form of the ODE, they tolerate the training noise quite well. As the noise level increases, the NN prediction deviates more from the exact dynamics, while the main structure of the solution is still well captured. On the other hand, since the time lag is $\Delta = 0.1$, this noisy data case is certainly not amenable to the standard equation recovery approaches requiring time derivative computations.

Example 2

We now consider another linear ODE system:

$$\begin{cases} \dot{x}_1 = x_1 - 4x_2, \\ \dot{x}_2 = 4x_1 - 7x_2. \end{cases} \tag{5.2}$$

The numerical results for the three trained network models are presented in Fig. 5.4. Again, we show the prediction results of the trained models for up to $t = 2$. While all predictions agree well with the reference solution, one can visually see that the RS-ResNet model is more accurate than the RT-ResNet model, which in turn is more accurate than the one-step ResNet



(a) x_1, ResNet (b) x_2, ResNet (c) phase plot, ResNet

(d) x_1, RT-ResNet (e) x_2, RT-ResNet (f) phase plot, RT-ResNet

(g) x_1, RS-ResNet (h) x_2, RS-ResNet (i) phase plot, RS-ResNet

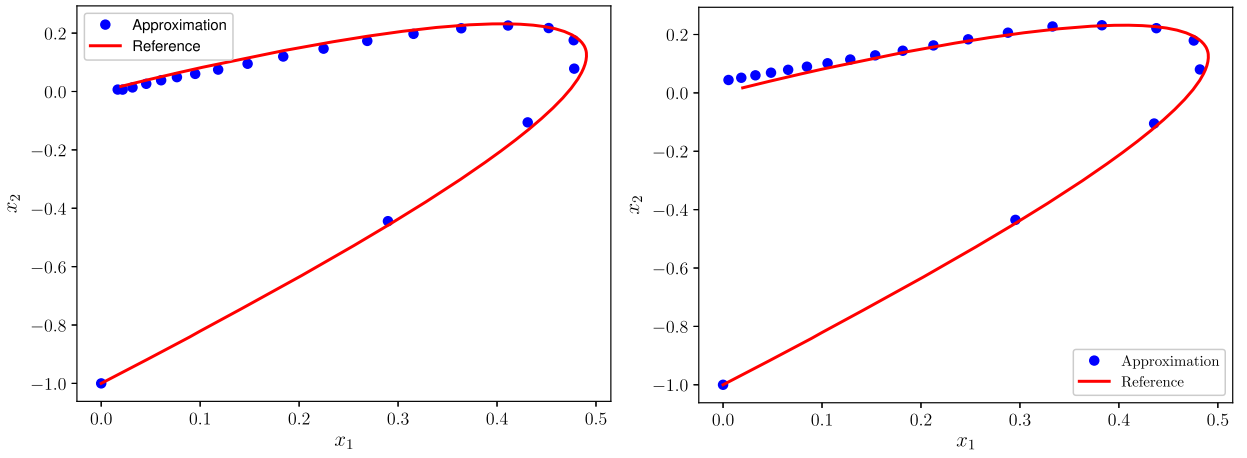
Fig. 5.4. Trajectory and phase plots for the Example 2 with $\mathbf{x}_0 = (0, -1)$. Top row: one-step ResNet model; Middle row: Multi-step RT-ResNet model; Bottom row: Multi-step RS-ResNet model.

model. This is expected, as the multi-step methods should be more accurate than the one-step method (ResNet), and the RS-ResNet should be even more accurate due to its adaptive nature. On the other hand, RS-ResNet introduces larger number of parameters and induces more training cost. For a given problem, the balance between accuracy and training cost should be considered by the user.

In Fig. 5.5, we present the results for noisy data case. The noisy data are generated in the same manner as in Example 1, and the simulation results are obtained by the one-step ResNet method. Again, we observe that the proposed model performs robustly in the presence of data noise.

5.2. Nonlinear ODEs

We now consider two nonlinear problems. The first one is the well known damped pendulum problem, and the second one is an nonlinear differential-algebraic equation (DAE) for modelling a generic toggle ([17]). In both examples, our one-step ResNet model has 2 hidden layers, each of which has 40 neurons. Our multi-step RT-ResNet and RS-ResNet models both have 3 of the same ResNet blocks ($K = 3$). Again, our training data are collected over $\Delta = 0.1$ time lag. We produce predictions of the trained model over time for up to $t = 20$ and compare the results against the reference solutions.



(a) with 2% noise (b) with 5% noise

Fig. 5.5. Phase plots for Example 2 with $\mathbf{x}_0 = (0, -1)$. One-step ResNets. The training data contains 2% (left) and 5% (right) relative noise, respectively.

Example 3: Damped pendulum

The first nonlinear example we are considering is the following damped pendulum problem,

$$\begin{cases} \dot{x}_1 = x_2, \\ \dot{x}_2 = -\alpha x_2 - \beta \sin x_1, \end{cases}$$

where $\alpha = 8.91$ and $\beta = 0.2$. The computational domain is $D = [-\pi, \pi] \times [-2\pi, 2\pi]$. In Fig. 5.6, we present the prediction results by the three network models, starting from the initial condition $\mathbf{x}_0 = (-1.193, -3.876)$ and for time up to $t = 20$. We observe excellent agreements between the network models and the reference solution.

Example 4: Genetic toggle switch

We now consider a system of nonlinear differential-algebraic equations (DAE), which are used to model a genetic toggle switch in *Escherichia coli* ([17]). It is composed of two repressors and two constitutive promoters, where each promoter is inhibited by the represssor that is transcribed by the opposing promoter. Details of experimental measurement can be found in [10]. This system of equations are as follows,

$$\begin{cases} \dot{x}_1 = \frac{\alpha_1}{1+x_2^{\beta}} - x_1, \\ \dot{x}_2 = \frac{\alpha_2}{1+z^{\gamma}} - x_2, \\ z = \frac{x_1}{(1+[\mathrm{IPTG}]/K)^{\eta}}. \end{cases}$$

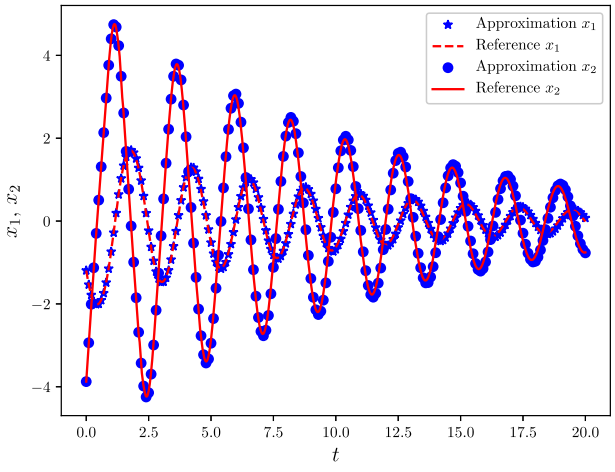
In this system, the components x_1 and x_2 denote the concentration of the two repressors. The parameters α_1 and α_2 are the effective rates of the synthesis of the repressors; β and γ represent cooperativity of repression of the two promoters, respectively; [IPTG] is the concentration of IPTG, the chemical compound that induces the switch; and K is the dissociation constant of IPTG.

In the following numerical experiment, we take $\alpha_1 = 156.25$, $\alpha_2 = 15.6$, $\gamma = 1$, $\beta = 2.5$, $K = 2.9618 \times 10^{-5}$ and $[\mathrm{IPDG}] = 10^{-5}$. We consider the computational domain $D = [0, 20]^2$.

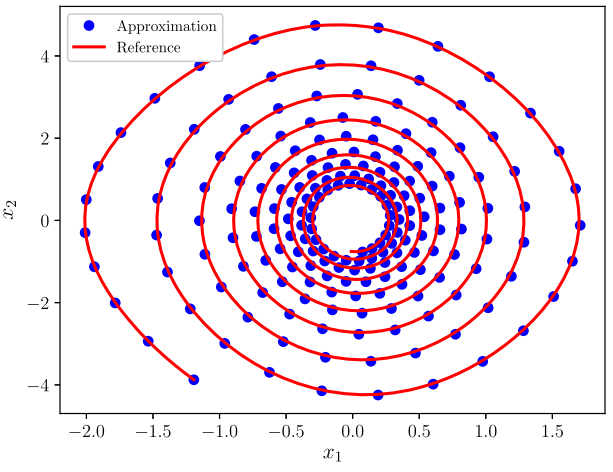
In Fig. 5.7 we present the prediction results generated by the ResNet, the RT-ResNet and the RS-ResNet, for time up to $t = 20$. The initial condition is $\mathbf{x}_0 = (19, 17)$. Again, all these three models produce accurate approximations, even for such a long-time simulation.

6. Conclusion

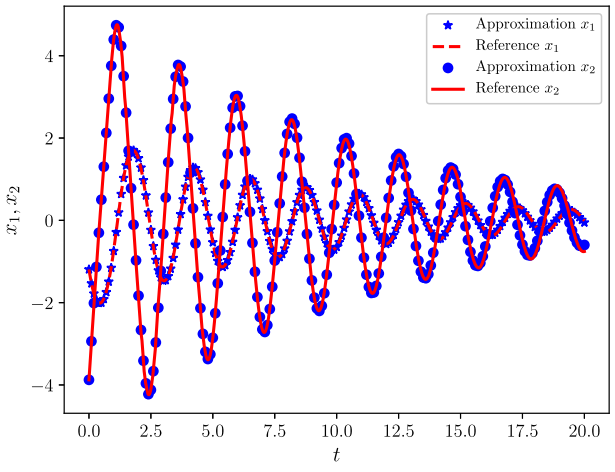
We presented several deep neural network (DNN) structures for approximating unknown dynamical systems using trajectory data. The DNN structures are based on residual network (ResNet), which is a one-step method exact time integrator. Two multi-step variations were presented. One is recurrent ResNet (RT-ResNet) and the other one is recursive ResNet (RS-ResNet). Upon successful training, the methods produce discrete dynamical systems that approximate the underlying unknown governing equations. All methods are based on integral form of the underlying system. Consequently, their constructions do not require time derivatives of the trajectory data and can work with coarsely distributed data as well. We presented the construction details of the methods, their theoretical justifications, and used several examples to demonstrate the effectiveness of the methods.



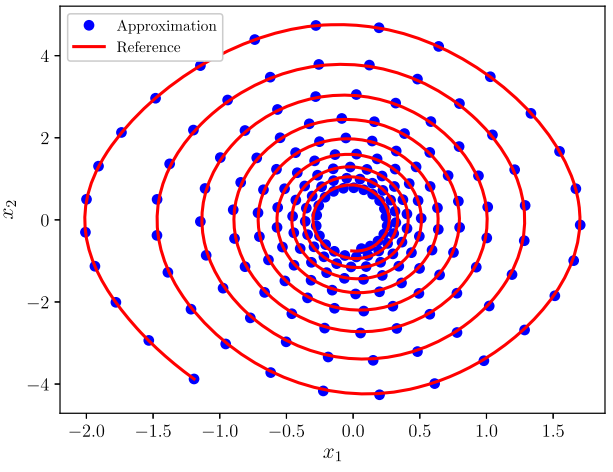
(a) x_1 and x_2, ResNet



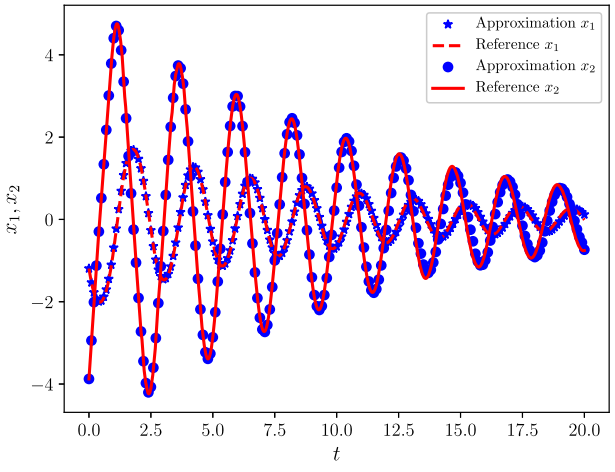
(b) phase plot, ResNet



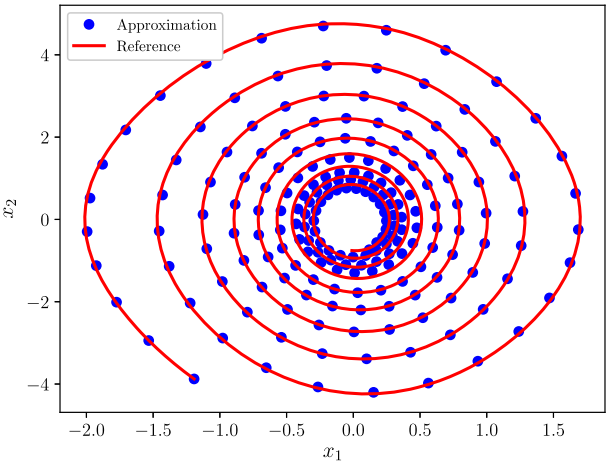
(c) x_1 and x_2, RT-ResNet



(d) phase plot, RT-ResNet

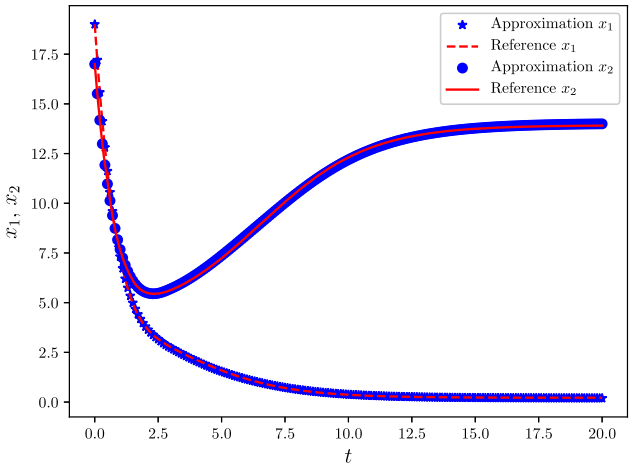


(e) x_1 and x_2, RS-ResNet

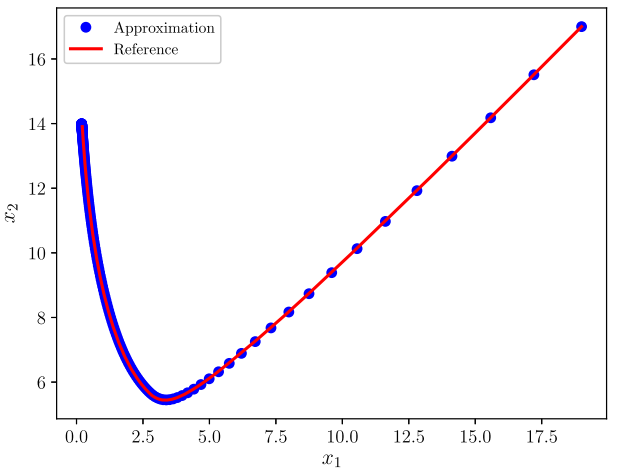


(f) phase plot, RS-ResNet

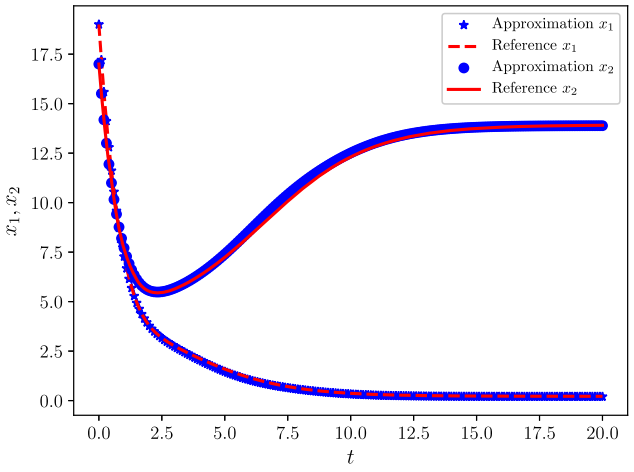
Fig. 5.6. Trajectory and phase plots for the Example 3 with $\mathbf{x}_0 = (-1.193, -3.876)$. Top row: one-step ResNet model; Middle row: Multi-step RT-ResNet model; Bottom row: Multi-step RS-ResNet model. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)



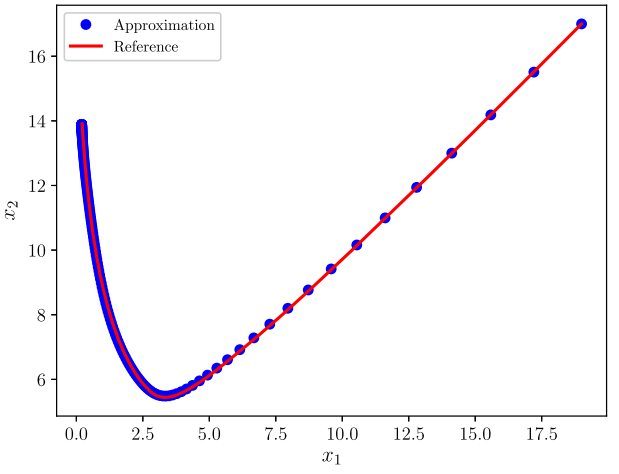
(a) x_1 and x_2, ResNet



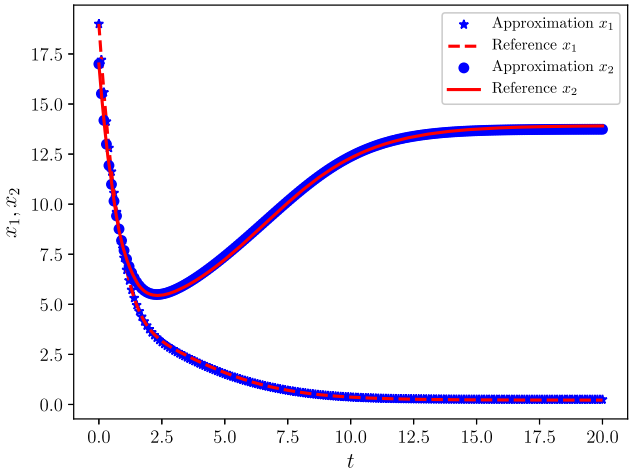
(b) phase plot, ResNet



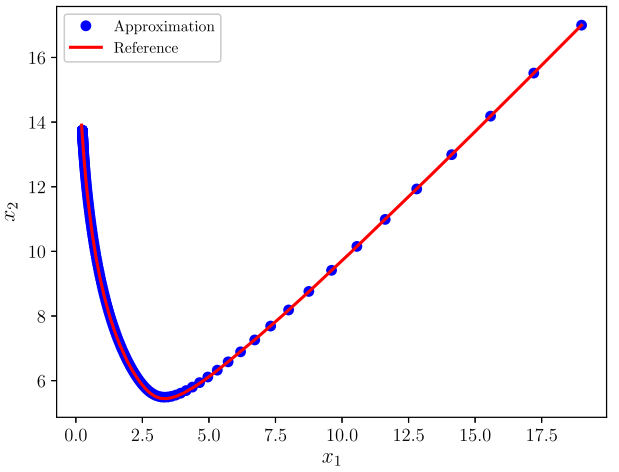
(c) x_1 and x_2, RT-ResNet



(d) phase plot, RT-ResNet



(e) x_1 and x_2, RS-ResNet



(f) phase plot, RS-ResNet

Fig. 5.7. Trajectory and the phase plots for the Example 4 with $\mathbf{x}_0 = (19, 17)$. Top row: one-step ResNet model; Middle row: Multi-step RT-ResNet model; Bottom row: Multi-step RS-ResNet model.

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] A.R. Barron, Universal approximation bounds for superpositions of a sigmoidal function, IEEE Trans. Inf. Theory 39 (1993) 930–945.

[3] P.L. Bartlett, S.N. Evans, P.M. Long, Representing smooth functions as compositions of near-identity functions with implications for deep network optimization, preprint, arXiv:1804.05012, 2018.

[4] M. Bianchini, F. Scarselli, On the complexity of neural network classifiers: a comparison between shallow and deep architectures, IEEE Trans. Neural Netw. Learn. Syst. 25 (2014) 1553–1565.

[5] J. Bongard, H. Lipson, Automated reverse engineering of nonlinear dynamical systems, Proc. Natl. Acad. Sci. USA 104 (2007) 9943–9948.

[6] 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).

[7] S.L. Brunton, J.L. Proctor, J.N. Kutz, Discovering governing equations from data by sparse identification of nonlinear dynamical systems, Proc. Natl. Acad. Sci. USA 113 (2016) 3932–3937.

[8] S. Chan, A. Elsheikh, A machine learning approach for efficient uncertainty quantification using multiscale methods, J. Comput. Phys. 354 (2018) 494–511.

[9] B. Chang, L. Meng, E. Haber, F. Tung, D. Begert, Multi-level residual networks from dynamical systems view, in: International Conference on Learning Representations, 2018.

[10] R. Chartrand, Numerical differentiation of noisy, nonsmooth data, ISRN Appl. Math. 2011 (2011).

[11] R.T.Q. Chen, Y. Rubanova, J. Bettencourt, D. Duvenaud, Neural ordinary differential equations, preprint, arXiv:1806.07366, 2018.

[12] B.C. Daniels, I. Nemenman, Automated adaptive inference of phenomenological dynamical models, Nat. Commun. 6 (2015).

[13] B.C. Daniels, I. Nemenman, Efficient inference of parsimonious phenomenological models of cellular dynamics using S-systems and alternating regression, PLoS ONE 10 (2015) e0119821.

[14] K.-L. Du, M. Swamy, Neural Networks and Statistical Learning, Springer-Verlag, 2014.

[15] W. E, B. Engquist, Z. Huang, Heterogeneous multiscale method: a general methodology for multiscale modeling, Phys. Rev. B 67 (2003) 092101.

[16] R. Eldan, O. Shamir, The power of depth for feedforward neural networks, in: Conference on Learning Theory, 2016, pp. 907–940.

[17] T.S. Gardner, C.R. Cantor, J.J. Collins, Construction of a genetic toggle switch in escherichia coli, Nature 403 (2000) 339.

[18] D. Giannakis, A.J. Majda, Nonlinear Laplacian spectral analysis for time series with intermittency and low-frequency variability, Proc. Natl. Acad. Sci. USA 109 (2012) 2222–2227.

[19] R. Gonzalez-Garcia, R. Rico-Martinez, I.G. Kevrekidis, Identification of distributed parameter systems: a neural net based approach, Comput. Chem. Eng. 22 (1998) S965–S968.

[20] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.

[21] 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.

[22] J. Hesthaven, S. Ubbiali, Non-intrusive reduced order modeling of nonlinear problems using neural networks, J. Comput. Phys. 363 (2018) 55–78.

[23] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural Netw. 4 (1991) 251–257.

[24] I.G. Kevrekidis, C.W. Gear, J.M. Hyman, P.G. Kevrekidid, O. Runborg, C. Theodoropoulos, et al., Equation-free, coarse-grained multiscale computation: enabling mocroscopic simulators to perform system-level analysis, Commun. Math. Sci. 1 (2003) 715–762.

[25] Y. Khoo, J. Lu, L. Ying, Solving parametric pde problems with artificial neural networks, preprint, arXiv:1707.03351, 2018.

[26] M. Leshno, V.Y. Lin, A. Pinkus, S. Schocken, Multilayer feedforward networks with a nonpolynomial activation function can approximate any function, Neural Netw. 6 (1993) 861–867.

[27] Z. Long, Y. Lu, X. Ma, B. Dong, PDE-Net: learning PDEs from data, preprint, arXiv:1710.09668, 2017.

[28] 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. Lond., Ser. A, Math. Phys. Eng. Sci. 473 (2017).

[29] A. Mardt, L. Pasquali, H. Wu, F. Noe, VAMPnets for deep learning of molecular kinetics, Nat. Commun. 9 (2018) 5.

[30] G.F. Montufar, R. Pascanu, K. Cho, Y. Bengio, On the number of linear regions of deep neural networks, in: Advances in Neural Information Processing Systems, 2014, pp. 2924–2932.

[31] A. Pinkus, Approximation theory of the MLP model in neural networks, Acta Numer. 8 (1999) 143–195.

[32] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, Q. Liao, Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review, Int. J. Autom. Comput. 14 (2017) 503–519.

[33] M. Raissi, Deep hidden physics models: deep learning of nonlinear partial differential equations, preprint, arXiv:1801.06637, 2018.

[34] M. Raissi, G.E. Karniadakis, Hidden physics models: machine learning of nonlinear partial differential equations, J. Comput. Phys. 357 (2018) 125–141.

[35] M. Raissi, P. Perdikaris, G.E. Karniadakis, Machine learning of linear differential equations using gaussian processes, J. Comput. Phys. 348 (2017) 683–693.

[36] M. Raissi, P. Perdikaris, G.E. Karniadakis, Multistep neural networks for data-driven discovery of nonlinear dynamical systems, preprint, arXiv:1801.01236, 2018.

[37] D. Ray, J. Hesthaven, An artificial neural network as a troubled-cell indicator, J. Comput. Phys. 367 (2018) 166–191.

[38] S.H. Rudy, S.L. Brunton, J.L. Proctor, J.N. Kutz, Data-driven discovery of partial differential equations, Sci. Adv. 3 (2017) e1602614.

[39] S.H. Rudy, J.N. Kutz, S.L. Brunton, Deep learning of dynamics and signal-noise decomposition with time-stepping constraints, preprint, arXiv:1808.02578, 2018.

[40] H. Schaeffer, Learning partial differential equations via data discovery and sparse optimization, Proc. R. Soc. Lond., Ser. A, Math. Phys. Eng. Sci. 473 (2017).

[41] H. Schaeffer, S.G. McCalla, Sparse model selection via integral terms, Phys. Rev. E 96 (2017) 023302.

[42] J. Schmidhuber, Deep learning in neural networks: an overview, Neural Netw. 61 (2015) 85–117.

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

[44] M.D. Schmidt, R.R. Vallabhajosyula, J.W. Jenkins, J.E. Hood, A.S. Soni, J.P. Wikswo, H. Lipson, Automated refinement and inference of analytical models for metabolic networks, Phys. Biol. 8 (2011) 055011.

[45] A. Stuart, A.R. Humphries, Dynamical Systems and Numerical Analysis, vol. 2, Cambridge University Press, 1998.

[46] G. Sugihara, R. May, H. Ye, C. Hsieh, E. Deyle, M. Fogarty, S. Munch, Detecting causality in complex ecosystems, Science 338 (2012) 496–500.

[47] R. Tibshirani, Regression shrinkage and selection via the lasso, J. R. Stat. Soc., Ser. B, Methodol. (1996) 267–288.

[48] G. Tran, R. Ward, Exact recovery of chaotic systems from highly corrupted data, Multiscale Model. Simul. 15 (2017) 1108–1129.

[49] R. Tripathy, I. Bilionis, Deep UQ: learning deep neural network surrogate model for high dimensional uncertainty quantification, J. Comput. Phys. 375 (2018) 565–588.

[50] H.U. Voss, P. Kolodner, M. Abel, J. Kurths, Amplitude equations from spatiotemporal binary-fluid convection data, Phys. Rev. Lett. 83 (1999) 3422.
[51] Y. Wang, S.W. Cheung, E.T. Chung, Y. Efendiev, M. Wang, Deep multiscale model learning, preprint, arXiv:1806.04830, 2018.
[52] K. Wu, D. Xiu, Numerical aspects for approximating governing equations using data, J. Comput. Phys. (2019), in press.
[53] H. Ye, R.J. Beamish, S.M. Glaser, S.C.H. Grant, C. Hsieh, L.J. Richards, J.T. Schnute, G. Sugihara, Equation-free mechanistic ecosystem forecasting using empirical dynamic modeling, Proc. Natl. Acad. Sci. USA 112 (2015) E1569–E1576.
[54] Y. Zhu, N. Zabaras, Bayesian deep convolutional encoder-decoder networks for surrogate modeling and uncertainty quantification, J. Comput. Phys. 366 (2018) 415–447.