Journal of Computational Physics 438 (2021) 110362

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

On generalized residual network for deep learning of unknown dynamical systems

Zhen Chen, Dongbin Xiu *,[1]

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

ARTICLE INFO

Article history:
Available online 19 April 2021

Keywords:
Deep neural network
Residual network
Governing equation discovery
Model correction

ABSTRACT

We present a general numerical approach for learning unknown dynamical systems using deep neural networks (DNNs). Our method is built upon recent studies that identified residual network (ResNet) as an effective neural network learning structure. In this paper, we present a generalized ResNet framework and broadly define "residue" as the discrepancy between observation data and prediction made by another model, which can be an existing coarse model or reduced order model. In this case, the generalized ResNet serves as a model correction to the existing model and recovers the unresolved/missing dynamics. When an existing coarse model is not available, we present numerical strategies for fast creation of coarse models, to be used in conjunction with the generalized ResNet. These coarse models are constructed using the same data set and thus do not require additional resource. The generalized ResNet is capable of learning the underlying unknown dynamics and producing predictions with accuracy higher than the standard ResNet structure. This is demonstrated via several numerical examples, including long-term prediction of a chaotic system.

1. Introduction

There has been a surge of interest in developing algorithms to recover unknown governing equations via observation data. Most efforts focus on recovering unknown system of ordinary differential equations, i.e., dynamical systems. Some (relatively) earlier efforts seek to construct certain optimal linear approximations for the unknown system. These include dynamic model decomposition (DMD) ([39]) and its variants for nonlinear systems using Koopman theory, cf., [3,16,15]. More recent efforts cast the problem into an approximation problem, where the unknown governing equations are treated as target functions 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., [41]) from a large set of dictionaries; see, for example, [4]. Studies have been conducted to deal with noises in data [4,37,11], corruptions in data [42], limited data [38], partial differential equations [33,36], etc. Variations of the approaches have been developed in conjunction with other methods such as model selection approach [19], Koopman theory [3], Gaussian process regression [28,27], and expectation-maximization approach [20], to name a few. Methods using standard basis functions and without requiring exact recovery were also developed for dynamical systems [48] and Hamiltonian systems [47].

* Corresponding author.
E-mail addresses: chen.7168@osu.edu (Z. Chen), xiu.16@osu.edu (D. Xiu).

[1] Funding: This work was partially supported by AFOSR FA9550-18-1-0102.

https://doi.org/10.1016/j.jcp.2021.110362
0021-9991/

The use of modern machine learning techniques, particularly deep neural networks (DNNs), offers a new line of approaches for the task. DNN structures have been developed to recover ordinary differential equations (ODEs) [31,25,34] and partial differential equations (PDEs) [18,29,30,26,17,40]. It was shown that residual network (ResNet) is particularly suitable for equation recovery, in the sense that it can be an exact integrator [25]. Neural networks have also been explored for other aspects of scientific computing, including reduced order modeling [8,22], solution of conservation laws [32,44], multiphase flow simulation [46], high-dimensional PDEs [6,14], uncertainty quantification [5,43,49,12], etc.

The focus and contribution of this paper is on generalization of ResNet (gResNet) structure for recovering unknown dynamical systems. In the standard ResNet method for equation recovery, residue is defined as the difference between the output data and input data, and a deep neural network (DNN) is used to model the residue. (For more detail, see [25].) In this paper, we broaden the concept of residue and broadly define it as the difference between the output data and *the predictive output made by another model*, which shall be referred to as "prior prediction" hereafter. We then use a standard feedforward neural network to model this generalized residue and to construct the final prediction, hereafter referred to as "posteriori prediction", by taking into account of the prior prediction.[2] The gResNet structure can then be viewed as a model correction method for the prior predictive model. The prior model can be an existing coarse model, reduced order model, empirical model, etc. The DNN in the gResNet can be considered a model for the discrepancy between the prior model and the true model. We remark that model correction/calibration is an ongoing research topic, where several methods exist. See, for example, [2,7,9,10,13,35,45,24] and the references therein. Our method here represents a new and drastically different approach, via the use of DNN, to this line of study. It is also straightforward to see that the generalized ResNet (gResNet) includes the standard ResNet as a special case, in the sense that in the standard ResNet the prior model takes the trivial form of an identity operator, i.e., the prior prediction is the same as the input data.

In many practical situations, one may not possess a prior model and only has access to data. In this case, we propose a number of numerical strategies to create a prior model using the same data set. In particular, we discuss two approaches. The first one seeks to construct an affine approximation for the underlying system as the prior model. This is an extension of the well known DMD (dynamic mode decomposition) method, which utilizes linear approximation ([39]). The use of affine approximation, which has a constant vector term in addition to linear transformation, allows one to model possible non-homogeneous terms in the underlying dynamical system. This proves to be more flexible in practice. The other method seeks to construct the prior model as a nonlinear approximation using a single hidden layer NN. This is an extension of the affine approximation modeling, which is a linear procedure. Both approaches can produce prior models in efficient manner and do not increase the overall computational cost of the entire modeling process. Note that the emphasis of the paper is on the gResNet framework, and the discussion of the two approaches for prior model construction is to provide some viable choices. In practice, one is free to choose any other suitable (for the given problem) method to construct the prior model.

It is also worth noting that the proposed gResNet learning method, along with its predecessor ResNet learning method [25], does not seek to directly recover the underlying governing equations. Instead, these methods construct an approximation to the flow map of the unknown dynamical system and use the approximate flow map to conduct system prediction.

This paper is organized as follows. After the basic setup in Section 2, we present the main modeling approach in Section 3. We then present, in Section 4, a set of numerical examples of both linear and nonlinear system, including a differential-algebraic system and a chaotic system, to demonstrate the effectiveness of the proposed methods.

2. Setup and preliminaries

Let us consider an autonomous dynamical system

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

where $\mathbf{x} \in \mathbb{R}^n$, $n \geq 1$, are the state variables. Note that the dimension n can be large, especially when the system is obtained via a spatial discretization of a PDE system. In this paper, we assume the form of the governing equations $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ is unknown. What is available is measurement data of the solution states. Let $t_0 < t_1 < \cdots < t_K$ be a sequence of time instances. We use

$$\mathbf{x}_k^{(i)} = \mathbf{x}(t_k; \mathbf{x}_0^{(i)}, t_0), \qquad k = 1, \ldots, K, \quad i = 1, \ldots, I_{traj}, \tag{2.2}$$

to stand for the i-th trajectory solution state at time instance t_k, originated from the initial state $\mathbf{x}_0^{(i)}$ at t_0, for a total number of I_{traj} trajectories. Note that the data can also contain measurement noises, which are usually modeled as random variables. Our goal is then to create an accurate predictive model for the unknown dynamics by using the solution state data.

[2] *Prior prediction* refers to the prediction made by an existing model; *Posteriori prediction* refers to the prediction made by modifying the prior prediction. The use of *prior* and *posteriori* has no direct connection with Bayesian statistics.

2.1. Flow map and data pairing

While many of the existing work seek to construct a system $d\mathbf{x}/dt = \widetilde{f}(\mathbf{x})$ as an approximation to the unknown system (2.1), we here adopt a different approach, developed in [25]. This approach does not directly approximate the right-hand-side of the system (2.1). Instead, it seeks to approximate the flow map of the underlying system.

Let $\mathbf{\Phi}_s : \mathbb{R}^n \to \mathbb{R}^n$ be the flow map of system (2.1), whose solution follows

$$\mathbf{x}(t; \mathbf{x}_0, t_0) = \mathbf{\Phi}_{t-t_0}(\mathbf{x}_0). \tag{2.3}$$

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. Also, $\mathbf{\Phi}_{s+r} = \mathbf{\Phi}_s \circ \mathbf{\Phi}_r$. The flow map completely determines the evolution of the solution from one state to another state at a different time. Recovery of the flow map allows one to conduct prediction of the system via recursive use of the flow map. Let $\Delta_k = t_k - t_{k-1}$, $k = 1, \dots$ be the time lags in the time instances. We then organize the solution state data in (2.2) into pairs, separated by the time lag Δ_k,

$$\left\{\mathbf{x}_{k-1}^{(i)}, \mathbf{x}_k^{(i)}\right\}, \qquad k = 1, \dots, K, \quad i = 1, \dots, I_{traj}. \tag{2.4}$$

For notational convenience, hereafter we assume $\Delta_k = \Delta$ is a constant for all k. We then denote the entire data set as

$$\mathcal{S} = \{(\mathbf{x}_j^{(1)}, \mathbf{x}_j^{(2)}) : j = 1, \dots, J\}, \tag{2.5}$$

where $J = K \times I_{traj}$ is the total number of data pairs. For each j-th pair, $\mathbf{x}_j^{(1)}$ is the "initial state", $\mathbf{x}_j^{(2)}$ is the "end state", and the two states are separated by the time lag Δ. In the noiseless case, the two states are governed by the (unknown) flow map such that

$$\mathbf{x}_j^{(2)} = \mathbf{\Phi}_\Delta(\mathbf{x}_j^{(1)}). \tag{2.6}$$

2.2. ResNet modeling of flow map

In [25], a method was proposed to discover the unknown dynamical system (2.1) via numerically approximating its underlying flow map. This is accomplished by using the data pairs (2.5) to approximate the unknown flow map over the discrete time lag Δ in (2.6). Once this Δ-flow map is constructed, it can be recursively applied to conduct solution prediction over much longer time horizon.

Moreover, the work of [25] also proposed to utilize residual network (ResNet) to conduct deep learning of the Δ-flow map. While the standard feedforward deep neural networks (DNNs) utilize multiple hidden layers to approximate input-output maps, ResNet applies the identity operator on the input data and superimposes it on the neural network outputs. This effectively creates a DNN model for the "residue" of the input-output data.

Let $\mathcal{N}(\cdot; \Theta) : \mathbb{R}^n \to \mathbb{R}^n$ be the operator from a standard fully connected feedforward neural network, where Θ denotes its parameter set. Its corresponding ResNet then creates a mapping

$$\mathbf{N}(\cdot; \Theta) = \mathcal{I} + \mathcal{N}(\cdot; \Theta), \tag{2.7}$$

where $\mathcal{I}$ is the identity operator. By applying this structure to the data set (2.5) and defining a loss function in the form of

$$L(\Theta) = \frac{1}{J} \sum_{j=1}^{J} \left\| \mathbf{x}_j^{(2)} - \mathbf{N}(\mathbf{x}_j^{(1)}; \Theta) \right\|^2, \tag{2.8}$$

one can train a ResNet model

$$\mathbf{x}_j^{(2)} \approx \mathbf{x}_j^{(1)} + \mathcal{N}(\mathbf{x}_j^{(1)}; \Theta^*), \qquad j = 1, \dots, J, \tag{2.9}$$

where Θ^* is the network parameter set after successful training. Upon obtaining the trained network model, one can recursively apply the model to conduct system prediction in the following form,

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathcal{N}(\mathbf{x}_k; \Theta^*), \qquad k = 0, 1, \dots, \tag{2.10}$$

for a given initial condition $\mathbf{x}_0$. Even though the form of (2.10) resembles Euler forward time stepper, it was shown in [25] that this model is an exact time integrator, in the sense that there is no error explicitly associated with the time step Δ.

3. Generalized ResNet (gResNet) modeling

In this section we present generalized ResNet (gResNet) method, as an extension to the ResNet method developed in [25]. We first present the general approach of the gResNet method and then discuss a few practical options.

3.1. General approach of gResNet

Let $\mathcal{L}: \mathbb{R}^n \to \mathbb{R}^n$ be an operator such that

$$\mathbf{x}(t_{k+1}) \approx \mathcal{L}(\mathbf{x}(t_k)). \tag{3.1}$$

The proposed gResNet mapping takes the following form

$$\mathbf{N}(\cdot;\Theta) = \mathcal{L} + \mathcal{N}(\cdot;\Theta), \tag{3.2}$$

where $\mathcal{N}(\cdot;\Theta)$ is the operator associated with a standard fully connected DNN. The training of the gResNet is conducted by using the data set (2.5) to minimize the following loss function

$$L(\Theta) = \frac{1}{J}\sum_{j=1}^{J}\left\|\mathbf{x}_j^{(2)} - \mathbf{N}(\mathbf{x}_j^{(1)};\Theta)\right\|^2 = \frac{1}{J}\sum_{j=1}^{J}\left\|\mathbf{x}_j^{(2)} - \mathcal{L}(\mathbf{x}_j^{(1)}) - \mathcal{N}(\mathbf{x}_j^{(1)};\Theta)\right\|^2. \tag{3.3}$$

It can be seen that the DNN in gResNet is used to model the difference between the output data $\mathbf{x}^{(2)}$ and the prediction by the operator $\mathcal{L}$. Upon satisfactory training of the optimized network parameter set Θ^*, we obtain the trained network $\mathcal{N}(\cdot,\Theta^*)$ and the corresponding gResNet prediction model: for a given initial state $\mathbf{x}_0$,

$$\mathbf{x}_{k+1} = \mathcal{L}(\mathbf{x}_k) + \mathcal{N}(\mathbf{x}_k;\Theta^*), \qquad k = 0, 1, \cdots. \tag{3.4}$$

It is straightforward to see, upon comparing (3.2) with the standard ResNet mapping (2.7), that the original ResNet model is a special case of the gResNet when $\mathcal{L} = \mathcal{I}$, the identity operator. In fact, when the time lag Δ is sufficiently small, $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathcal{O}(\Delta)$ and the exact flow map satisfies

$$\mathbf{\Phi} = \mathcal{I} + \mathcal{O}(\Delta).$$

This implies that the identity operator is a reasonable choice for $\mathcal{L}$ in term of (3.1). This is the reason why the original ResNet provides a computational advantage, as discussed in [25]. When Δ is not sufficiently small, the standard ResNet may not be advantageous.

3.2. Model correction for known $\mathcal{L}$

The operator $\mathcal{L}$ (3.1) in the gResNet model (3.2) is critical. It should be available prior to the gResNet model construction. Hereafter we will loosely refer to the operator $\mathcal{L}$ as "prior model". In general, $\mathcal{L}$ stands for any available models for the underlying dynamical system. This can be a linear approximation, a reduce order model, a coarse grained model, etc. By construction (3.2), the operator $\mathcal{N}$, which is defined by the DNN inside gResNet, serves as

$$\mathcal{N}(\cdot;\Theta^*) \approx \mathbf{\Phi} - \mathcal{L},$$

where $\mathbf{\Phi}$ is the flow map of the true model. In another word, the DNN in gResNet models the discrepancy between the true model $\mathbf{\Phi}$ and the prior model $\mathcal{L}$. If one assumes that the operator $\mathcal{L}$ is an approximation of the underlying dynamics via (3.1) such that

$$\mathbf{x}(t_{k+1}) = \mathcal{L}(\mathbf{x}(t_k)) + \mathcal{O}(\epsilon),$$

then it is natural to see the neural network operator $\mathcal{N} \sim \mathcal{O}(\epsilon)$. Consequently, this provides a computational advantage for the gResNet.

Since the prior model $\mathcal{L}$ represents a "coarse" approximation of the underlying dynamics, one can then view the network operator $\mathcal{N}$ in the gResNet as a "model correction" to $\mathcal{L}$, as shown in (3.2). Hereafter we will refer to the trained gResNet model (3.2) as "posteriori model". Examples of coarse, or reduced order, modeling are abundant in scientific computing. Here we give a very specific example, which is adopted from [21] and will be used in this paper as a numerical test.

Consider a true (and unknown) dynamical system

$$\begin{cases} \dfrac{dx_1}{dt} = -x_2 - x_3, \\ \dfrac{dx_2}{dt} = x_1 + \dfrac{1}{5}x_2, \\ \dfrac{dx_3}{dt} = \dfrac{1}{5} + y - 5x_3, \\ \dfrac{dy}{dt} = -\dfrac{y}{\epsilon} + \dfrac{x_1 x_3}{\epsilon}, \end{cases} \tag{3.5}$$

where $\epsilon > 0$ is a real parameter. This is a chaotic system. A reduced order model for this system is

$$\begin{cases} \dfrac{dX_1}{dt} = -X_2 - X_3, \\ \dfrac{dX_2}{dt} = X_1 + \dfrac{1}{5}X_2, \\ \dfrac{dX_3}{dt} = \dfrac{1}{5} + X_3(X_1 - 5), \end{cases} \tag{3.6}$$

where the fast variable y is averaged out. The reduced system (3.6) serves as a good approximation of the true system (3.5) when $\epsilon \ll 1$. This is our prior model in this case. The operator $\mathcal{L}$ of this prior model does not have an explicit expression and needs to be computed via solving (3.6) numerically.

3.3. Affine prior modeling for unknown $\mathcal{L}$

In many practical situations, one does not have an existing prior model and subsequently the operator $\mathcal{L}$ is not available. In this case, it is possible to construct a prior model and its associated operator $\mathcal{L}$ using the same dataset. It is also desirable that such an construction should be reasonably faster than the neural network training of the gResNet model, in order not to increase the overall computational cost. Any efficient method to create an approximation model using the data set (2.5) can be adopted. Here we present a construction using affine approximation as a possible choice. This affine approximation is a modification of dynamic model decomposition (DMD) method ([39]), which has been used as linear modeling for a variety of problems.

The idea of DMD is to construct a best-fit linear dynamical model to approximate the underlying system based on data. For the given data set (2.5), DMD seeks a linear flow map $\mathbf{A} \in \mathbb{R}^{n\times n}$ such that

$$\mathbf{x}_j^{(2)} \approx \mathbf{A}\mathbf{x}_j^{(1)}, \qquad \forall j = 1, \dots, J. \tag{3.7}$$

With sufficient number of data pairs, also known as snapshots, the matrix $\mathbf{A}$ can be solved in a least squares sense. For more detailed discussion of DMD, see [15].

The form of DMD (3.7) makes it effective for homogeneous systems. In order to cope with potential non-homogeneity of the underlying dynamics, we employ the following modification

$$\mathbf{x}_j^{(2)} \approx \mathbf{A}\mathbf{x}_j^{(1)} + \mathbf{b}, \qquad \forall j = 1, \dots, J, \tag{3.8}$$

where $\mathbf{b} \in \mathbb{R}^n$. Hereafter, we will refer to this as modified DMD (mDMD) method. This effectively creates an affine mapping as an approximation of the underlying flow map, i.e.,

$$\mathbf{\Phi}_\Delta(\mathbf{x}) \approx \mathbf{A}(\Delta)\mathbf{x} + \mathbf{b}(\Delta), \tag{3.9}$$

where the matrix $\mathbf{A}$ and vector $\mathbf{b}$ are solved via the following optimization problem

$$(\mathbf{A}, \mathbf{b}) = \underset{\substack{\hat{\mathbf{A}}\in\mathbb{R}^{n\times n} \\ \hat{\mathbf{b}}\in\mathbb{R}^n}}{\operatorname{argmin}} \frac{1}{J}\sum_{j=1}^{J} \left\| \mathbf{x}_j^{(2)} - \hat{\mathbf{A}}\mathbf{x}_j^{(1)} - \hat{\mathbf{b}} \right\|^2. \tag{3.10}$$

To solve the optimization problem, we take the data set (2.5) and write

$$\mathbf{X}_1 := \left[\mathbf{x}_1^{(1)}, \cdots, \mathbf{x}_J^{(1)} \right], \qquad \mathbf{X}_2 := \left[\mathbf{x}_1^{(2)}, \cdots, \mathbf{x}_J^{(2)} \right]. \tag{3.11}$$

Let $\mathbf{1} := [1 \cdots 1]$ be a row vector of size $1 \times J$ and

$$\widetilde{\mathbf{X}}_1 := \begin{bmatrix} \mathbf{X}_1 \\ \mathbf{1} \end{bmatrix}. \tag{3.12}$$

The solution to (3.10) is then readily available as

$$\begin{bmatrix} \mathbf{A}, & \mathbf{b} \end{bmatrix} = \mathbf{X}_2 \widetilde{\mathbf{X}}_1^{\dagger}, \tag{3.13}$$

where † stands for matrix pseudo inverse.

We now define the $\mathcal{L}$ operator in the gResNet (3.2) as the mDMD model, i.e.,

$$\mathcal{L}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}. \tag{3.14}$$

After training the network operator $\mathcal{N}$ using the loss function (3.3) to obtain the trained parameter set Θ^*, we obtain the mDMD based gResNet prediction model

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{b} + \mathcal{N}(\mathbf{x}_k; \Theta^*), \qquad k = 0, 1, \cdots. \tag{3.15}$$

Let

$$\epsilon_j^{\text{mDMD}} = \mathbf{x}_j^{(2)} - \mathbf{A}\mathbf{x}_j^{(1)} - \mathbf{b}, \qquad j = 1, \ldots, J,$$

be the residue of the mDMD model construction from the minimization problem (3.10). It is then straightforward to see that the loss function (3.3) can be viewed as

$$L(\Theta) = \frac{1}{J}\sum_{j=1}^{J} \left\| \epsilon_j^{\text{mDMD}} - \mathcal{N}(\mathbf{x}_j^{(1)}; \Theta) \right\|^2. \tag{3.16}$$

This further indicates that the neural network operator $\mathcal{N}$ is trained to learn the "residue" of the prior model, in this case the mDMD model.

3.4. Adaptive nonlinear prior modeling for unknown $\mathcal{L}$

The mDMD (resp. DMD) in the previous section employs affine (resp. linear) approximation as the prior model $\mathcal{L}$, which remains fixed for all solution data (2.5). Upon examining the affine approximation (3.14), it is evident that its form $\mathcal{L}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$ resembles that of a feedforward neural network with a single hidden layer, where $\mathbf{A}$ resembles the weight matrix and $\mathbf{b}$ resembles the bias vector. Motivated by this observation, we propose to use a single-layer neural network as the prior model $\mathcal{L}$. Let $\mathcal{N}_{\text{prior}}(\cdot; \Theta)$ be the operator associated with a feedforward neural network with a single hidden layer. We first train this network using the same data set (2.5) with the loss function

$$\Theta_{\text{prior}}^* = \underset{\Theta}{\text{argmin}} \frac{1}{J}\sum_{j=1}^{J} \left\| \mathbf{x}_j^{(2)} - \mathcal{N}_{\text{prior}}(\mathbf{x}_j^{(1)}; \Theta) \right\|^2. \tag{3.17}$$

This in turn gives us a trained prior model

$$\mathcal{L}(\mathbf{x}) = \mathcal{N}_{\text{prior}}(\mathbf{x}; \Theta_{\text{prior}}^*). \tag{3.18}$$

We then construct gResNet network by minimizing the loss function (3.3) with this newly trained prior model $\mathcal{L}$. It is straightforward to see that the loss function (3.3) can be written as

$$L(\Theta) = \frac{1}{J}\sum_{j=1}^{J} \left\| \epsilon_j^{\text{prior}} - \mathcal{N}(\mathbf{x}_j^{(1)}; \Theta) \right\|^2, \tag{3.19}$$

where

$$\epsilon_j^{\text{prior}} = \mathbf{x}_j^{(2)} - \mathcal{N}_{\text{prior}}(\mathbf{x}_j^{(1)}; \Theta_{\text{prior}}^*), \qquad j = 1, \ldots, J,$$

is the residue of the training error of the prior model $\mathcal{L}$.

Note that one is free to construct the prior model using a neural network with multiple hidden layers. However, with the gResNet inherently having a network $\mathcal{N}$ with multiple layers, there is no compelling reason to introduce multiple layers in the prior model, especially that the construction of the prior model should be reasonably fast as not to increase the overall cost of the model construction.

4. Numerical examples

In this section we present numerical examples to demonstrate the effectiveness of the proposed methods. For benchmarking purpose, in all examples the true dynamical models are known. We use the true models to generate synthetic data and then construct the corresponding gResNet approximation models. The gResNet models are then used for system predictions and compared against the solutions of the true systems. Example 1 and 2 are linear systems, one homogeneous and one inhomogeneous. These are to verify the need of using mDMD from Section 3.3, as the standard DMD fails for inhomogeneous systems. Example 3 and 4 are nonlinear counterparts of Example 1 and 2. Example 5 is an industry problem of differential-algebraic system, whose solution exhibits sharp spikes and change of scale. Example 6 is a chaotic system with complex oscillatory behavior over long-term. Note that even though the examples have rather simple analytical form in their governing equations, the solution behaviors can be highly complex and challenging to capture, particularly for Example 5 and 6.

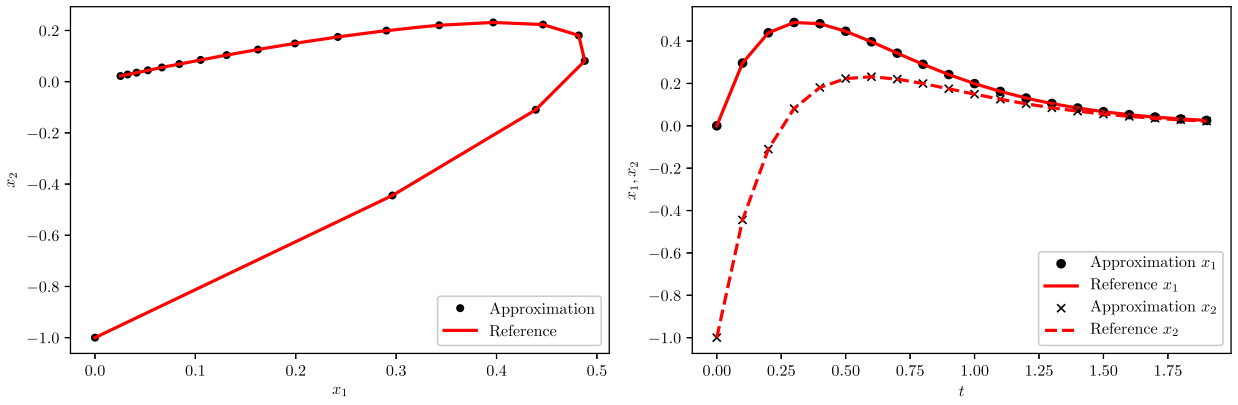

Fig. 4.1. Example 1. Left: phase plot; Right: trajectory.

4.1. Data and network training

To generate data sets in the form of (2.5), we conduct random sampling for the "initial condition" $\mathbf{x}_j^{(1)}$ and use the true models to advance the time lag Δ to obtain the corresponding $\mathbf{x}_j^{(2)}$. It has been established in [48] that random sampling is more effective for equation recovery work. Depending on the network structure, the total number of data pairs in (2.5), J, is usually kept at $5 \sim 10$ times of the number of parameters in the network structure. This is to ensure that the network training does not suffer from overfitting issues. Note there is no comprehensive theory regarding the sufficient number of data entries to ensure accurate network training. Therefore, we purposefully keep the data set sufficiently large so that we can focus on the fundamental properties of the network. All models are trained via the loss function (3.3) and by using the open-source Tensorflow library [1]. The training data sets are usually divided into mini-batches of size 10. All models are trained for ~ 300 epochs with reshuffling after each epoch. All the weights are initialized randomly from Gaussian distributions and all the biases are initialized to be zeros. We use $\sigma(x) = \tanh(x)$ as activation function in all the examples.

It should be noted that our training data set construction represents one of the unique features of the learning method. The training data are pairs of state variables, separated by a short time interval Δ. In another word, each entry in the data set is a solution trajectory of very short length Δ. In practice, this poses very little burden for data collection: upon selecting (randomly) a large number of initial conditions, one only needs to evolve the system for a very short time Δ and collect the ending state. Also, since each of the trajectories in the data set is very short, it contains no "physics" or other notable temporal behaviors – the state variables barely evolve during the short time Δ. However, the DNN model is able to learn the mechanism behind the short trajectories, i.e., the flow map. During prediction, for any arbitrarily given initial condition, the trained DNN model executes the learned dynamics and produces long-term system trajectories that may contain complex temporal behaviors, which are not present in the training data.

4.2. Linear systems

We first study two linear ODE systems, whose exact solutions are known. In both examples, our gResNet networks have 3 hidden layers, each of which with 30 neurons.

4.2.1. Example 1

We consider the following simple linear ODE system:

$$\begin{cases} \dot{x}_1 = x_1 - 4x_2, \\ \dot{x}_2 = 4x_1 - 7x_2. \end{cases} \tag{4.1}$$

The computational domain is taken to be $D = [0, 2]^2$ and the time lag $\Delta = 0.1$. No pre-existing prior model is involved. Instead, we adopt the approach discussed in Section 3.3 and construct a standard DMD as the prior model in the gResNet model. Note that the true dynamical system is linear and homogeneous. Subsequently, the standard DMD can be exact.

After satisfactory training, we conduct system prediction for up to $t = 2$ for some arbitrarily chosen initial conditions. The phase plot and trajectories of the DMD based gResNet (denoted as "DMD-ResNet") are shown in Fig. 4.1. We observe that the numerical predictions match the reference solutions extremely well. This is mostly due to the high accuracy of the DMD prior model. The neural network, which serves as a correction to the DMD prior model, has almost negligible impact in this case. This is manifested from Fig. 4.2 (left), where we observe the training loss for DMD-ResNet reaches extremely small magnitude. In the right of Fig. 4.2, we plot the numerical errors in the trajectory prediction. We observe that DMD based gResNet incurs much smaller errors than the standard ResNet.

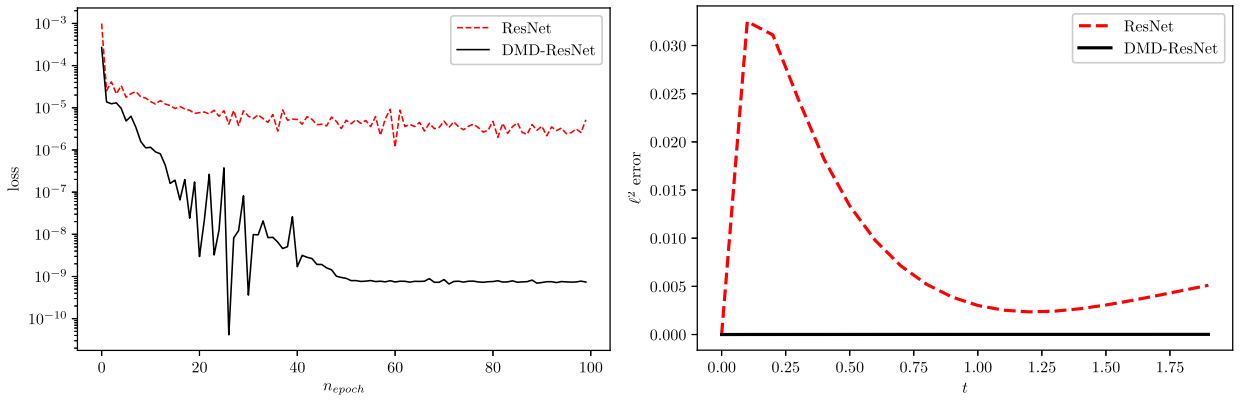

Fig. 4.2. Example 1. Left: loss history during training; Right: errors in trajectory prediction.

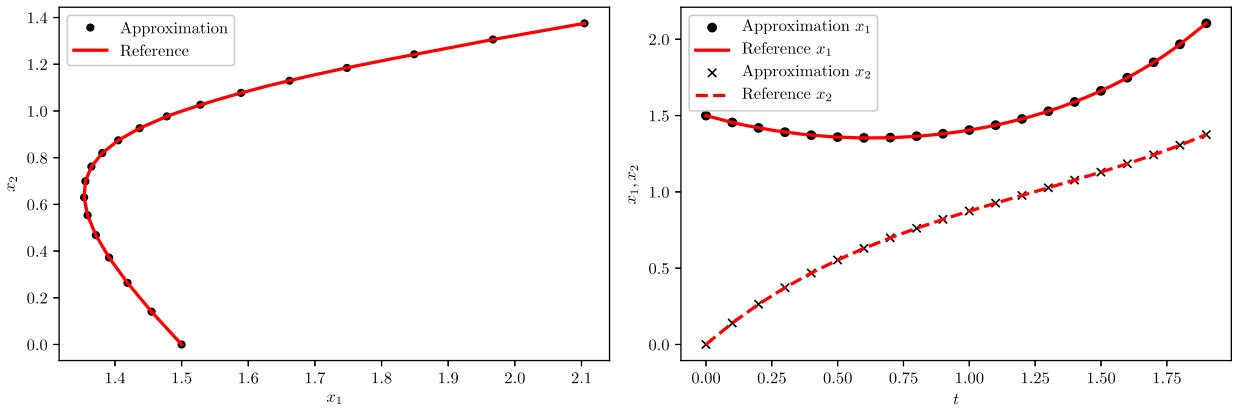

Fig. 4.3. Example 2. Results by mDMD-ResNet. Left: phase plot; Right: trajectory.

4.2.2. Example 2

We now consider a non-homogeneous ODE system:

$$\begin{cases} \dot{x}_1 = x_1 + x_2 - 2, \\ \dot{x}_2 = x_1 - x_2. \end{cases} \tag{4.2}$$

The computational domain is taken to be $D = [0, 2]^2$ and the time lag $\Delta = 0.1$. We employ the standard ResNet, gResNet using the standard DMD as prior model (DMD-ResNet) and gResNet using the modified DMD as prior model (mDMD-ResNet). System predictions by the learned models are conducted for time up to $t = 2$. In Fig. 4.3, we show the trajectory plots and phase portrait produced by the mDMD-ResNet model. We observe very good agreement with the exact solution. The mDMD-ResNet is in fact the most accurate model of the three approaches. This can be seen from Fig. 4.4. It can be seen that the numerical errors in the prediction by mDMD-ResNet are two orders of magnitude smaller than those by ResNet and DMD-ResNet. This demonstrates that the gResNet method can be advantageous when a proper prior model is available or can be constructed (in this case via mDMD). The standard DMD is not a very accurate prior model, as it can not model the non-homogeneous term in the system. In this case, its performance is similar to the standard ResNet, which corresponds to using the identity operator as the prior model.

We then consider the case of noisy data by adding randomly generated small noises to the synthetic data. The results by mDMD-ResNet are shown in Fig. 4.5, with noise at 2% and 5% relative levels. Again, mDMD-ResNet produces accurate system predictions, whose discrepancy with the exact solution is higher at the noise level 5% than at the noise level 2%. This is expected.

4.3. Nonlinear systems

We now consider four nonlinear examples: (1) a nonlinear non-homogeneous system; (2) the well-known damped pendulum problem; (3) a nonlinear differential-algebraic system for electric network model; and (4) the chaotic multiscale

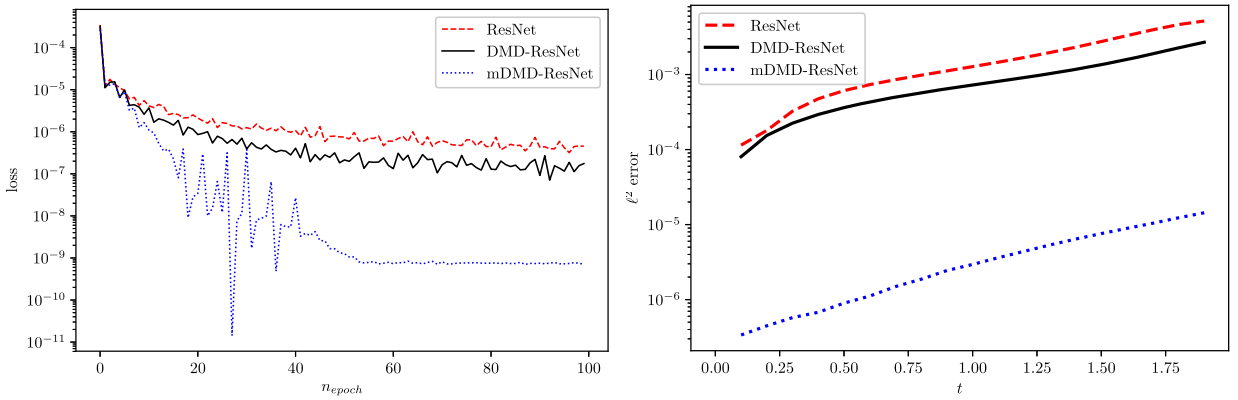

Fig. 4.4. Example 2. Left: loss history during training; Right: errors in trajectory prediction.

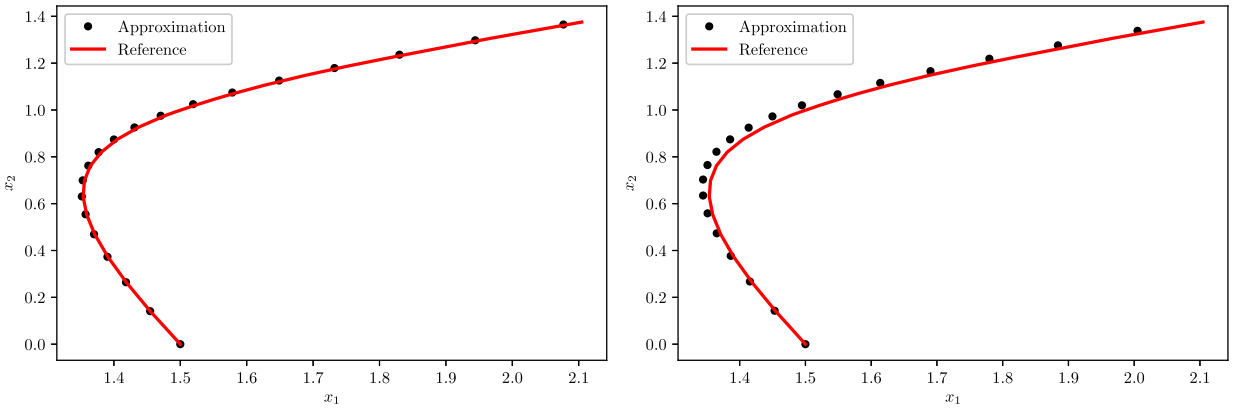

Fig. 4.5. Example 2 with noisy data. Phase plots of mDMD-ResNet with $x_0 = (1.5, 0)$. Left: 2% noises; Right: 5% noises.

system (3.5) from Section 3.2. In the first and fourth example, the neural networks have 3 hidden layers, each of which with 30 neurons. In the second and third examples, the networks have 2 hidden layers, each of which with 40 neurons.

4.3.1. Example 3

We consider the following system

$$\begin{cases} \dot{x}_1 = x_1 + x_2 - 2, \\ \dot{x}_2 = x_1 - x_2 + 0.5 \sin x_2. \end{cases} \tag{4.3}$$

This is a modification of Example 2 by adding a nonlinear term. The computational domain is taken to be $D = [0, 3]^2$ and the time lag $\Delta = 0.1$. Upon learning the system, predictions are conducted up to $t = 2$. The phase plot and trajectories produced by the mDMD-ResNet are plotted in Fig. 4.6. The training error history and numerical errors in the trajectory predictions are plotted In Fig. 4.7, along with those produced by ResNet and DMD-ResNet. Again, we observe that mDMD-ResNet produces far superior predictions than those by ResNet and DMD-ResNet. This is further demonstrated in Table 4.1. Note that the network norm from mDMD-ResNet is much smaller than ResNet and DMD-ResNet. This is the primary reason for the better performance by mDMD-ResNet.

4.3.2. Example 4: damped pendulum

We now consider the damped pendulum problem

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

where $\alpha = 0.2$ and $\beta = 8.91$. The computational domain is $D = [-\pi, \pi] \times [-2\pi, 2\pi]$, and the time lag $\Delta = 0.1$. Here we employ the adaptive nonlinear approximation method from Section 3.4 as the prior model. In Fig. 4.8, we present the trajectory prediction results of the adaptive gResNet model, starting from an arbitrarily chosen initial condition $\mathbf{x}_0 =$

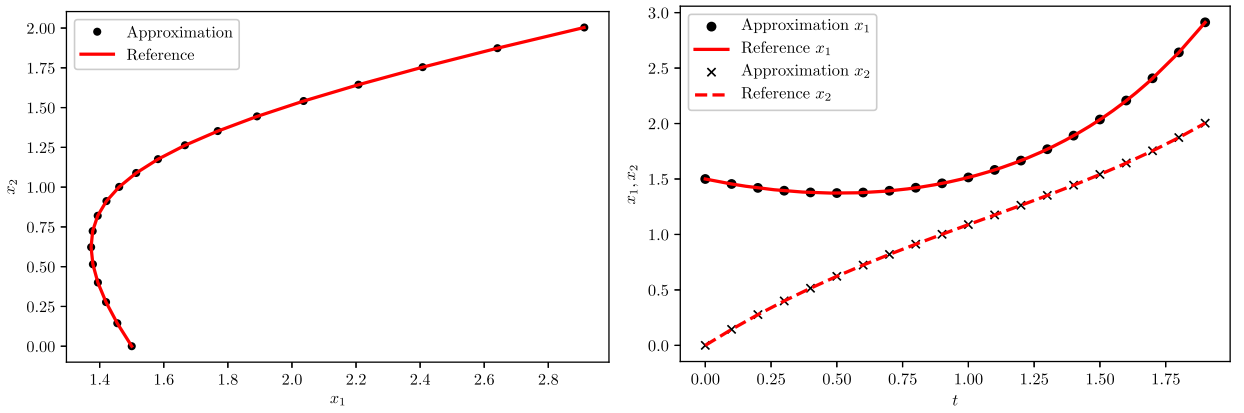

Fig. 4.6. Example 3 with mDMD-ResNet. Left: phase plot with $\mathbf{x}_0 = (1.5, 0)$; Right: trajectory prediction.

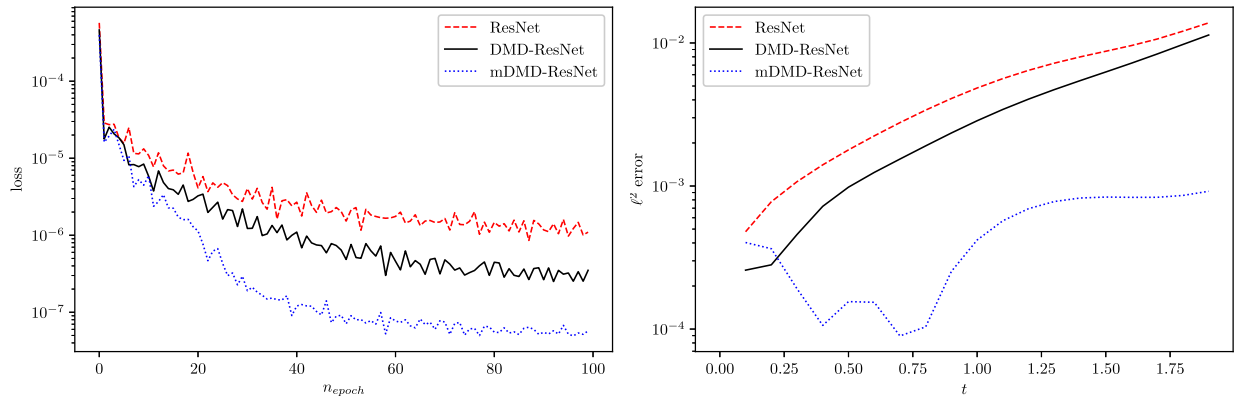

Fig. 4.7. Example 3. Left: loss history during training; Right: errors in trajectory prediction.

Table 4.1
Example 3. Key network properties for ResNet, DMD-ResNet and mDMD-ResNet.

	Prediction error	Training loss	Validation loss	Network norm
ResNet	5.5296e − 03	7.9243e − 08	7.8290e − 08	2.3129e − 02
DMD-ResNet	3.8551e − 03	1.8546e − 08	1.8381e − 08	8.9920e − 03
mDMD-ResNet	4.9431e − 04	6.6394e − 09	6.6046e − 09	1.4054e − 03

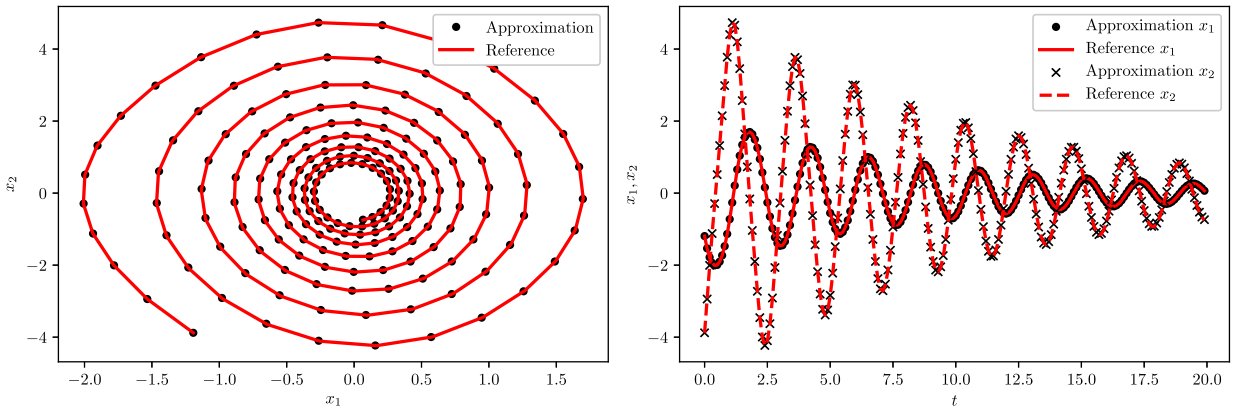

Fig. 4.8. Example 4 with adaptive gResNet modeling. Left: phase portrait; Right: trajectory prediction.

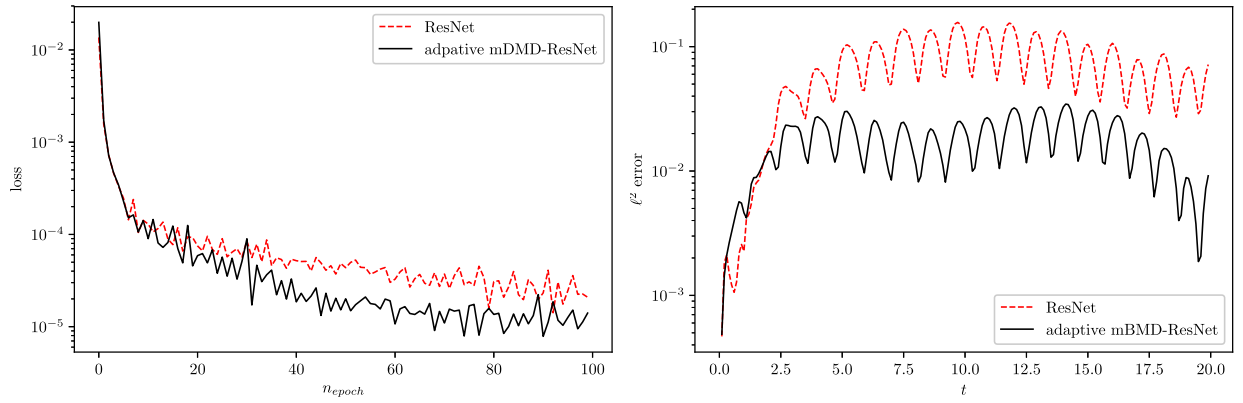

Fig. 4.9. Example 4. Left: loss history during training; Right: errors in trajectory prediction.

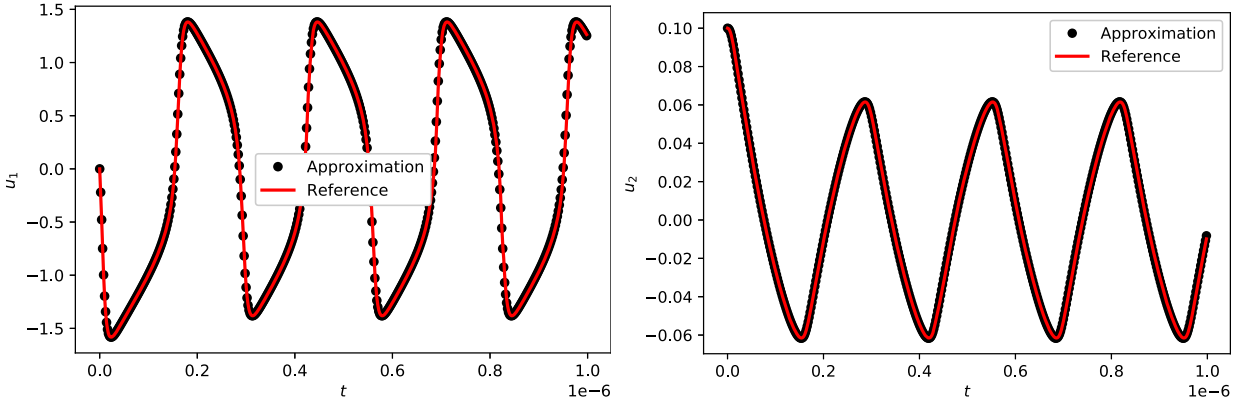

Fig. 4.10. Example 5. Solution prediction of (u_1, u_2) via mDMD-ResNet.

$(-1.193, -3.876)$ and for time up to $t = 20$. We observe excellent agreements between the network model predictions and the reference solutions. In Fig. 4.9, we show the training loss history and trajectory errors in the prediction, along with those obtained by ResNet. It can be seen that, with its prediction errors one order of magnitude smaller, the adaptive gResNet method produces much more accurate results than the standard ResNet method.

Example 5: nonlinear electric network

We now consider a system of nonlinear differential-algebraic equations (DAE), a model for a electric network from [23],

$$\begin{cases} \dot{u}_1 = v_2/C, \\ \dot{u}_2 = u_1/L, \\ 0 = v_1 - (G_0 - G_\infty)U_0 \tanh(u_1/U_0) - G_\infty u_1, \\ 0 = v_2 + u_2 + v_1, \end{cases}$$

where u_1 is the node voltage, and u_2, v_1 and v_2 are branch currents. The physical parameters are specified as $C = 10^{-9}$, $L = 10^{-6}$, $U_0 = 1$, $G_0 = -0.1$ and $G_\infty = 0.25$. In our test, we define the computational domain of (u_1, u_2) as $D = [-2, 2] \times [-0.2, 0.2]$ and fix the time lag $\Delta t = 2 \times 10^{-9}$. For system prediction, we choose an (arbitrary) initial condition $\mathbf{u}_0 = (0, 0.1)$ and produce prediction result for up to $t = 1 \times 10^{-6}$ (1,000 times of the size of Δ).

The solution trajectories predicted by mDMD-ResNet are shown in Fig. 4.10 and Fig. 4.11. We observe high accuracy in the prediction, when compared with the reference solution. The training loss history and numerical errors in the system prediction are plotted in Fig. 4.12, along with those produced by the standard ResNet. It can be seen that the performance of ResNet and DMD-ResNet is similar. This is because in this particular example the time lag $\Delta = 10^{-9}$ is very small, which is dictated by the scaling of the physical problem. Consequently, the identity operator $\mathcal{I}$ can be considered a very good prior model and the standard ResNet performs well. A more detailed comparison of the ResNet and mDMD-ResNet models is presented in Table 4.2. It can be seen that the mDMD-ResNet still offers slight advantage.

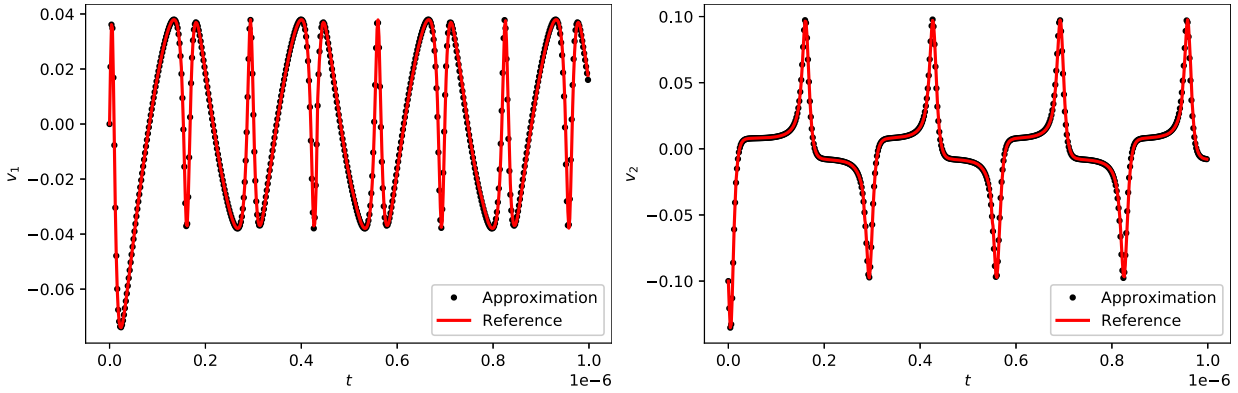

Fig. 4.11. Example 5. Solution prediction of (v_1, v_2) via mDMD-ResNet.

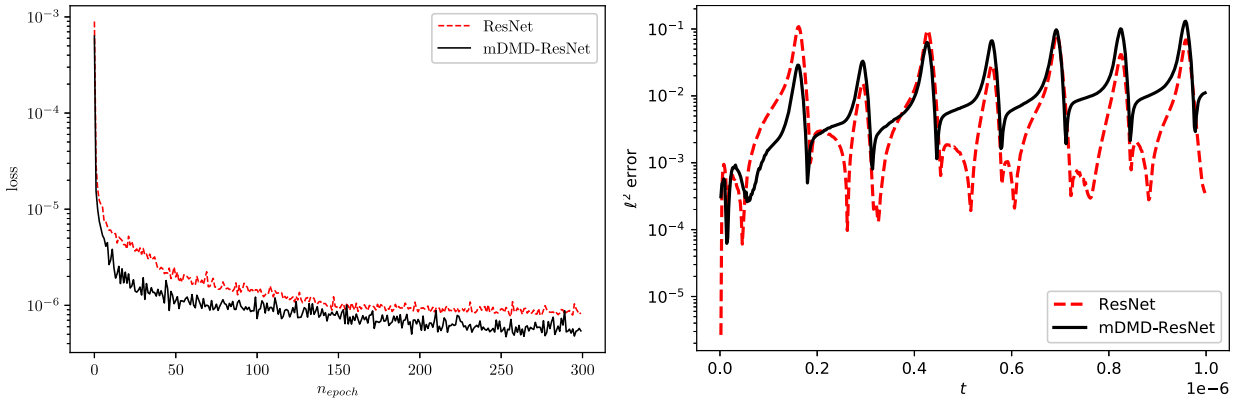

Fig. 4.12. Example 5. Left: loss history during training; Right: errors in trajectory predictions.

Table 4.2
Example 5: Key network properties for ResNet and mDMD-ResNet.

	Prediction error	Training loss	Validation loss	Network norm
ResNet	1.0724e − 02	9.67135e − 08	1.0063e − 07	2.4569e − 01
mDMD-ResNet	1.4715e − 02	2.9232e − 08	3.0879e − 08	8.3937e − 02

Example 6

We now consider the chaotic multiscale system (3.5) from Section 3.2. The prior model is the averaged system (3.6). This represents a case discussed in Section 3.2, where the prior model is available as an existing coarse model. The operator $\mathcal{L}$ associated with the prior model does not have an explicit form and needs to be computed by numerically solving the reduced system (3.6).

The prior model (3.6) is a good approximation of the true model (3.5) when the parameter $\epsilon \ll 1$. Here we set $\epsilon = 0.1$, which is not exceedingly small. In this case, the approximation offered by the prior model (3.6) is relatively coarse. We fix the computational domain as $D = [-15, 15] \times [-15, 10] \times [-5, 25] \times [-30, 140]$ and set the time lag as $\Delta = 0.05$. After satisfactory training, we utilize the trained gResNet model to conduct long-term prediction for time up to $t = 100$. The results from an arbitrarily chosen initial condition are shown in Fig. 4.13. For comparison, we also plot the prediction results obtained by the reduced system (3.6) (labeled as "Reduced"), the standard ResNet model, along with the reference exact solution via solving the true system (3.5) numerically. We first observe that the gResNet method offers significantly better results than the standard ResNet. This again confirms that it is highly advantageous to have a good prior model. In this case, the reduced system (3.6) in gResNet is obviously much better than the crude model of identity operator in the standard ResNet. More careful examination of the results also reveals that the gResNet has better predictive accuracy than the reduced model, especially in term of capturing the correct phase over longer time. To visual this closely, we compute the spectrum density of the trajectories in Fig. 4.14, in order to examine the dominant frequencies in the solutions. It can be clearly seen that the gResNet offers significant improvement in accuracy over the reduced system.

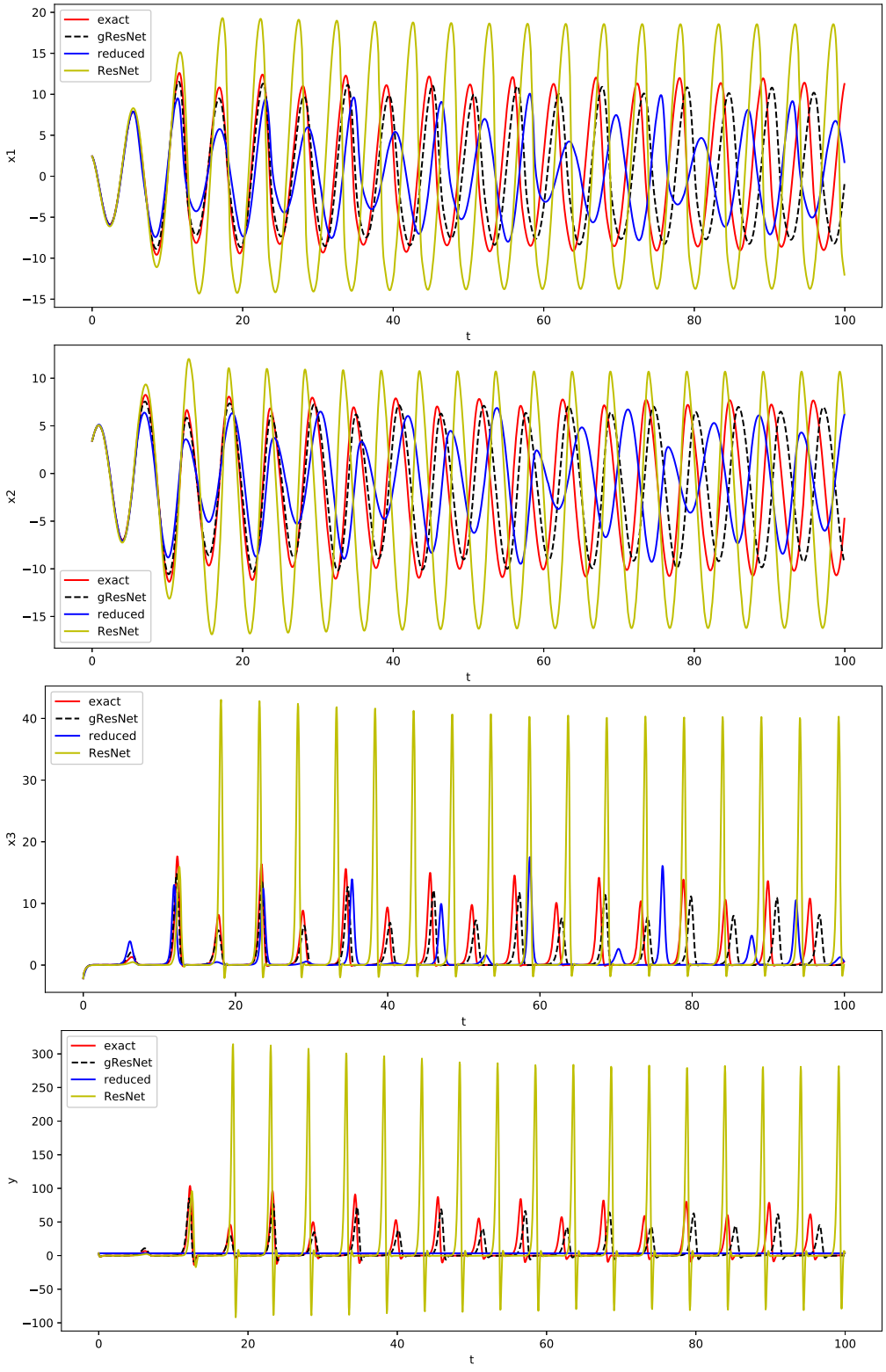

Fig. 4.13. Example 6. Trajectory predictions by different models using initial condition (2.4350451, 3.416925, -2.16129375, 3.4650658). Note that the Reduced system does not contain variable y. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

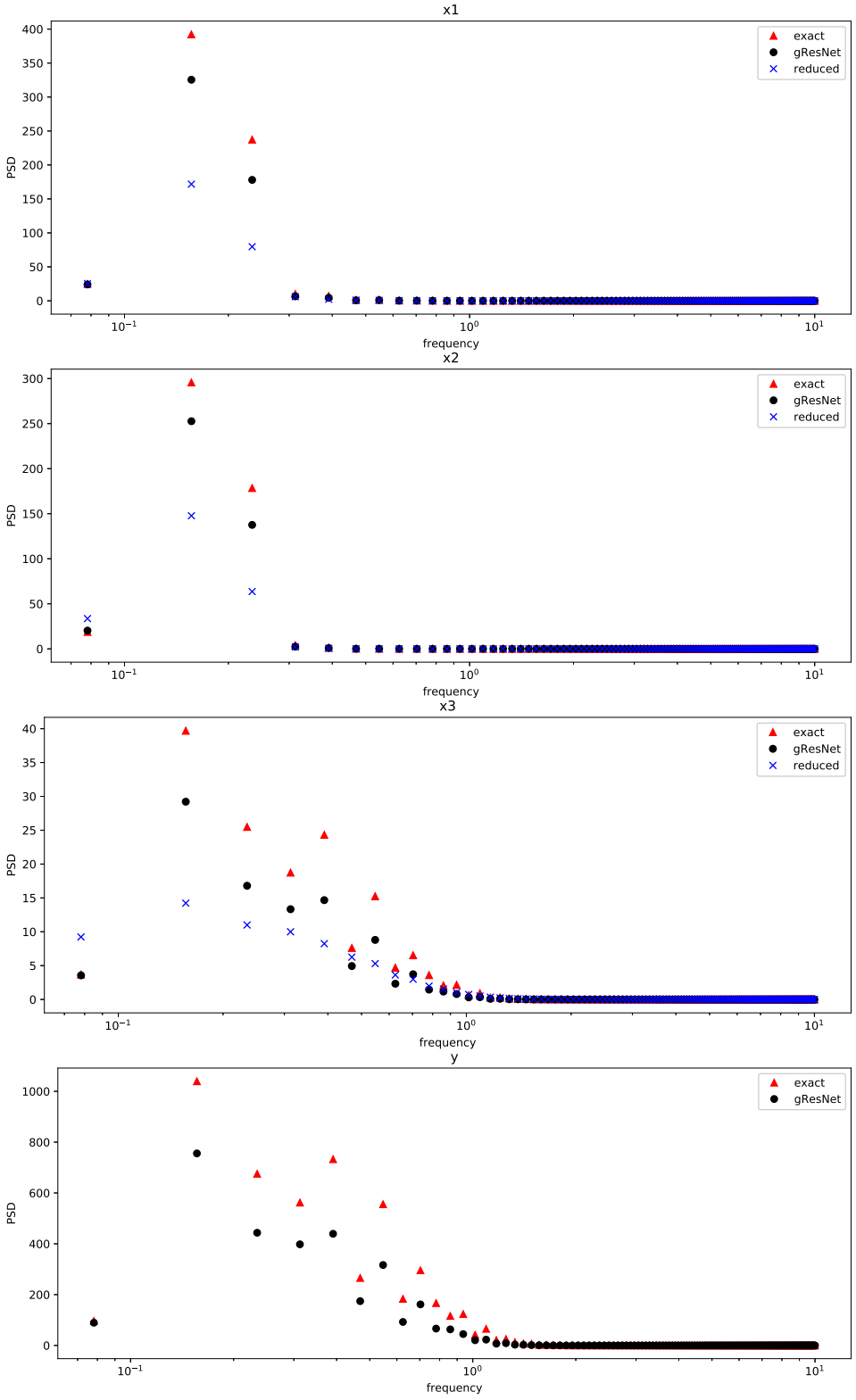

Fig. 4.14. Example 6. Power spectral density (PSD) of each trajectory obtained by different models.

5. Conclusion

We presented a generalized residual network (gResNet) framework for effective learning of unknown dynamical systems from observational data. The gResNet incorporates the standard residual network (ResNet) as a special case. In gResNet, "residue" is more broadly defined as the difference between the data and the prediction of a prior model, and DNN is used to model the residue. Consequently, gResNet can be considered as a model correction to the prior model, which is usually a reduced/coarse model. In situations where prior models are not available, we propose a few choices for fast construction of prior models using the same data set and without incurring much computational cost. Various numerical examples demonstrated that gResNet is a viable tool for data driven learning of unknown dynamics and offers better accuracy than the standard ResNet. It is especially useful as a model correction tool, to improve the predictive accuracy of existing coarse or reduced order models.

CRediT authorship contribution statement

The authors of the manuscript have equal contributions.

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] M. Bayarri, J. Berger, R. Paulo, J. Sacks, J. Cafeo, C.L.J. Cavendish, J. Tu, A framework for validation of computer models, Technometrics 49 (2007) 138–154.
[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.H. Elsheikh, A machine learning approach for efficient uncertainty quantification using multiscale methods, J. Comput. Phys. 354 (2018) 493–511, https://doi.org/10.1016/j.jcp.2017.10.034.
[6] J. Han, A. Jentzen, W. E, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci. 115 (2018) 8505–8510, https://doi.org/10.1073/pnas.1718942115, https://www.pnas.org/content/115/34/8505, https://www.pnas.org/content/115/34/8505.full.pdf.
[7] Y. He, D. Xiu, Numerical strategy for model correction using physical constraints, J. Comput. Phys. 313 (2016) 617–634.
[8] J. Hesthaven, S. Ubbiali, Non-intrusive reduced order modeling of nonlinear problems using neural networks, J. Comput. Phys. 363 (2018) 55–78, https://doi.org/10.1016/j.jcp.2018.02.037, http://www.sciencedirect.com/science/article/pii/S0021999118301190.
[9] D. Higdon, M. Kennedy, J. Cavendish, J. Cafeo, R. Ryne, Combining field data and computer simulations for calibration and prediction, SIAM J. Sci. Comput. 26 (2004) 448–466.
[10] V. Joseph, S. Melkote, Statistical adjustments to engineering models, J. Qual. Technol. 41 (2009) 362–375.
[11] S.H. Kang, W. Liao, Y. Liu, IDENT: identifying differential equations with numerical time evolution, arXiv preprint, arXiv:1904.03538, 2019.
[12] 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.
[13] M. Kennedy, A. O'Hagan, Bayesian calibration of computer models, J. R. Stat. Soc. 63 (2001) 425–464.
[14] Y. Khoo, J. Lu, L. Ying, Solving for high-dimensional committor functions using artificial neural networks, Res. Math. Sci. 6 (2018) 1, https://doi.org/10.1007/s40687-018-0160-2.
[15] J.N. Kutz, S.L. Brunton, B.W. Brunton, J.L. Proctor, Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems, SIAM, 2016.
[16] J.N. Kutz, S.L. Brunton, D.M. Luchtenburg, C.W. Rowley, J.H. Tu, On dynamic mode decomposition: theory and applications, J. Comput. Dyn. 1 (2014) 391–421, https://doi.org/10.3934/jcd.2014.1.391.
[17] 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.
[18] 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 2018, in: Proceedings of Machine Learning Research, PMLR, vol. 80, pp. 3208–3216.
[19] 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).
[20] D. Nguyen, S. Ouala, L. Drumetz, R. Fablet, EM-like learning chaotic dynamics from noisy and partial observations, arXiv preprint, arXiv:1903.10335, 2019.
[21] G. Pavliotis, A. Stuart, Multiscale Methods: Averaging and Homogenization, Springer, 2008.
[22] 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, https://doi.org/10.1063/1.5113494.
[23] R. Pulch, Polynomial chaos for semiexplicit differential algebraic equations of index 1, Int. J. Uncertain. Quantificat. 3 (2013).
[24] Z. Qian, C. Wu, Bayesian hierarchical modeling for integration low-accuracy and high-accuracy experiments, Technometrics 50 (2008) 192–204.
[25] T. Qin, K. Wu, D. Xiu, Data driven governing equations approximation using deep neural networks, J. Comput. Phys. 395 (2019) 620–635.
[26] M. Raissi, Deep hidden physics models: deep learning of nonlinear partial differential equations, J. Mach. Learn. Res. 19 (2018) 1–24.
[27] M. Raissi, G.E. Karniadakis, Hidden physics models: machine learning of nonlinear partial differential equations, J. Comput. Phys. 357 (2018) 125–141.
[28] M. Raissi, P. Perdikaris, G.E. Karniadakis, Machine learning of linear differential equations using Gaussian processes, J. Comput. Phys. 348 (2017) 683–693.
[29] 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.

[30] 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.
[31] M. Raissi, P. Perdikaris, G.E. Karniadakis, Multistep neural networks for data-driven discovery of nonlinear dynamical systems, arXiv preprint, arXiv:1801.01236, 2018.
[32] D. Ray, J.S. Hesthaven, An artificial neural network as a troubled-cell indicator, J. Comput. Phys. 367 (2018) 166–191, https://doi.org/10.1016/j.jcp.2018.04.029, http://www.sciencedirect.com/science/article/pii/S0021999118302547.
[33] S.H. Rudy, S.L. Brunton, J.L. Proctor, J.N. Kutz, Data-driven discovery of partial differential equations, Sci. Adv. 3 (2017) e1602614.
[34] 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.
[35] K. Sargsyan, H. Najm, R. Ghanem, On the statistical calibration of physical models, Int. J. Chem. Kinet. (2015), https://doi.org/10.1002/kin.20906.
[36] H. Schaeffer, Learning partial differential equations via data discovery and sparse optimization, Proc. R. Soc. Lond., Ser. A, Math. Phys. Eng. Sci. 473 (2017).
[37] H. Schaeffer, S.G. McCalla, Sparse model selection via integral terms, Phys. Rev. E 96 (2017) 023302.
[38] H. Schaeffer, G. Tran, R. Ward, Extracting sparse high-dimensional dynamics from limited data, SIAM J. Appl. Math. 78 (2018) 3279–3295.
[39] P. Schmid, Dynamic mode decomposition of numerical and experimental data, J. Fluid Mech. 656 (2010) 5–28.
[40] 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.
[41] R. Tibshirani, Regression shrinkage and selection via the lasso, J. R. Stat. Soc., Ser. B, Methodol. (1996) 267–288.
[42] G. Tran, R. Ward, Exact recovery of chaotic systems from highly corrupted data, Multiscale Model. Simul. 15 (2017) 1108–1129.
[43] R.K. Tripathy, I. Bilionis, Deep UQ: learning deep neural network surrogate models for high dimensional uncertainty quantification, J. Comput. Phys. 375 (2018) 565–588, https://doi.org/10.1016/j.jcp.2018.08.036.
[44] Q. Wang, J.S. Hesthaven, D. Ray, Non-intrusive reduced order modeling of unsteady flows using artificial neural networks with application to a combustion problem, J. Comput. Phys. 384 (2019) 289–307, https://doi.org/10.1016/j.jcp.2019.01.031, http://www.sciencedirect.com/science/article/pii/S0021999119300828.
[45] S. Wang, W. Chen, K. Tsui, Bayesian validation of computer models, Technometrics 51 (2009) 439–451.
[46] Y. Wang, G. Lin, Efficient deep learning techniques for multiphase flow simulation in heterogeneous porous media, https://arxiv.org/abs/1907.09571, 2019.
[47] K. Wu, T. Qin, D. Xiu, Structure-preserving method for reconstructing unknown Hamiltonian systems from trajectory data, arXiv preprint, arXiv:1905.10396, 2019.
[48] K. Wu, D. Xiu, Numerical aspects for approximating governing equations using data, J. Comput. Phys. 384 (2019) 200–221.
[49] Y. Zhu, N. Zabaras, Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification, J. Comput. Phys. 366 (2018) 415–447, https://doi.org/10.1016/j.jcp.2018.04.018.