# Recurrent neural networks for dynamical systems: Applications to ordinary differential equations, collective motion, and hydrological modeling

Cite as: Chaos **33**, 013109 (2023); https://doi.org/10.1063/5.0088748
Submitted: 20 February 2022 • Accepted: 07 December 2022 • Published Online: 05 January 2023

K. Gajamannage, D. I. Jayathilake, Y. Park, et al.

# Recurrent neural networks for dynamical systems: Applications to ordinary differential equations, collective motion, and hydrological modeling

Cite as: Chaos **33**, 013109 (2023); doi: 10.1063/5.0088748
Submitted: 20 February 2022 · Accepted: 7 December 2022 ·
Published Online: 5 January 2023

K. Gajamannage,[1,a)] D. I. Jayathilake,[2] Y. Park,[1] and E. M. Bollt[3]

## AFFILIATIONS

[1]Department of Mathematics and Statistics, Texas A&M University—Corpus Christi, Corpus Christi, Texas 78412, USA
[2]Department of Physical and Environmental Sciences, Texas A&M University—Corpus Christi, Corpus Christi, Texas 78412, USA
[3]Department of Electrical and Computer Engineering and The Clarkson Center for Complex Systems, Clarkson University, Potsdam, New York 13699, USA

[a)]**Author to whom correspondence should be addressed:** kelum.gajamannage@tamucc.edu

## ABSTRACT

Classical methods of solving spatiotemporal dynamical systems include statistical approaches such as autoregressive integrated moving average, which assume linear and stationary relationships between systems' previous outputs. Development and implementation of linear methods are relatively simple, but they often do not capture non-linear relationships in the data. Thus, artificial neural networks (ANNs) are receiving attention from researchers in analyzing and forecasting dynamical systems. Recurrent neural networks (RNNs), derived from feed-forward ANNs, use internal memory to process variable-length sequences of inputs. This allows RNNs to be applicable for finding solutions for a vast variety of problems in spatiotemporal dynamical systems. Thus, in this paper, we utilize RNNs to treat some specific issues associated with dynamical systems. Specifically, we analyze the performance of RNNs applied to three tasks: reconstruction of correct Lorenz solutions for a system with a formulation error, reconstruction of corrupted collective motion trajectories, and forecasting of streamflow time series possessing spikes, representing three fields, namely, ordinary differential equations, collective motion, and hydrological modeling, respectively. We train and test RNNs uniquely in each task to demonstrate the broad applicability of RNNs in the reconstruction and forecasting the dynamics of dynamical systems.

**Learning non-linear systems in nature requires researchers to solve complicated mathematical equations that often involve both high computational complexity and less computational precision.[1] Artificial neural networks (ANNs) are a class of algorithms that model underlying relationships of a set of data through a process that mimics the way the human brain operates. Data-driven approaches, such as ANNs, are capable of affording such nonlinear systems with both high efficiency and good accuracy. ANN models are increasingly popular for two reasons. First, they guarantee that arbitrary continuous dynamical systems can be approximated by ANNs with a sufficient amount of sub-models, which are called hidden layers.[2] Second, ANNs themselves can be viewed as discretizations of continuous dynamical systems, which make them suitable for studying dynamics.[3] A recurrent neural network (RNN) is a class of ANNs where the connections between nodes form a directed graph along a temporal sequence where the output of a hidden node at the current time step is sent to the corresponding hidden node for the next time step. Among all the ANN architectures, RNNs are considered to be the most faithful candidates for modeling and forecasting temporal data sequences such as time series data. Thus, here, we implement RNN to reconstruct and forecast data generated by several nonlinear dynamical systems.**

## I. INTRODUCTION

Dynamical systems may exhibit dynamics that are highly sensitive to initial conditions, which is referred to as the butterfly effect. As a result of this sensitivity, which often manifests itself as an exponential growth of perturbations in the initial conditions, the behavior of dynamical systems is sophisticated.[4] From a historical point of view, reconstruction or forecast of dynamical systems, especially those with spatiotemporal solutions, has been based on the techniques like linear parametric autoregressive models, moving-average models, autoregressive moving-average models, etc. All these models are linear and some of them possess the assumption that the data is stationary; thus, these models are not able to cope with complex relationships in the data.[5]

In recent decades, deep learning based frameworks have shown superior performance in the reconstruction or forecast of dynamical systems.[54,55] Among them, RNNs,[6] long short-term memories (LSTMs),[7] neural ordinary differential equations (neural-ODEs),[8] and temporal convolutional neural networks (temporal-CNNs)[9] are prominent. RNNs have been successfully applied in diverse research fields for more than four decades in learning patterns in sequential data, especially spatiotemporal, that are observed in dynamical systems. Each RNN unit has one gate to regulate the information flow to perform recurrent connections between units. RNNs are simple and less computationally expensive, with respect to both memory and space, deep learning tools compared to the aforesaid deep learning tools that were developed during the recent decades. LSTMs are an extension of RNNs where each LSTM unit has four gates to efficiently regulate the information flow to offer short-term memory into the ANN.[7] LSTMs are widely used in learning spatiotemporal dynamics of physical systems; however, they are computationally expensive and easy to overfit the training data due to their four-gate architecture.[10] The general structure of a neural-ODE consists of an RNN encoder, an ODE solver, and a decoder.[11] Neural-ODEs share some structural similarities to RNNs but they are more computationally expensive than RNNs, as an RNN is just one module out of the three modules in a neural-ODE. The main functional difference between the neural-ODEs and RNNs is that neural-ODEs are capable of processing uneven time sampling of the data whereas RNNs are not capable of processing such data.[11]

A temporal-CNN is a two-step convolutional process such that, first, it computes low-level features using a CNN that encodes spatial–temporal information, and, second, it passes these low-level features into a classifier that captures high-level temporal information using an RNN.[9] Temporal-CNNs offer better control of memory as they are more flexible in changing their receptive field size by either stacking more convolutional layers, using larger dilation factors, or increasing filter size.[12] Moreover, a temporal-CNN has a backpropagation path different from the temporal direction of the sequence which avoids exploding or vanishing gradient problem.[12] Since a temporal-CNN consists of both RNN and CNN modules, it is highly computationally expensive than an RNN itself. In general, temporal-CNNs suffer the following deficiencies: (1) they require more memory during the evaluation as they take in the raw data sequence up to the effective history length, and (2) it is difficult to change their parameters for a transfer of the domain. Among all the advantages and drawbacks pertaining to each of the discussed deep learning methods above, we emphasize the computational complexity (both memory and space) and simplicity; thus, we utilize RNNs to perform reconstructions/forecasts on dynamical systems in this paper.

We utilize RNNs with unique routings of training and testing to model a general class of dynamical systems since they are considered to be state-of-the-art machine learning methods for sequential data. Applications of RNNs include image classification, object recognition, object detection, speech recognition, language translation, voice synthesis, etc.[13] An ANN usually involves a large number of parallelly operating processing units that are arranged in layers. The first layer receives the raw input information analogous to optic nerves in human visual processing. Each successive layer receives the output from both the layer preceding it and the same layer, rather than from the raw input, as analogous to the way that the neurons further from the optic nerve receive signals from those closer to it.[13] The last layer produces the output of the system. RNN possesses an internal state or short-term memory due to the recurrent feedback connections, as seen by directed loops in Fig. 1. This makes RNN is suitable for modeling sequential data, especially time series. Moreover, these connections allow well-trained RNNs to regenerate dynamics of any temporal nonlinear dynamical system up to a satisfactory accuracy. Thus, RNN models are widely used to analyze diverse time series problems including dynamical systems.[14] Most of the complex RNN architectures, such as LSTM[7] and Gated Recurrent Unit,[15] can be interpreted as a variation or as an extension of the basic RNN scheme.[16]

The objective of this study is to demonstrate how well RNNs learn spatiotemporal dynamics of multi-scale dynamical systems. Our focus is either reconstruct or forecast short-term or long-term trajectories of a dynamical system of interest with given initial conditions. We use three exemplary datasets that are generated from three dynamical systems, namely, the Lorenz attractor,[17] a generalized Vicsek model,[18] and a stream flow model, which represent three diverse fields, ordinary differential equations, collective motion, and hydrological modeling, respectively. For given orbits that are generated from a Lorenz system with a formulation error, we train an RNN to eliminate the formulation error and reconstruct the correct system's responses. Then, we generate a collective motion dataset

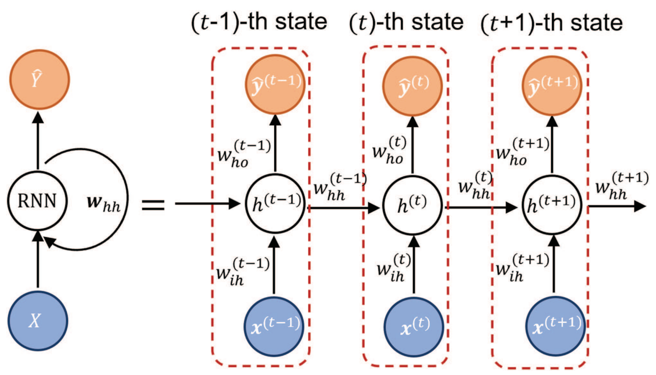

**FIG. 1.** Standard single-layer recurrent neural network (RNN), i.e., one-stacked RNN, architecture. The right-hand side schematic is the unrolled version, with respect to time, of the left-hand side RNN architecture where the directed loop represents recurrent feedback between different states of the same layer. Here, $X$, $\boldsymbol{w}_{hh}$, and $\hat{Y}$ represent input, the weights of the recurrent connections, and the output, respectively. Moreover, for the $t$th state, while $\boldsymbol{h}^{(t)}$ represents the hidden state, $w_{ih}^{(t)}$, $w_{hh}^{(t)}$, and $w_{ho}^{(t)}$ represent weights between input and hidden layer, states of the hidden layer, and hidden layer and output, respectively. Note that each state itself is a conventional artificial neural network; however, the RNN is formed by linking consecutive states within each layer with a directed edge. This recurrent feedback accords memory for an RNN through its states (i.e., time steps).

using the generalized version in Ref. 18, of the classic Vicsek model in Ref. 19, and create two copies of that, one by imposing some additive Gaussian noise and the other by deleting some segments of the trajectories. Here, our goals in these experiments are to eliminate the additive noise from the noisy trajectories. Finally, we forecast streamflow data that are naturally highly volatile and consist of frequent spiky fluctuations.

Low-rank matrix completion (LMC) is a family of linear algebraic tools in the domain of machine learning that can be adopted for low-rank completion of a given partially observed data matrix. Our previous works in Refs. 18 and 20 present that LMC methods have significant performance in the reconstruction of fragmented trajectories because the coordinate matrix representing agents' trajectories of collective motion is often low-rank due to mutual interactions and dependencies between the agents. Moreover, the orbits generated from a differential dynamical system, such as a Lorenz system, by changing the initial conditions are similar to the aforesaid collective motion agents' trajectories in terms of the low-rank nature. Thus, we conjecture that LMC can also be used to reconstruct corrupted orbits of this differential dynamical system. The state-of-the-art LMC framework given in Ref. 21 guarantees that the observed segments of the trajectories/orbits in the input coordinates matrix remain the same as those in the reconstructed dataset while each unobserved segment of a trajectory is approximated between given upper and lower bounds. In a similar vein, we have also shown in Refs. 22 and 23 that the LMC framework in Ref. 21 attains superior performance in topology recovery of both partially observed directed graphs and partially observed uncorrected graphs since graphs are naturally low-rank. In this paper, we employ the aforesaid LMC framework as an alternative technique to RNNs for the task of corrupted orbits and trajectory reconstruction and then use its results as a baseline to compare the reconstruction performance of RNNs.

This paper is structured as follows: In Sec. II A, we present the architecture and the optimization routine of RNNs for analyzing and forecasting data generated from spatiotemporal dynamical systems. In Sec. III, we validate the performance of RNNs applied to three dynamical system datasets representing three fields, ordinary differential equations, collective motion, and hydrological modeling. The analysis conducted using RNNs addresses three unique issues and solutions for them. Section IV states the discussion that includes key findings, limitations, future work, and conclusions. Table I represents the notations used in this paper along with their descriptions.

## II. METHODS

We utilize the deep learning framework RNN available in Ref. 6 to reconstruct and forecast the dynamics of diverse systems, and then compare RNNs' reconstruction performance with that of the LMC framework available in Ref. 21. Thus, first, we present the formulation of RNNs in Sec. II A and, then, present the LMC technique in Sec. II B.

### A. Recurrent neural networks

RNNs are a major class of machine learning tools that are particularly well suited to work on sequential data. Connections

**TABLE I.** Notations used in this paper and their descriptions.

| Notation | Description |
|---|---|
| $N_i$ | Number of nodes in the input layer of an RNN |
| $N_h$ | Number of nodes in the hidden layer of an RNN |
| $N_o$ | Number of nodes in the output layer of an RNN |
| $N_1$ | Total number of training orbits, trajectories, and agents |
| $N_2$ | Total number of testing orbits, trajectories, and agents |
| $K$ | Total number of layers |
| $t$ | Index for time steps where $1 \leq t \leq T$ |
| $k$ | Index for the RNN's layers where $1 \leq k \leq K$ |
| $n$ | Index for orbits, trajectories, or agents where $1 \leq n \leq N$ |
| $\sigma, \rho, \beta$ | Parameter in Lorenz |
| $T$ | Number of total time steps |
| $\delta$ | Step size of the generalized Vicsek model |
| $\eta$ | Corruption level of orbits |
| $\sigma$ | Standard deviation of Gaussian noise distribution |
| $t$ | Index for time steps where $1 \leq t \leq T$ |
| $\epsilon_i^{(t)}$ | Noise parameter imposed to the $i$th agent at the $t$th time step |
| $\theta_i^{(t)}$ | Orientation of the $i$th agent at the $t$th time step |
| $r_d$ | Radius of interaction |
| $r^{(t)}$ | Radius of spiral example |
| $k^{(t)}$ | Angle of rotation of the a spiral at time $t$ from the initial position |
| $N_i^{(t)}$ | $r$-radius neighborhood of the $i$th agent at the $t$th time step |
| $\boldsymbol{x}^{(t)}$ | Input vector at time step $t$ |
| $\boldsymbol{h}_k^{(t)}$ | Hidden state of the $k$th layer for the $t$th time step |
| $\boldsymbol{y}^{(t)}$ | Label vector at time step $t$ |
| $\boldsymbol{b}_i$ | Bias vector for the input layer |
| $\boldsymbol{b}_h$ | Bias vector for the hidden layer |
| $\boldsymbol{x}_n^{(t)}$ | Input vector of $n$th orbit at time step $t$ ($t$th point of $n$th orbit) |
| $\boldsymbol{y}_n^{(0)}$ | Initial condition of $n$th orbit |
| $\boldsymbol{y}_n^{(t)}$ | Label vector of the $n$th orbit at time step $t$ |
| $\hat{\boldsymbol{y}}_n^{(t)}$ | Output vector of the $n$th orbit at time $t$ corresponding to $\boldsymbol{x}_n^{(t)}$ |
| $\boldsymbol{u}_i^{(t)}$ | Average direction of motion of the agents in the neighborhood of $N_i^{(t)}$ |
| $\boldsymbol{c}^{(t)}$ | The $t$th point on the centroid of the spiral |
| $\boldsymbol{v}_i^{(t)}$ | Velocity of the $i$th agent at the $t$th time step |
| $X$ | Set of all input data |
| $\hat{Y}$ | Set of all output |
| $Y$ | Set of all label |
| $f$ | Linear transformation in RNN |
| $g$ | Non-linear transformation in RNN |
| $L(\cdot)$ | Loss function |
| $\top$ | Matrix transpose |

TABLE I. *(Continued.)*

| Notation | Description |
|---|---|
| $W_{ih}$ | Weight matrix between the input layer $i$ and the hidden layer $h$ |
| $W_{h_k h_{k'}}$ | Weight matrix between the hidden layer $h_k$ and the hidden layer $h_{k'}$ |
| $W_{ho}$ | Weight matrix between the hidden layer $h$ and the output layer $o$ |
| $R_j^{(t)}$ | Rotation matrix of the $j$th agent at the $t$th time step |

between nodes form a directed graph along a temporal sequence that can be used to learn nonlinear dependencies of a system over time.[24] In its most general form, an RNN can be seen as a weighted directed graph that contains three different kinds of nodes, namely, the input nodes, hidden nodes, and output nodes.[25] As seen in Fig. 1, where we represent the structure of a simple RNN with one layer, input nodes do not have incoming connections and output nodes do not have outgoing connections, but hidden nodes have both. By RNN's design, two different nodes that are either at the same time level or at different time levels can be connected by an edge, as seen in Fig. 2.

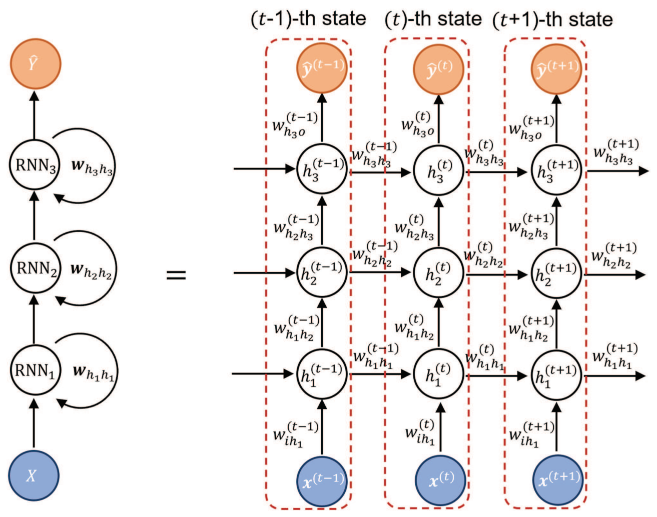

FIG. 2. Standard stacked recurrent neural network (RNN) architecture of having three layers, i.e., three-stacked RNN. The right-hand side schematic is the unrolled version, with respect to time, of the left-hand side RNN architecture where directed loops represent recurrent feedback between different states of the same layer. Here, $X$, $\boldsymbol{w}_{h_kh_k}$, and $Y$ represent input, the weights of the recurrent connections of the $k$th layer, and the output. For the $t$th state, where $1 \leq t \leq T$ for some $T$, $\boldsymbol{x}^{(t)}$, $\boldsymbol{h}_k^{(t)}$, and $\hat{\boldsymbol{y}}^{(t)}$ represent the input vector, the hidden state of the $k$th layer where $1 \leq k \leq K$ for some $K$, and the output vector, respectively. Moreover, $w_{ih_1}^{(t)}$, $w_{h_kh_{k'}}^{(t)}$, and $w_{h_3o}^{(t)}$ are the weights between input and hidden layer 1, hidden layer $h_k$ and hidden layer $h_{k'}$, and hidden layer 3 and output, respectively. For the $k$th layer, $w_{h_kh_k}^{(t)}$ represents the weight between two consecutive states.

An RNN has a memory that stores *knowledge* about the data sequence that it has already seen, however, its memory of internal states through iterations is short-term.[26] By applying activation functions (also called, transfer functions) to previous states and inputs, RNNs compute new states. Typical activation functions in RNNs include sigmoid, hyperbolic tangent, and ReLU; however, sigmoid and hyperbolic tangent encounter vanishing of the gradient in the optimization step.[27] Properly customized RNNs with the best activation function and the best number of hidden nodes can capture the dynamic of any nonlinear system up to a high precision.[28] The model capacity and flexibility for learning non-linear systems can be increased by a multi-layer recurrent structure.[29] By this multi-layer structure, the hidden output of one recurrent layer can be propagated through time and is used as the input data to the next recurrent layer. In our study, we used three-stacked RNNs (or recurrent layers) as seen in Fig. 2.

### 1. Architecture of RNNs

For simplicity, we formulate an RNN with only one hidden layer, i.e., a one-stacked RNN, as shown in Fig. 1. We denote the number of nodes in the input layer, hidden layer, and output layer as $N_i$, $N_h$, and $N_o$. For the $t$th state, the weight matrix from input to hidden layer is denoted as $w_{ih}^{(t)} \in \mathbb{R}^{N_i \times N_h}$, the weight matrix between hidden layer as $w_{hh}^{(t)} \in \mathbb{R}^{N_h \times N_h}$, and the weight matrix from the hidden layer to the output layer as $w_{ho}^{(t)} \in \mathbb{R}^{N_h \times N_o}$. Moreover, for the $t$th state, we denote the bias vector for the input layer as $\boldsymbol{b}_i^{(t)} \in \mathbb{R}^{N_i}$ and the bias vector for the hidden layer as $\boldsymbol{b}_h^{(t)} \in \mathbb{R}^{N_h}$. Here, we develop the formulation for the $t$th state, where $1 \leq t \leq T$ for some $T$, i.e., for the $t$th ANN, in the RNN. The input $\boldsymbol{x}^{(t)}$ of the $t$th state is multiplied with $w_{ih}^{(t)}$, and summed that with $\boldsymbol{b}_i^{(t)}$, i.e., $w_{ih}^{(t)}\boldsymbol{x}^{(t)} + \boldsymbol{b}_i^{(t)}$, to get the flow of information from the input to the hidden layer. Similarly, the information flow by the recurrent feedback from the hidden node of the $(t-1)$th state to that of the $t$th state is $w_{hh}^{(t-1)}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h^{(t-1)}$. The hidden state $\boldsymbol{h}^{(t)}$ of the $t$th state is the sum of the above two information flows passing through a nonlinear transformation, say $f$, such that

$$\boldsymbol{h}^{(t)} = f\left(w_{ih}^{(t)}\boldsymbol{x}^{(t)} + \boldsymbol{b}_i^{(t)} + w_{hh}^{(t-1)}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h^{(t-1)}\right), \tag{1}$$

where $f$ is the activation function of the neurons which is usually sigmoid or hyperbolic tangent. The information flow from the hidden layer of the $t$th state to its output, i.e., $w_{ho}^{(t)}\boldsymbol{h}^{(t)} + \boldsymbol{b}_o$, is composed with another transformation, say $g$, to get the output, say $\hat{\boldsymbol{y}}^{(t)}$, of the RNN such that,

$$\hat{\boldsymbol{y}}^{(t)} = g\left(w_{ho}^{(t)}\boldsymbol{h}^{(t)} + \boldsymbol{b}_o^{(t)}\right), \tag{2}$$

where $g$ is linear in general. For all $t$'s, $\boldsymbol{h}^{(t)}$ is usually initialized with a vector of zeros. The explicit correspondence of Eqs. (1) and (2) with the RNN's architecture, helps us adopt spatiotemporal notion of RNNs using so-called *finite unfolding in time* with shared weight matrices.[30]

By learning the formulation pattern in Eqs. (1) and (2), we derive the formulation for a stacked RNN with $K$ hidden layers. Note that Fig. 2 provides the architecture for such an RNN with $K = 3$, i.e., three-stacked RNN. For the $t$th time step, let $\boldsymbol{x}^{(t)}$, $\boldsymbol{h}_k^{(t)}$, and $\boldsymbol{y}^{(t)}$ represent the input vector, the hidden state of the $k$th layer, and the output vector, respectively. We denote the number of nodes in the input layer, $k$th hidden layer, and output layer as $N_i$, $N_{h_k}$, and $N_o$. Moreover, $w_{ih_1}^{(t)} \in \mathbb{R}^{N_i \times N_{h_1}}$, $w_{h_k h_{k'}}^{(t)} \in \mathbb{R}^{N_{h_k} \times N_{h'_k}}$, and $w_{h_3 o}^{(t)} \in \mathbb{R}^{N_{h_3} \times N_o}$ are the weights between the input and hidden layer 1, hidden layer $h_k$ and hidden layer $h_{k'}$, and hidden layer 3 and the output, respectively. Similar to Eq. (1), the $t$th state of the first hidden layer of the RNN is sent through a nonlinear activation function $f$ along with both the input $\boldsymbol{x}^{(t)}$ and the $(t-1)$th state of the first hidden layer, $\boldsymbol{h}_1^{(t-1)}$, as

$$\boldsymbol{h}_1^{(t)} = f\left(w_{ih_1}^{(t)} \boldsymbol{x}^{(t)} + \boldsymbol{b}_i^{(t)} + w_{h_1 h_1}^{(t-1)} \boldsymbol{h}_1^{(t-1)} + \boldsymbol{b}_{h_1}^{(t-1)}\right), \tag{3}$$

where $\boldsymbol{b}_i^{(t)}$ is the bias vector for the input at the $t$th state and $\boldsymbol{b}_{h_1}^{(t-1)}$ is the bias vector for the first hidden layer at the $t$th state. The hidden state, $\boldsymbol{h}_k^{(t)}$ for $k = 2, 3, \ldots, K$ of the $t$th state is based on both its $t$th state at the previously hidden layer, i.e., $\boldsymbol{h}_{k-1}^{(t)}$, and the $(t-1)$th state of the same hidden layer, i.e., $\boldsymbol{h}_k^{(t-1)}$, as

$$\boldsymbol{h}_k^{(t)} = f\left(w_{h_{k-1} h_k}^{(t)} \boldsymbol{h}_{k-1}^{(t)} + \boldsymbol{b}_{h_{k-1} h_k}^{(t)} + w_{h_k h_k}^{(t-1)} \boldsymbol{h}_k^{(t-1)} + \boldsymbol{b}_{h_k h_k}^{(t-1)}\right), \tag{4}$$

where $\boldsymbol{b}_{h_{k-1} h_k}^{(t)}$ is the bias vector for the $(k-1)$th hidden later at the $t$th state and $\boldsymbol{b}_{h_k h_k}^{(t-1)}$ is the bias vector for the $k$th hidden layer at the $(t-1)$th state. The output of the $t$th state, say $\hat{\boldsymbol{y}}^{(t)}$, is only based on its $K$th hidden layer such that,

$$\hat{\boldsymbol{y}}^{(t)} = g\left(w_{h_K o}^{(t)} \boldsymbol{h}_K^{(t)} + \boldsymbol{b}_{h_K o}^{(t)}\right), \tag{5}$$

where $\boldsymbol{b}_{h_K o}^{(t)}$ is the bias vector for the $K$th hidden layer at the $t$th state. The stacked RNN's initial hidden states $\boldsymbol{h}_k^{(0)}, \ldots, \boldsymbol{h}_k^{(T)}$ should be initialized for each layer $k$. All the weight matrices and bias vectors are optimized using the gradient descent method according to a straightforward backpropagation through time (BTT) procedure.[31]

### 2. Optimization of RNNs

Training an RNN is mathematically implemented as a minimization of a relevant reconstruction error, widely called the loss, function with respect to weights and bias vectors of Eqs. (3)–(5). This optimization is carried out in four steps: first, forward propagation of input data through the neural network to get the output; second, calculate the loss between forecasted output and the expected output; third, calculate the derivatives of the loss function with respect to the ANN's weights and bias vectors using BTT; and fourth, adjusting the weights and bias vectors by gradient descent method.[32]

BTT unrolls backward all the dependencies of the output on the weights of the system,[33] which is represented from the left side to the right side in Figs. 1 and 2. We train the RNN by one data instance, say instant $n$, at a time; thus, we formulate the loss function of RNNs as a normalization over all the training data instances, say $N_1$. We denote the $n$th input data instance as $\boldsymbol{x}_n = [\boldsymbol{x}_n^{(1)}, \ldots, \boldsymbol{x}_n^{(T)}] \in X$, where $X$ represents the set of all the input data, and the corresponding output as $\hat{\boldsymbol{y}}_n = [\hat{\boldsymbol{y}}_n^{(1)}, \ldots, \hat{\boldsymbol{y}}_n^{(T)}] \in \hat{Y}$, where $\hat{Y}$ represents the set of all the outputs. We denote the label of the $n$th input data as $\boldsymbol{y}_n = [\boldsymbol{y}_n^{(1)}, \ldots, \boldsymbol{y}_n^{(T)}] \in Y$, where $Y$ represents the set of all the labels for the input data. The loss function, denoted by $L$, is defined as the normalized squared difference between labels and RNN's output,

$$L\left(\hat{Y}, Y\right) = \frac{1}{|Y|} \sum_{y_n \in Y} (\hat{\boldsymbol{y}}_n - \boldsymbol{y}_n)^2, \tag{6}$$

where the cardinality $|Y|$ is defined as the number of data instances in the set $Y$. We use BTT to compute the derivatives of Eq. (6) with respect to the weights and bias vectors. We update the weights using the gradient descent based method, called Adaptive Moment Estimation (ADAM).[34] ADAM is an iterative optimization algorithm that is widely used in modern machine learning algorithms to minimize loss functions where it employs the averages of both the first moment gradients and the second moment of the gradients for computations. ADAM generally converges faster than standard gradient descent methods and saves memory by not accumulating the intermediate weights. After the training is over, we input the erroneous testing orbits or noisy testing trajectories, a total of $N_2$, and compute their corresponding true representations.

## B. Low-rank matrix completion

Here, we present the formulation of the LMC technique available in Ref. 21 since it is the baseline for comparison against RNNs. In this study, each training orbit, i.e., the true orbit, or each training trajectory, i.e., the true trajectory, of a total of $N_1$, consists $T$ total time steps such that the $n$th orbit or trajectory at the $t$th time step is denoted by $\boldsymbol{y}_n^{(t)} = \left[y_{n,1}^{(t)}, y_{n,2}^{(t)}, y_{n,3}^{(t)}\right] \in \mathbb{R}^3$. The vector representing all the coordinates of an orbit or a trajectory is denoted as

$$\boldsymbol{y}_n = \left[\boldsymbol{y}_n^{(1)}, \ldots, \boldsymbol{y}_n^{(T)}\right]^\top \in \mathbb{R}^{3T \times 1}, \tag{7}$$

where $\top$ denotes the transpose. The dataset of all the true orbits or the true trajectories is given as

$$Y = \left[\boldsymbol{y}_1 | \cdots | \boldsymbol{y}_{N_1}\right] \in \mathbb{R}^{3T \times N_1}, \tag{8}$$

which are the training labels for the RNNs in Sec. II A. Similarly, we denote the total set of $N_2$ erroneous testing orbits or noisy testing trajectories as $X$ such that

$$X = \left[\boldsymbol{x}_1 | \cdots | \boldsymbol{x}_{N_2}\right] \in \mathbb{R}^{3T \times N_2}, \tag{9}$$

which is the input test data for the RNN in Sec. II A. The data matrix for LMC that we denote as $\mathcal{D}$ consists of both the true data given in Eq. (8) and erroneous data given in Eq. (9) such that

$$\mathcal{D} = [X | Y] \in \mathbb{R}^{3T \times (N_1 + N_2)}. \tag{10}$$

The LMC scheme in Ref. 21 inputs a low-rank partially observed or unobserved feature matrix, coordinates matrix $\mathcal{D}$ in our case here, and reconstructs the partially observed or unobserved entries of that matrix by utilizing optimization techniques. Using this LMC technique, we are interested in a low-rank reconstruction of erroneous orbits and noisy trajectories. The LMC scheme in Ref. 21 decomposes the data matrix $\mathcal{D}$ into a low-rank matrix, denoted as $L$, and a sparse matrix, denoted as $S$, by imposing that the reconstructed matrix, i.e., $(L+S)$, satisfies user-input lower bound matrix $\mathcal{B}_l = \left[\boldsymbol{b}^l_1|\cdots|\boldsymbol{b}^l_n|\cdots|\boldsymbol{b}^l_{(N_1+N_2)}\right]_{3T\times(N_1+N_2)}$ and upper bound matrix $\mathcal{B}_u = \left[\boldsymbol{b}^u_1|\cdots|\boldsymbol{b}^u_n|\cdots|\boldsymbol{b}^u_{(N_1+N_2)}\right]_{3T\times(N_1+N_2)}$. Specifically, while we set the lower bound equal to the upper bound for the true orbits or true trajectories, we set carefully chosen unequal upper and lower bounds for the erroneous orbits or noisy trajectories. In this setup, the recovered matrix consists of the same coordinates for the true orbits or trajectories while it consists of LMC approximated the true versions for the erroneous orbits or noisy trajectories. Let $\lambda \in \mathbb{R}$ is a regularization parameter, the optimization scheme of this LMC technique is

$$\arg\min_{L,S} \|L\|_* + \lambda\|S\|_1, \quad \text{s.t. } \mathcal{B}_l \preceq \mathcal{D} - L - S \preceq \mathcal{B}_u, \tag{11}$$

where $\|\cdot\|_*$, $\|\cdot\|_1$, and $\preceq$ denote nuclear norm given in Definition 2, $L_1$ norm given in Definition 3, and pointwise inequality, respectively.[21] The optimization problem in Eq. (11) is *convex* and can efficiently be solved for a big number of orbits or trajectories with many time steps on commodity computing hardware using splitting techniques and iterative matrix decomposition algorithms.[21] Numerical implementation of the optimization problem in Eq. (11) is performed in two steps. First, we write the *augmented Lagrangian function* of Eq. (11). Second, we optimize the augmented Lagrangian function using the *Alternating Direction Method of Multipliers*.[35]

**Definition 1** Consider that $\text{Diag}(\sigma_1, \ldots \sigma_{\min(n,m)})$ represents a diagonal matrix formed by the vector $(\sigma_1, \ldots \sigma_{\min(n,m)})$ as its diagonal. Let $L \in \mathbb{R}^{n\times m}$ be a real-valued matrix, and $U_{m\times m}$ and $V_{n\times n}$ are two unitary matrices such that $U^\top U = I$ and $V^\top V = I$, respectively. Then, singular value decomposition (SVD) of $L$ is $L = U\Sigma V^\top$, where $\Sigma_{m\times n} = \text{Diag}(\sigma_1, \ldots \sigma_{\min(n,m)})$. Here, for $j = 1, \ldots, \min(n,m)$, $\sigma_j$ represents $j$th singular value (SV) of $L$ and $\sigma_j \geq \sigma_{j+1}$; $\forall j$.

**Definition 2** For a given real-valued matrix $L \in \mathbb{R}^{m\times n}$, the nuclear norm of $L$, denoted by $\|L\|_*$, is defined as

$$\|L\|_* = \sum_{j=1}^{n} \sigma_j, \tag{12}$$

where $\sigma_j$ denotes the $j$th SV of $L$ computed using Definition 1.

**Definition 3** Let $S = [s_{ij}] \in \mathbb{R}^{m\times n}$ is a real-valued matrix, then the $L_1$ norm of $S$, denoted by $\|S\|_1$, is defined as the max of the column sums such that

$$\|S\|_1 = \max_j \sum_i^n |s_{ij}|. \tag{13}$$

This research focuses on the reconstruction of the true orbits or trajectories from their corrupted ones. We only retain $L$ of Eq. (11) while ignoring $S$ since orbits or trajectories are only low-rank and do not present the sparse component. Thus, we set $\lambda = 0$ when we implement the LMC algorithm in Ref. 21. The last $N_2$ columns of the recovered matrix $L$ represent the approximations of the true trajectories for the corresponding erroneous orbits or noisy trajectories. We use these reconstructions to compare the performance of the RNNs' reconstructions.

## III. PERFORMANCE ANALYSIS

In this section, we seek the solutions for the following three diverse problems using RNNs: (1) a Lorenz system having a formulation error, (2) denoising of noisy trajectories of particle swarms in collective motion, and (3) forecasting of groundwater streamflow of hydrologic catchments. These three problems, in that order, represent three fields, ordinary differential equations, collective motion, and hydrological modeling. The RNN's solutions for the first and the second problems are compared with the solutions generated by an LMC technique available in Ref. 21. The RNN's solution for the third problem is compared with the solutions of a hydrological modeling scheme called GR4J available in Ref. 36.

Here, we use the RNN model available in the machine learning library *Pytorch*,[37] which requires the parameter inputs sequential length, input size, learning rate, output size, number of RNNs, and number of hidden nodes. The sequential length is the length of the input data sequence, which is $T$ in Fig. 2. The input size is the number of features or dimensions in the dataset, which is the length of the vector $\boldsymbol{x}^{(t)}$ in Fig. 2. The learning rate is associated with the ADAM optimization routing where a large learning rate allows the RNN to learn faster at a cost of arriving into a sub-optimal final set of weights. A smaller learning rate may allow the RNN to learn further optimal or even globally optimal set of weights but may take significantly longer time to train. The number of RNNs is the number of basic RNN units that are stacked together which is equal to the number of hidden layers, where it is $K = 3$ in Fig. 2. The number of hidden nodes is the user input number of nodes at each hidden layer of the RNN, which is $N_{h_k}$ for $k$th hidden layer of Fig. 2. Finding both the optimal number of RNNs and the optimal number of hidden nodes could prevent possible over-fitting or under-fitting scenarios of the RNN's training process.

### A. Lorenz system with a formulation error

The Lorenz system is a chaotic dynamical system[38] that is formulated as a set of ordinary differential equations, which is known for having chaotic solutions for certain parameter values and initial conditions.[39] Here, we assume that we are given a Lorenz system with a formulation error so our task is to make an ANN to generate the true solutions related to the input erroneous solutions generated by the system with the formulation error. The standard Lorenz system of the temporal variable $t$ and three spacial variables $y_1$, $y_2$, and $y_3$ is given by

$$\begin{aligned} \frac{dy_1}{dt} &= \sigma(y_2 - y_1), \\ \frac{dy_2}{dt} &= y_1(\rho - y_3) - y_2, \\ \frac{dy_3}{dt} &= y_1 y_2 - \beta y_3, \end{aligned} \tag{14}$$

where $\sigma$, $\rho$, and $\beta \in \mathbb{R}$ are parameters. To mimic a formulation error, we eliminate the term $-y_2$ from Eq. (14) as

$$\frac{dx_1}{dt} = \sigma(x_2 - x_1),$$
$$\frac{dx_2}{dt} = x_1(\rho - x_3), \qquad (15)$$
$$\frac{dx_3}{dt} = x_1 x_2 - \beta x_3.$$

Note that, we replaced the variable $y$ in Eq. (14) with variable $x$ to distinguish data from two systems. We set $\sigma = 3$, $\rho = 28$, and $\beta = 8/3$ in both Eqs. (14) and (15) since the solutions with these parameters are well-known for showing chaotic behaviors. For different initial conditions, we solve the system in Eq. (14) and treat them as training labels for the RNN. Now, we generate the erroneous solutions for Eq. (15) based on Eq. (14) to use as training data as we will explain in the sequel. We use the trained RNN to generate the true solutions for the input erroneous solutions.

We generate 500 random initial conditions, $\boldsymbol{y}_n^{(0)} = \left(y_{n,1}^{(0)}, y_{n,2}^{(0)}, y_{n,3}^{(0)}\right)$ where $1 \le n \le 500$, from the uniform distribution $\mathbb{U}[-15, 15]$. We solve Eq. (14) and generate 500 orbits, denoted as $N$ with the step size, denoted as $\delta$, is 0.01, and the total time steps, denoted as $T$, is 5000 (i.e., $t \in [0, 50]$ of Eqn. (14)). Here, the $n$th orbit is given as $\boldsymbol{y}_n = \left[\boldsymbol{y}_n^{(0)}, \ldots, \boldsymbol{y}_n^{(t)}, \ldots, \boldsymbol{y}_n^{(5000)}\right]$, where $\boldsymbol{y}_n^{(t)} = \left[y_{n,1}^{(t)}, y_{n,2}^{(t)}, y_{n,3}^{(t)}\right]^\top$. Now, we explain the process of generating the erroneous orbit $\boldsymbol{x}_n = \left[\boldsymbol{x}_n^{(0)}, \ldots, \boldsymbol{x}_n^{(t)}, \ldots, \boldsymbol{x}_n^{(5000)}\right]$ from the above true orbit. For some $t$, we treat $\boldsymbol{y}_n^{(t)}$ in the $n$th orbit as an initial condition for the system in Eq. (15) and solve it for some time steps, denoted as $\eta$, where $\eta < T$, with the same values for the parameters $\sigma$, $\rho$, $\beta$, and $\delta$ as above. The parameter $\eta$ governs the corruption level of the orbits such that a big value of $\eta$ contributes a big level of corruption to the orbits. We treat the last point of the solution set, that is, the $\eta$th point, as the erroneous solution relevant to the correct point $\boldsymbol{y}_n^{(t)}$, that we denote by $\boldsymbol{x}_n^{(t)}$ (see Fig. 3). We carry out this process for each $n$ and generate the entire erroneous orbit $\left[\boldsymbol{x}_n^{(0)}, \ldots, \boldsymbol{x}_n^{(t)}, \ldots, \boldsymbol{x}_n^{(5000)}\right]$. Similarly, we compute the erroneous orbits corresponding to each of the 500 true orbits that we generated by changing initial conditions. The first 400 orbits of the system in Eq. (15) are chosen as the training set, and the corresponding 400 orbits that are derived from the system in Eq. (14) are chosen as their corresponding labels.

First, we set $\eta = 1$ and generate 500 true and erroneous orbits each with the 5000 time steps as explained above, where the first 400 orbits are used for training while the last 100 orbits are used for testing. In order to avoid both the over-fitting and the semi-convergence of RNN, the best number of training epochs was chosen by running the RNN twice as follows: first, we run the RNN with a fixed, but big, number of epochs to find the epoch number that gives the minimum loss; second, we re-run the RNN for that many epochs to complete the training. For the Lorenz system, based on the corruption level $\eta$, the optimum number of epochs ranges between 179 and 930. We utilize the activation function ReLU as it improves the reconstruction error about 10 times than that of the hyperbolic tangent activation function. In order to find the optimal parameters, which are learning rate, number of RNN, number of hidden nodes; first, we

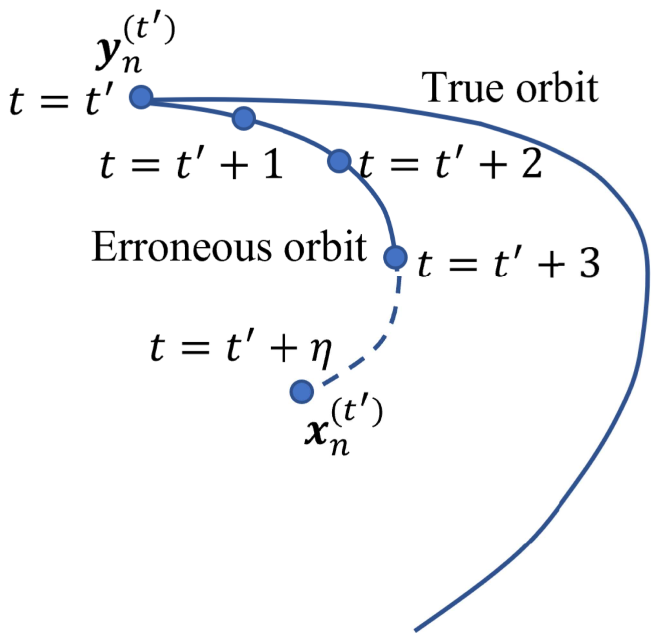

FIG. 3. For some $t'$, where $1 \le t' \le T$, we treat $\boldsymbol{y}_n^{(t')}$ in the $n$th true orbit as the initial condition for the erroneous Lorenz system and solve it for some time steps, denoted as $\eta$, with the same parameter values used for solving the true Lorenz system. We consider the last point $\eta$, denoted by $\boldsymbol{x}_n^{(t')}$, of the solution set as the erroneous solution relevant to the true point $\boldsymbol{y}_n^{(t')}$. We carry out this process for each $t'$ to generate the corrupted orbit $\left[\boldsymbol{x}_n^{(0)}, \ldots, \boldsymbol{x}_n^{(t)}, \ldots, \boldsymbol{x}_n^{(T)}\right]$. Here, $\eta$ can be considered as the corruption level of the orbit where a big $\eta$ contributes a high corruption.

discretize the parameter space; then, we train and test the RNN with those discretized parameter combinations; finally, we select the best values for the parameters that give the least reconstruction error. In all the experiments, a data batch is equal to the entire dataset. We set the other parameters of RNN as follows: sequential length to 5000 since each orbit has 5000 time steps; input size to three since each orbit is three-dimensional; output size to three since the output is also three-dimensional; the learning rate of the Lorenz orbits to 0.01.

We set the RNN to three hidden recurrent layers with 256 nodes each. For each $1 \le n \le 400$, we input each $\boldsymbol{x}_n$ into the RNN one at a time and generate the corresponding output of the RNN, denoted as $\hat{y}_n$. We compute the reconstruction error $L$ between the outputs and the labels using Eq. (6). We use BTT to compute the derivatives of Eq. (6) with respect to the weights and the bias vectors and update the weights and the bias vectors using ADAM as explained in Sec. II A 2. The remaining 100 test orbits, $\boldsymbol{x}_n$ where $401 \le n \le 500$, are fed into this trained RNN one orbit at a time and obtain the recovered solutions, $\hat{\boldsymbol{y}}_n^{(t)}; 401 \le n \le 500$. We repeat the same experiment nine more times with $\eta = 2, 3 \ldots, 9$. Figures 4(a)–4(i) show two-dimensional projections of the three-dimensional true, erroneous, and RNN's recovered orbits for the cases $\eta = 1$, 5, and 9.

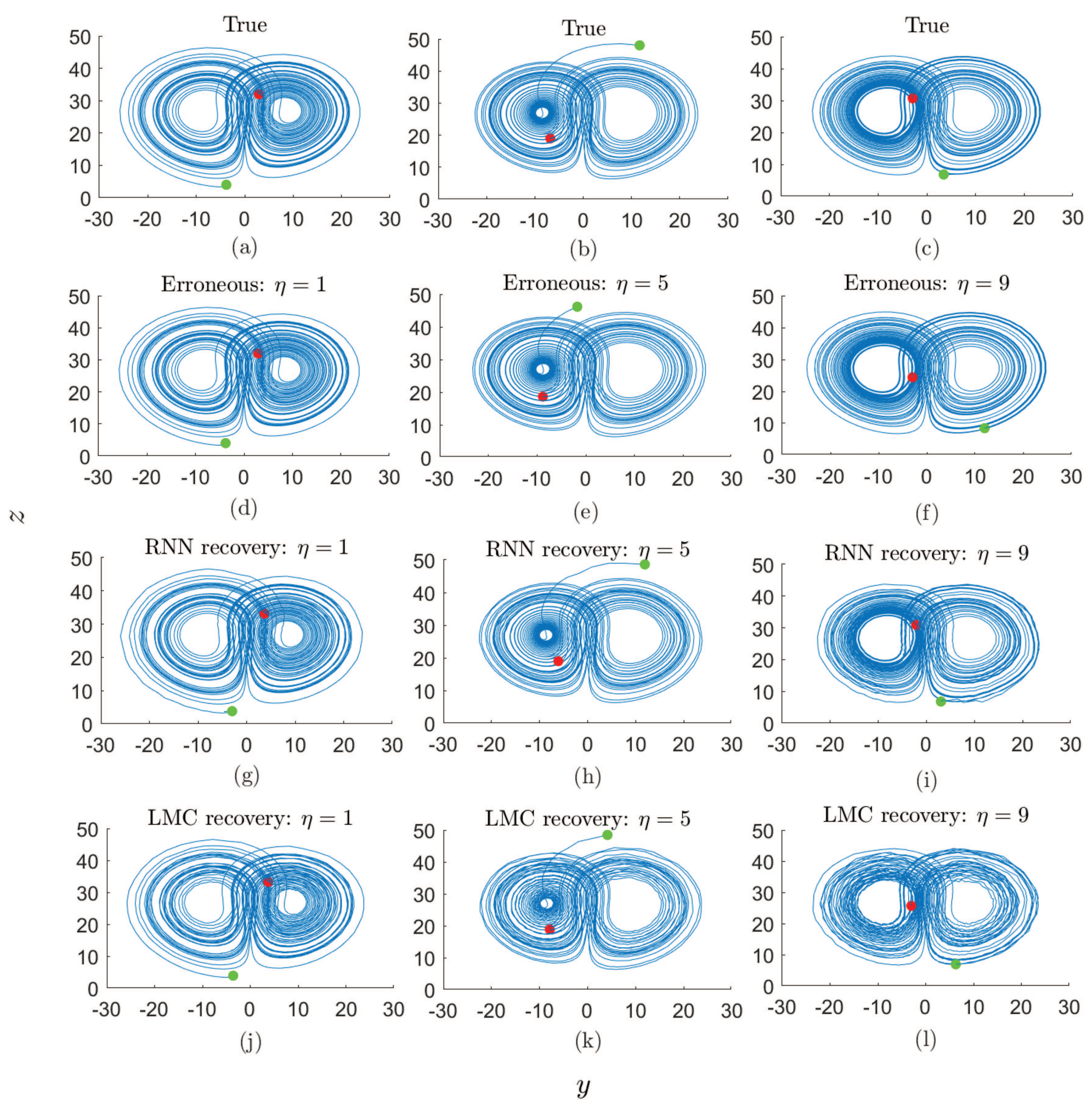

**FIG. 4.** Recovery of the true solution of an erroneous Lorenz system using RNNs. Here, all the figures are two-dimensional ($y$-$z$ plan) projections of the three-dimensional orbits and $\eta$ represents the corruption level. Here, while (a)–(c) show three arbitrarily chosen true orbits, (d)–(f) show the erroneous orbits corresponding to them, respectively, where the green disks represent the starting point and red disks represent the ends. First, we train an RNN and then we pass the erroneous orbits through the RNNs and recover them as shown in (g)–(i); then, we use LMC and recover the erroneous orbits as shown in (j)–(l). The start and the end points of the recovered orbits by RNN are close to those of the true orbits whereas the start and the end points of the recovered orbits by LMC are far apart from those of the true orbits. Root mean square errors of the reconstructions are $(RNN, LMC) = (0.07, 0.09)$, $(0.15, 0.32)$, and $(0.27, 0.65)$ for the corruption levels $\eta = 1, 5$, and 9, respectively.

As the baseline for the comparison against the orbit reconstructed by RNNs, we recover the same erroneous orbits using LMC that we presented in Sec. II B. For that, the first step is to prepare the lower bound matrix $\mathcal{B}_l = \left[\boldsymbol{b}_1^l|\cdots|\boldsymbol{b}_n^l|\cdots|\boldsymbol{b}_{(N_1+N_2)}^l\right]$ and the upper bound matrix $\mathcal{B}_u = \left[\boldsymbol{b}_1^u|\cdots|\boldsymbol{b}_n^u|\cdots|\boldsymbol{b}_{(N_1+N_2)}^u\right]$ of Eq. (11). For a column, say $\boldsymbol{d}_n$, representing a true orbit in $\mathcal{D} = \left[\boldsymbol{d}_1|\cdots|\boldsymbol{d}_n|\cdots|\boldsymbol{d}_{(N_1+N_2)}\right]$ of Eq. (10), i.e., a column in $Y$, we set the corresponding columns of $\mathcal{B}_l$ and $\mathcal{B}_u$ to be equal, i.e., $\boldsymbol{b}_n^l = \boldsymbol{b}_n^u$. Conversely, for a column, say $\boldsymbol{d}_n$, representing an erroneous orbit in $\mathcal{D}$ of Eq. (10), i.e., a column in $X$, we set a lower bound $\boldsymbol{b}_n^l$ as $\boldsymbol{d}_n - \zeta$ and an upper bound $\boldsymbol{b}_n^u$ as $\boldsymbol{d}_n + \zeta$, where $\zeta \in \mathbb{R}^+$ is a parameter that guarantees the true orbit is between $\boldsymbol{d}_n - \zeta$ and $\boldsymbol{d}_n + \zeta$. Since the orbits generated from the same dynamical system are low-rank and our goal is to recover the true orbits corresponding to the erroneous orbits, this problem is considered to be a low-rank matrix completion problem. Thus, we only require the low-rank component $L$ of Eq. (11), which we implement by setting the regularization parameter $\lambda$ to zero. For each corruption level $\eta$, we implement this LMC technique a few times with different $\zeta$'s until we find the best $\zeta$ giving the best reconstruction. The columns in $L$ corresponding to the erroneous columns in $\mathcal{D}$ are the low-rank reconstructions. Figures 4(j)–4(l) show two-dimensional projections of the three-dimensional LMC's recovered orbits for the cases $\eta = 1, 5,$ and 9.

We observe in Figs. 4(d)–4(f) that due to the formulation error of the Lorenz system, the erroneous orbits show some shifting of the initial and the end points from their true locations. This shift does not only occur at the two extremes but all the intermediate points are also shifted along the orbit in the same direction. We observe in Figs. 4(g)–4(i) that the recovered orbits by RNNs have their extreme points around their true locations. However, Figs. 4(j)–4(l) show that the recovered trajectories by LMC have extreme points away from their true locations. Moreover, the LMC's recovery quality gets worse than that of RNN's recovery quality, as $\eta$ increases. To analyze the recovery performance with increasing corruption levels, we compute the normalized root mean square error, denoted by RMSE, between the recovered and the true orbits as

$$RMSE = \sqrt{\frac{1}{|Y|} \sum_{\forall y_n \in Y} \|\boldsymbol{y}_n - \hat{\boldsymbol{y}}_n\|_F^2}, \tag{16}$$

where $|Y|$ denotes the carnality, i.e., the number of observations

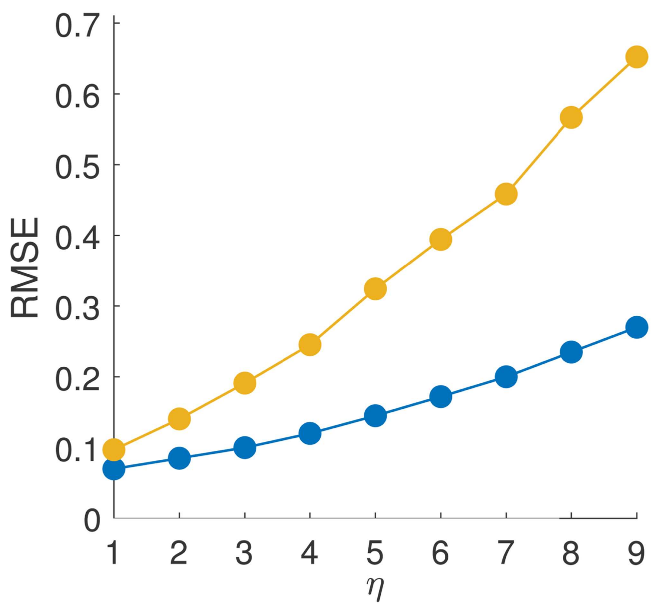

**FIG. 5.** Root mean squared error (RMSE) of the reconstructions, of corrupted Lorenz orbits, performed by RNNs (blue) and LMC (yellow) vs different levels of corruption ($\eta$). When the corruption level increases, while the RMSE associated with RNN increases slowly that of LMC increases rapidly.

in the test set, and $F$ denotes the Frobenius norm. Figure 5 shows the RMSE of the recovery of both RNN and LMC with respect to the corruption level. Here, we see that the reconstruction error increases rapidly for LMC with respect to the increase of the corruption level whereas that increment is not much significant for the reconstruction by RNNs.

## B. Noisy trajectories of collective motion

Analysis of the trajectories of collectively moving agents, such as fish[40] and birds,[41] is a highly active field in computer vision. Tracked trajectories of such agents are often contaminated with noise due to the causes like insufficient camera precision and lack of accuracy of the tracking method.[20] Here, we use RNNs to reconstruct the true collective motion trajectories from observed noisy trajectories. We simulate the collective motion trajectories using a modified version of a classic self-propelled particle model, named the Vicsec model.[19]

Collective motion is defined as a spontaneous emergence of the ordered movement in a system consisting of many self-propelled agents. The Vicsek model given in Ref. 19 is one of the widely used models that describe such behavior. This model performs simulations on a square-shaped region with periodic boundary conditions. We generate a synthetic collective motion dataset by using the generalized version presented in Ref. 18 of the classic Vicsek model where we incorporated a rotational matrix, denoted as $R_n^{(t)}$. The agent-wise temporal rotation matrix $R_n^{(t)}$ imposed on the $n$th agent at the $t$th time step allows us to simulate interesting collective motion scenarios while ensuring the intra-group interactions between the agents. Based on the average direction of motion of all the particles in a neighborhood, denoted by $N_n^{(t)}$, within a radius $r_d$ of the $n$th particle, the generalized Vicsek model updates the orientation of the $n$th particle at the $t$th time step, denoted by $\theta_n^{(t)} \in [-\pi, \pi)$. Therefore, the orientation of the $n$th agent at the $t$th time step, defined as $\theta_n^{(t+1)}$, is computed as

$$\theta_n^{(t+1)} = \arg\left(\boldsymbol{u}_n^{(t)}\right) + \epsilon_n^{(t)}, \tag{17}$$

where $\epsilon_n^{(t)}$ is a noise parameter imposed to the orientation of the $n$th agent at the $t$th time step, and

$$\boldsymbol{u}_n^{(t)} = \frac{1}{\left|N_n^{(t)}\right|} \sum_{j \in N_n^{(t)}} R_j^{(t)} \begin{bmatrix} \cos\left(\theta_j^{(t)}\right) \\ \sin\left(\theta_j^{(t)}\right) \end{bmatrix}. \tag{18}$$

The position vector of the $n$th agent at the $t$th time step is given as

$$\boldsymbol{y}_n^{(t+1)} = \boldsymbol{y}_n^{(t)} + v_n^{(t)} R_n^{(t)} \begin{bmatrix} \cos\left(\theta_n^{(t)}\right) \\ \sin\left(\theta_n^{(t)}\right) \end{bmatrix} \delta, \tag{19}$$

where $v_n^{(t)}$ denotes the speed of the $n$th agent at the $t$th time step and $\delta$ is the step size. Equations (17)–(19) with $R_n^{(t)} = [1, 0; 0, 1]$ for all $n$'s and $t$'s imply the classic Vicsek model.

We use this rotational matrix to formulate a spiral collective motion scenario since spiral collective motion is often observed in nature ranging from biology, such as bacteria colonies,[42] to astronomy, such as spiral galaxies.[43] Here, first, we formulate an anticlockwise Archimedean spiral that rotates at an angle of $3\pi$ and then compute the agent-wise temporal rotational matrix based on that spiral. We assume that the two-dimensional coordinates $\left[\boldsymbol{c}^{(1)};\ldots;\boldsymbol{c}^{(t)};\ldots;\boldsymbol{c}^{(T)}\right]$ where $\boldsymbol{c}^{(t)} \in \mathbb{R}^2$ represent this spiral such that

$$\boldsymbol{c}^{(t)} = r^{(t)} \begin{pmatrix} \cos\left(\kappa^{(t)}\right) \\ \sin\left(\kappa^{(t)}\right) \end{pmatrix}, \text{ where}$$
$$r^{(t)} = 1 + \frac{3}{T-1}(t-1) \text{ and } \kappa^{(t)} = \frac{3\pi}{T}(t-1) \tag{20}$$

for $t = 1,\ldots,T$. Here, $r^{(t)}$ is the variable radius of the spiral where it changes from 1 to 4 and $\kappa^{(t)}$ is the angle of rotation with respect to the origin that varies from 0 to $3\pi$. The rotation angle with respect to the $(t-1)$th coordinates, $\boldsymbol{c}^{(t-1)} = [c_1^{(t-1)}, c_2^{(t-1)}]^{Tr}$, of this spiral, denoted by $\gamma^{(t)}$, is

$$\gamma^{(t)} = \tan^{-1}\left(\frac{c_2^{(t)} - c_2^{(t-1)}}{c_1^{(t)} - c_1^{(t-1)}}\right), \; t = 2,\ldots,T. \tag{21}$$

Thus, the two-dimensional rotational matrix for the $i$th agent at the $t$th time step is given by

$$R_n^{(t)} = \begin{pmatrix} \cos\left(-\gamma^{(t)}\right) & -\sin\left(-\gamma^{(t)}\right) \\ \sin\left(-\gamma^{(t)}\right) & \cos\left(-\gamma^{(t)}\right) \end{pmatrix}. \tag{22}$$

Equations (17)–(19) with the rotational matrix in Eq. (22) provide the complete formulation of the system generating the spiral collective motion dataset.

We generate 30 agents $\{\boldsymbol{y}_n^{(t)} | n = 1,\ldots,N; t = 1,\ldots,T\}$ using Eqs. (17), (18), (19), and (22) with 201 for the time steps ($T$), a rectangular domain with periodic boundary conditions. We set two for the radius of interaction ($r_d$), 0.05 for the speed of the particles ($\boldsymbol{v}_i^{(t)}$ for all $t$ and $i$), 0.05 for the noise on the orientation ($\epsilon_i^{(t)}$ for all $t$ and $i$), and one for the time step size ($\delta$) [see Figs. 6(a)–6(c)]. Then, we impose three noise levels sampled from the Gaussian distribution $\mathcal{N}(0,\sigma^2)$, where $\sigma = 0.2$, 0.4, and 0.6, into the original dataset and make three copies of that [see Figs. 6(d)–6(f)].

Similar to the Lorenz experiment, in order to avoid both the over-fitting and the semi-convergence of RNNs, the best number of training epochs was chosen by training an RNN with a fixed, but big, number of epochs to find the epoch number that gives the minimum loss, and then training another RNN with that many epochs. Based on the noise level of the collective motion trajectories, the best number of epochs is between 6548 and 14 032. We tested both the hyperbolic tangent and ReLU as activation functions and found that the loss with the hyperbolic tangent is smaller than the one with ReLU. Thus, we set the activation function of the RNN to a hyperbolic tangent. Similar to Lorenz experiment, to find the optimal parameter values for the learning rate, the number of RNNs, and the number of hidden nodes, first, we discretize the parameter space, then, we train and test the RNN with those discretized parameter combinations, and, finally, we select the best values for the parameters that give the least reconstruction error. Thus, the best learning rate is 0.0005, the best number of RNNs is two, and the best number of nodes is 64. Moreover, the other parameters of the RNN are set as follows: a batch to one trajectory, sequential length to 201 since each trajectory has 201 time steps, input size to two since each trajectory is two-dimensional, and output size to two since the output is also two-dimensional.

For each noise level, we train one RNN with 80%, i.e., 24, of the trajectories, and test with the rest of them. We input each trajectory $\boldsymbol{x}_n$ into the RNN one at a time and generate the corresponding output of RNN, denoted as $\hat{\boldsymbol{y}}_n$. We compute the reconstruction error between $\hat{\boldsymbol{y}}_n$ and the label $\boldsymbol{y}_n$ using Eq. (16). We use BTT to compute the derivatives of Eq. (6) with respect to the weights and update the weights using ADAM as explained in Sec. II A 2. The remaining 20%, i.e., six, of the noisy trajectories are fed into the corresponding trained RNN one trajectory at a time and obtain the reconstructions, $\hat{\boldsymbol{y}}_n$ [see Figs. 6(g)–6(i)]. Root mean square errors of the reconstructions, computed using Eq. (16), are $0.05, 0.08, 0.10$ for the noise levels $\sigma = 0.2$, 0.4, and 0.6 of $\mathcal{N}(0,\sigma^2)$, respectively. This test justifies that the RNN's trajectory reconstruction performance decreases as the noise level increases.

Similar to the erroneous Lorenz system, we use LMC as the baseline for the comparison of the reconstruction performance of RNNs. For the lower bound matrix $\mathcal{B}_l = \left[\boldsymbol{b}_1^l|\cdots|\boldsymbol{b}_n^l|\cdots|\boldsymbol{b}_{(N_1+N_2)}^l\right]$ and the upper bound matrix $\mathcal{B}_u = \left[\boldsymbol{b}_1^u|\cdots|\boldsymbol{b}_n^u|\cdots|\boldsymbol{b}_{(N_1+N_2)}^u\right]$ in Eq. (11), we set the lower bound $\boldsymbol{b}_n^l$ and the upper bound $\boldsymbol{b}_n^u$ as $\boldsymbol{d}_n - \zeta$ and $\boldsymbol{d}_n + \zeta$, respectively, if the trajectory $\boldsymbol{d}_n$ of $\mathcal{D} = \left[\boldsymbol{d}_1|\cdots|\boldsymbol{d}_n|\cdots|\boldsymbol{d}_{(N_1+N_2)}\right]$ is a noisy trajectory. Note that $\zeta \in \mathbb{R}^+$ is a parameter that guarantees the true trajectory is between $\boldsymbol{d}_n - \zeta$ and $\boldsymbol{d}_n + \zeta$. Conversely, we set $\boldsymbol{b}_n^l = \boldsymbol{b}_n^u$ if $\boldsymbol{d}_n$ is noise-free trajectory. Here, our goal is to recover the noise-free representations to the noisy trajectories; thus, this problem is considered to be a low-rank matrix completion problem as collective motion trajectories are low-rank, which we impose by setting $\lambda = 0$ in Eq. (11). For each noise level $\sigma = 0.2$, 0.4, and 0.6 of $\mathcal{N}(0,\sigma^2)$, we implement this LMC technique a few times with different $\zeta$'s until we find the best $\zeta$ giving the best reconstruction. The columns in $L$ corresponding to the noisy trajectories in $\mathcal{D}$ are the low-rank reconstructions. Root mean square errors of the reconstructions, computed using Eq. (16), are $0.05, 0.10, 0.17$ for the noise levels $\sigma = 0.2$, 0.4, and 0.6 of $\mathcal{N}(0,\sigma^2)$, respectively. Figures 6(j)–6(l) show LMC's recovered trajectories for the noise levels $\sigma = 0.2$, 0.4, and 0.6. The results evidence that the LMC's performance decreases than RNN's performance as the noise level increases.

### C. Spiky time series of rainfall-runoff

We use RNNs to forecast on hydrology data, especially rainfall-runoff time series that tend to be spiky. Then, compare the RNN's results with another forecast produced by a widely used hydrological model named as GR4J available in Ref. 36. Hydrology data, especially rainfall-runoff, is highly volatile with frequent spikes so the hydrologists have been struggling over the past decades to improve modeling of them.[44] GR4J is a daily lumped rainfall-runoff model proposed to understand hydrologic catchments' behaviors.[36] This model is characterized by two independent variables, namely, precipitation (denoted by $p$) and potential evapotranspiration (denoted by PET), a dependent variable, namely, streamflow (denoted by $q$), and four parameters, namely, the capacity of the production store (denoted by $\nu_1$), ground exchange coefficient (denoted by $\nu_2$), the capacity of the nonlinear routing store (denoted by $\nu_3$), and unit hydrograph time base (denoted by $\nu_4$). The GR4J model was used in combination with the degree-day snowmelt module available in

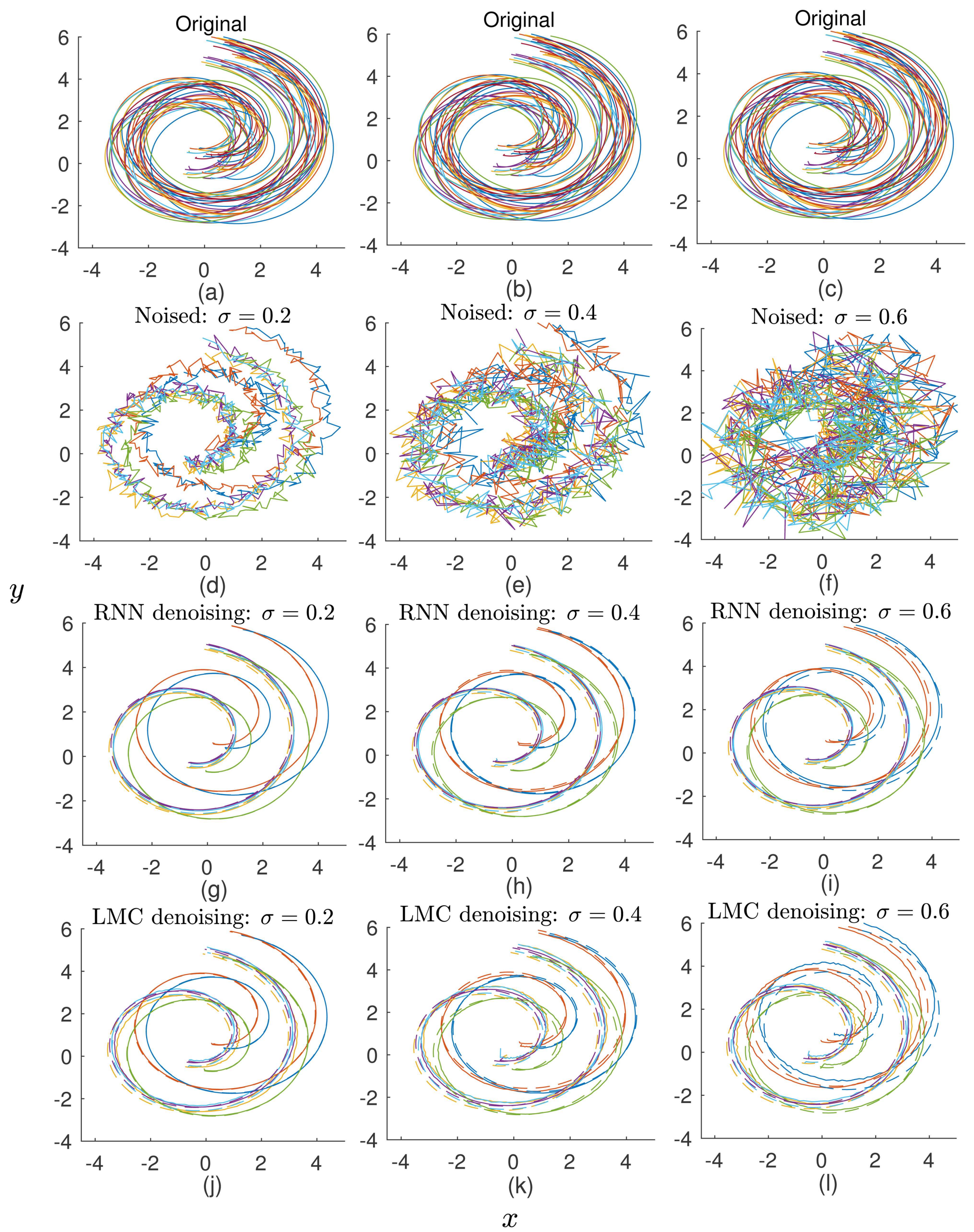

**FIG. 6.** Reconstruction of noisy collective motion trajectories using RNNs. (a)–(c) An original trajectory dataset of 30 agents is generated using a generalized version of the Vicsek model (all three datasets in a–c are identical). We make three copies of the original dataset by corrupting it with three levels of Gaussian noises with $\sigma = 0.2$, 0.4, and 0.6 of $\mathcal{N}(0, \sigma^2)$ where (d)–(f) show 20% of the noisy trajectories chosen for the testing. We pass the noisy test trajectories in each (d–f) through the corresponding RNN and denoise them as shown in (g)–(i) where the denoised trajectories are shown in continuous lines while the corresponding original trajectories are shown in dashed lines with the same colors. Similarly, the noisy trajectories are denoised using LMC as shown in (j)–(l) where the denoised and noisy trajectories are shown in the same color continuous and dashed lines, respectively. Root mean square errors of the reconstructions are $(RNN, LMC) = (0.05, 0.05)$, $(0.08, 0.10)$, and $(0.10, 0.17)$ for the noise levels $\sigma = 0.2$, 0.4, and 0.6 of $\mathcal{N}(0, \sigma^2)$, respectively.

Ref. 45, to account for snow in northern latitudes. This degree-day snowmelt module has four parameters, namely, threshold temperature (denoted by TT), degree-day factor (denoted by CFMAX), refreezing factor (denoted by CFR), and water holding capacity of the snowpack (denoted by CWH). GR4J requires tuned values for eight parameters in order to produce accurate forecasts. We use ten years of recent precipitation, evapotranspiration, and streamflow data from three arbitrarily chosen U.S. catchments, which we denote as Site 1 (MOPEX ID is 2365500), Site 2 (MOPEX ID is 2492000), and Site 3 (MOPEX ID is 7378000), of the model parameter estimation Experiment (MOPEX) database.[46]

From the ten recent years, we use the first seven years, i.e., day 1–day 2557, for the training and the last three years, i.e., day 2558–day 3653, for the forecasting. For all three hydrological sites, we follow the same training and forecasting routings and use the same parameter values in RNNs. Similar to the previous experiments, we utilize hyperbolic tangent as the activation function and 0.001 as the learning rate. We use three-stacked RNNs with 512 nodes at each hidden layer. We utilize a special training and forecasting approach here using sliding windows as it aids in sampling of adequate amount of training data just from the given time series. We make overlapping windows of the data such that each window is $L$ days long for some $L < 2557$ where two consecutive windows are misaligned by only one day (see Fig. 7 for a special case of $L = 5$). Since we vary window size in this experiment such that $L = [5, 15, 30, 45, 60, 10, 240, 365]$, we train one RNN for each case of $L$. Thus, the sequential length parameter of the RNNs is set to the corresponding value for $L$ one at a time. Since RNN's input is two one-dimensional vectors and the output is a single one-dimensional vector, the input size parameter is set to two and the output size parameter is set to one. Similar to the previous experiments, in order to avoid both the over-fitting and the semi-convergence of the RNN, the best number of training epochs was chosen by running the RNN with a fixed, but big, number of epochs to find the epoch number that gives the minimum loss, and then re-running the RNN with that many epochs to get the results. Based on $L$, the best number of epochs is between 99 178 and 99 994. We set 0.001 for the learning rate, 99 200 for the number of epochs, tanh for the activation function.

We denote ten years of MOPEX data by $\{\boldsymbol{x}_t = [p_t, PET_t, q_t]^\top \mid t = 1, \ldots, 3653\}$, where $p_t$, $PET_t$, and $q_t$ denote precipitation, evapotranspiration, and streamflows, respectively, for the $t$th day. In this study, $[p_1, \ldots, p_{2557}]$ and $[PET_1, \ldots, PET_{2557}]$ are used as the inputs to the RNN and $[q_1, \ldots, q_{2557}]$ is used as the labels during the training process. Specifically, for the first iteration, we train the RNN with $X_1 = [p_1, \ldots, p_L; PET_1, \ldots, PET_L]_{2\times L}$ as the input and $Y_1 = [q_1, \ldots, q_L]_{1\times L}$ as the labels. Similarly, for all $t \in [2, 2558 - L]$, we train the same RNN with all such $X_t$'s,

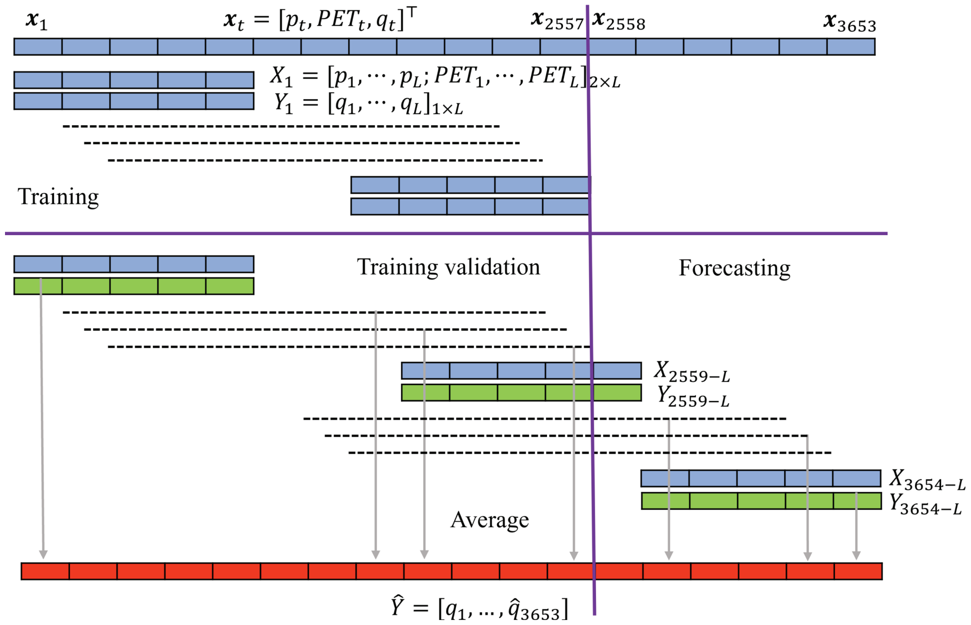
$\boldsymbol{x}_1$
$\boldsymbol{x}_t = [p_t, PET_t, q_t]^\top$
$\boldsymbol{x}_{2557}$
$\boldsymbol{x}_{2558}$
$\boldsymbol{x}_{3653}$
$X_1 = [p_1, \cdots, p_L; PET_1, \cdots, PET_L]_{2\times L}$
$Y_1 = [q_1, \cdots, q_L]_{1\times L}$
Training
Training validation
Forecasting
$X_{2559-L}$
$Y_{2559-L}$
$X_{3654-L}$
$Y_{3654-L}$
Average
$\hat{Y} = [q_1, \ldots, \hat{q}_{3653}]$

**FIG. 7.** Using a sliding window approach to train and forecast on RNNs with ten recent years of MOPEX data. Let the data be denoted as $\{\boldsymbol{x}_t = [p_t, PET_t, q_t]^\top \mid t = 1, \ldots, 3653\}$ where $p_t$, $PET_t$, and $q_t$ are precipitation, potential evapotranspiration, and streamflow, respectively, for the $t$th day. We train an RNN such that the first seven years of data, i.e., $\boldsymbol{x}_1 - \boldsymbol{x}_{2557}$ (2557 days), is used for the training so that the RNN makes forecasting for the last three years, i.e., $\boldsymbol{x}_{2558} - \boldsymbol{x}_{3653}$ (1096 days). We make overlapping windows of the data such that each window is $L$ time steps long for some $L < 2557$ where this figure shows a special case of $L = 5$. Here, $p$'s and $PET$'s are the inputs to the RNN, and $q$'s are the labels during the training process (see the blue color-bar pairs). For $t \in [1, 3654 - L]$, we pass each window $X_t$ through the trained RNN and model streamflow windows $Y_t$ (see the blue-green color-bar pairs). For each $t \in [1, 3653]$, the daily modeled streamflow is computed as the average of the streamflows between the overlapping windows corresponding to the $t$th day (see the red color-bar). The modeled streamflow for $t \in [1, 2557]$ is used to validate the training performance of the RNN and the modeled streamflow for $t \in [2558, 3653]$ is considered the streamflow forecast.

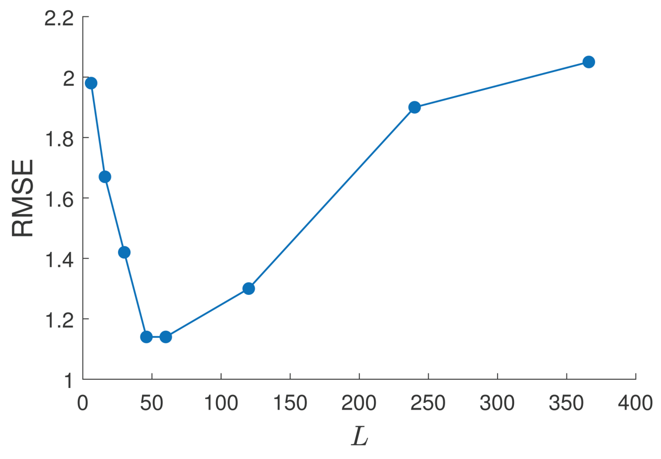

**FIG. 8.** RMSE of Site 1 (MOPEX ID is 2365500) for varying window sizes ($L$) where the RMSE is computed between the true streamflow and RNN's forecasted streamflow. We observe that the least RMSE is attained around $L = 50$.

where $X_t = [p_t, \dots, p_{t+L-1}; PET_t, \dots, PET_{t+L-1}]$, and $Y_t$'s, where $Y_t = [q_t, \dots, q_{t+L-1}]$ (see the blue color-bar pairs in Fig. 7). Now, we use this trained RNN to model the streamflow corresponding to precipitation and evapotranspiration during both the training period and the forecasting period where the modeled streamflow on the training period is used to validate the training performance. For that, we pass each window $X_t$, where $t \in [1, 3654 - L]$, through the RNN and model the streamflow windows $Y_t$ (see the blue-green color-bar pairs in Fig. 7). For each $t \in [1, 3653]$, modeled daily streamflow is computed as the average of the streamflows between the overlapping windows corresponding to the $t$th day. The modeled streamflow for $t \in [1, 2557]$ is used to validate the training performance of the RNN and the modeled streamflow for $t \in [2558, 3653]$ is considered as the streamflow forecast (see the red color vectors in Fig. 7).

We compute RMSEs between the true and the forecasted streamflows using Eq. (16) for each widow size $L$. Figure 8 shows RMSEs vs window size for Site 1 where we observe the least RMSE at $L = 45$. The orange color plots of Figs. 9(a), 9(c), and 9(e) show the true streamflows and the red color plots of those figures show the modeled streamflow by RNNs. The RMSEs, computed using

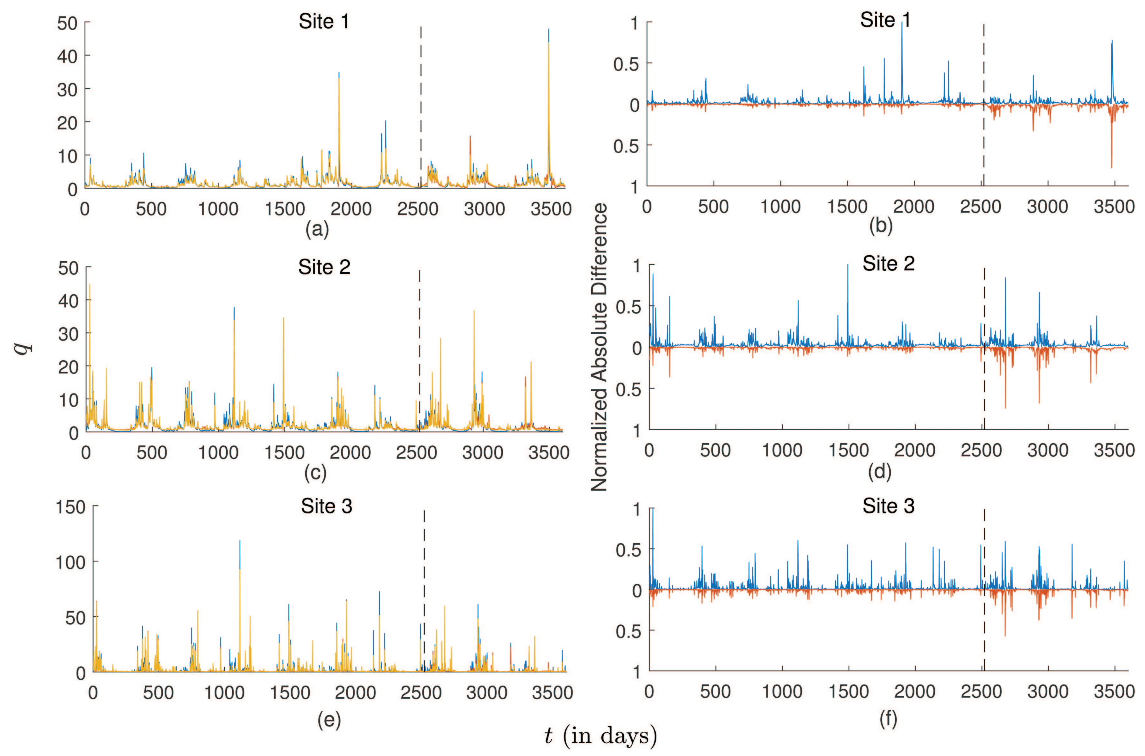

**FIG. 9.** Comparison of streamflow ($q = [q_t | \forall t]$) forecasting performance between RNN and GR4J for three hydrological sites, named as Site 1 (MOPEX ID is 2365500), Site 2 (MOPEX ID is 2492000), and Site 3 (MOPEX ID is 7378000). We use ten recent years of data for the study where the first seven years, i.e., day 1–day 2557, are used for training and the last three years, i.e., day 2558–day 3653, are used for forecasting. Orange color plots in (a), (c), and (e) show the true streamflows, and the red color plots therein (red color is not much visible as it is underneath of the orange color) show the streamflows modeled by RNN where the vertical black color line separates training and forecasting periods. The blue color plots in (a), (c), and (e) show the streamflows modeled by GR4J. For a better visual comparison between the modeled streamflows, we compute the normalized absolute difference between the true and the modeled streamflows that we show by red color plots in (b), (d), and (f) for RNN, and by blue color plots in (b), (d), and (f) for GR4J. The RMSE between forecasted and observed streamflows of Sites 1, 2, and 3 are 0.9, 1.1, and 2.0 for RNNs, respectively, and are 1.0, 1.3, and 2.7 for GR4J, respectively.

**TABLE II.** Parameter values used in the RNN for each datasets.

| Dataset | Sequential length | Input size | Output size | Learning rate | No. of RNNs | No. of hidden nodes | No. of epochs | Activation function |
|---|---|---|---|---|---|---|---|---|
| Lorenz system with a formulation error | 5000 | 3 | 3 | 0.01 | 3 | 128 | 179–930 | ReLU |
| Noise trajectories of collective motion | 201 | 2 | 2 | 0.0005 | 2 | 64 | 6548–14 032 | tanh |
| Spiky time series of rainfall-runoff | 5–365 | 2 | 1 | 0.001 | 4 | 512 | 99 178–99 994 | tanh |

Eq. (16), between the true and forecasted streamflows are 0.9, 1.1, and 2.0 for Sites 1, 2, and 3, respectively.

The eight parameters of GR4J have the recommended ranges as follows: $\nu_1 \in [0, 1000]$, $\nu_2 \in [-5, 5]$, $\nu_3 \in [0, 300]$, $\nu_4 \in [0.5, 5]$, TT $\in [-3, 3]$ , CFMAX $\in [0, 20]$, CFR $\in [0, 1]$, and CWH $\in [0.0.8]$.[36,44,45] We uniformly discretize GR4J's parameter space of the eight parameters into 50 000 points, where each point represents a combination of parameter values. We run GR4J with all of the parameter combinations for first seven years of the data where the best parameter combination with respect to RMSE is $\nu_1 = 571$, $\nu_2 = -0.03$, $\nu_3 = 48$, $\nu_4 = 2$, TT $= -0.20$, CFMAX $= 4.41$, CFR $= 0.36$, and CWH $= 0.23$, for the Site 1. We set the same values for the parameters of Site 2 and Site 3 since their optimum parameter values are either the same or sufficiently close to those of Site 1. We calibrate the GR4J model with the data for each site and model the streamflow $q_t$ for $t \in [1, 3653]$ that we show by blue color plots in Figs. 9(a), 9(b), and 9(c). The RMSEs between the true and the forecasted streamflows are 1.0, 1.3, and 2.7 for Sites 1, 2, and 3, respectively. We observe that the RMSEs associated with the RNN's modeled streamflows are less than that of the GR4J's modeled streamflows. GR4J is a widely used streamflow modeling scheme;[47] however, the performance of RNN in forecasting of streamflow is capable of exceeding the performance of GR4J.

For a better visual comparison between the streamflows modeled by RNN and GR4J, we carry out a normalization procedure. For that, first, for each of the three sites and each of the two methods RNN and GR4J, we compute the absolute difference between the true and the modeled streamflows. Then, for each site, we compute the overall maximum of the streamflow modeled by RNN, denoted as $\boldsymbol{q}^{RNN}$, and the streamflow modeled by GR4J, denoted as $\boldsymbol{q}^{GR4J}$, i.e., $max\{q_t^{all}\ \forall t \mid \boldsymbol{q}^{all} = [\boldsymbol{q}^{RNN}, \boldsymbol{q}^{GR4J}]\}$. Finally, for each site, we divide each separate streamflow vector by the site's overall maximum which we show in Figs. 9(b), 9(d), and 9(f). We see in this plot that both RNN and GR4J encounter high errors during the same time steps; however, the performance of RNN is better than that of GR4J.

Table II summarizes the parameter values used in RNNs for each of the datasets.

## IV. CONCLUSION

In this paper, we have utilized RNNs for three diverse tasks, namely, correction of a formulation error, reconstruction of corrupted particle trajectories, and forecasting of streamflow data, in three diverse fields of dynamical systems, namely, ODEs, collective motion, and hydrological modeling, respectively. The traditional approaches to solving spatiotemporal dynamical systems may not capture non-linear and complex relationships in the data since they are mostly either linear or non-linear model-based. Such spatiotemporal nonlinear systems can effectively be learned by nonlinear and model-free machine learning techniques such as RNNs as RNNs possess internal memory that enhances the learning ability.

We used the ODE model Lorenz system to produce two sets of data such that one consists of the solutions for a correct model whereas the other consists of the solution for an erroneous Lorenz model. We observed that, when the erroneous solution was provided, the trained RNN was capable of producing the corresponding correct solution better than that of a state-of-the-art LMC technique available in Ref. 21. This approach can be used for eliminating errors associated with ODE solvers even when only an imprecise closed-form solution is available. Future work in this context is to use a further advanced neural network tool such as LSTM presented in Ref. 7 as it is composed of three gates that regulate a better information flow through the unit. Such an approach enhances the learning process of spatiotemporal data so that it better aid in eliminating the formulation error even for longer corrupted orbits.

We used a generalized version of the classic Vicsek model to generate trajectories imitating a spiral collective motion scenario. We diagnosed the influence of noise contamination on RNN's denoising performance. Such noise contamination in collective motion trajectories can occur due to the causes like less precision in video cameras and less robustness of the tracking methods.[48] The results evidenced that RNNs are capable of denoising noisy trajectories while preserving the pattern of the underlying collective motion. In particular, RNNs performed better than LMC, which has shown evidence of trajectory reconstruction in the literature.[18,20] Thus, such ANN-based denoising methods can be integrated into multi-object tracking methods to generate smooth trajectories. As future work, we are planning to use RNN's denoised trajectories for the subsequent collective motion analysis tasks, such as phase transition detection as we presented in Ref. 49.

Streamflow modeling is a sophisticated task in the field of hydrology due to its high volatility and frequent spikes. GR4J is a popular tool for streamflow modeling since it has shown trustworthy streamflow modeling performance. However, we use RNNs to model streamflow with a sliding window approach. The sliding window approach is capable of generating a big amount of homogeneous data from a given time series. Moreover, the sliding window approach can be used to find the effective sequential length of the data sample that captures most of the features of the given time series. Thus, we trained an RNN with sliding windows of the empirically tested best length and then model streamflow for the entire time duration. We observed that the RNN's training process involves less reconstruction error than that of GR4J. The reason for that is RNN is model-free whereas GR4J is model-based, so RNN has more tendency to adjust to any natural system than that of GR4J. The observation that the streamflow forecast of RNN is better than that of GR4J justifies the applicability of RNN for real-life time series forecasting. We are planning to model streamflow using other well-known models such as HBV[50] and Simhyd[51] and then compare those results with that of RNNs.

Reconstruction or forecasting of high-dimensional time series data is considered to be a challenging task since learning the patterns underlying the high-dimensional data is rigorous. In this study, while the spiky rainfall-runoff time series and Lorenz trajectories were three-dimensional each, the collective motion time series was 48-dimensional. Increasing dimensionality of the time series reduces the learning performance of RNNs and then it reduces the reconstruction/forecasting performance. Manifold learning is a linear algebraic approach for dimensionality reduction that has shown promising results in machine learning and computer vision.[52] Thus, as future work, we propose a manifold learning based dimensionality reduction step in advance to the implementation of the RNN to reduce the dimensionality of the data. Then, the time series of reduced dimensions is utilized in the RNN for the reconstruction/forecast process. We, the authors, have developed diverse manifold learning based nonlinear dimensionality reduction tools (see Refs. 40, 52, and 53), which we plan to prepend onto this RNN framework when the data are high-dimensional. The proposed hybrid framework will be capable of reconstructing/forecasting high-dimensional time series data with high fidelity.

RNNs are capable of learning spatiotemporal characteristics of data derived from dynamical systems that we used for three such applications representing three fields. The observations based on our analysis justify that an RNN is an effective machine learning tool in learning dynamical systems. The internal memory module in RNNs grants this ability, which is not available in most of the spatiotemporal dynamic analyzing tools.

## ACKNOWLEDGMENTS

K. G. would like to thank the Google Cloud Platform for granting him research credit to access its GPU computing resources under Project No. 397744870419. E. M. Bollt has received funding from the Army Research Office (No. N68164-EG), the National Institutes of Health (NIH-CRCNS), Defense Advanced Research Projects Agency, and also the Office of Naval Research.

## AUTHOR DECLARATIONS

### Conflict of Interest

The authors have no conflicts to disclose.

### Author Contributions

**K. Gajamannage:** Conceptualization (equal); Data curation (equal); Formal analysis (equal); Funding acquisition (lead); Investigation (lead); Methodology (equal); Project administration (lead); Resources (lead); Software (equal); Supervision (equal); Validation (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **D. I. Jayathilake:** Data curation (equal); Formal analysis (equal); Methodology (equal); Validation (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **Y. Park:** Data curation (equal); Formal analysis (equal); Validation (equal); Visualization (equal); Writing – original draft (supporting). **E. M. Bollt:** Conceptualization (equal); Supervision (equal); Writing – review & editing (equal).

## DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author upon reasonable request.

## REFERENCES

[1] M. Hu and F. Li, "A new method to solve numeric solution of nonlinear dynamic system," Math. Probl. Eng. **2016**, 1485759 (2016).
[2] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," Neural Netw. **2**, 359–366 (1989).
[3] X. Wang and E. K. Blum, "Discrete-time versus continuous-time models of neural networks," J. Comput. Syst. Sci. **45**, 1–19 (1992).
[4] A. W. Jayawardena, *Environmental and Hydrological Systems Modelling* (CRC Press, 2013).
[5] E. Diaconescu, "The use of NARX neural networks to predict chaotic time series," Wseas Trans. Comput. Res. **3**, 182–191 (2008).
[6] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," Proc. Natl. Acad. Sci. U.S.A. **79**, 2554–2558 (1982).
[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Comput. **9**, 1735–1780 (1997).
[8] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," in *Advances in Neural Information Processing Systems* (Curran Associates, Inc., 2018), pp. 6571–6583.
[9] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks for action segmentation and detection," in *Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* (IEEE, 2017), pp. 1003–1012.
[10] K. Gajamannage and Y. Park, "Real-time forecasting of time series in financial markets using sequentially trained many-to-one LSTMs," 10.48550/arxiv.2205.04678 (2022).
[11] J. Lu, K. Deng, X. Zhang, G. Liu, and Y. Guan, "Neural-ODE for pharmacokinetics modeling and its advantage to alternative machine learning models in predicting new dosing regimens," iScience **24**, 102804 (2021).
[12] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," arXiv:1803.01271 (2018).
[13] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," IEEE Trans. Neural Networks Learn. Syst. **30**, 2805–2824 (2019).
[14] A. Sagheer and M. Kotb, "Unsupervised pre-training of a deep LStM-based stacked autoencoder for multivariate time series forecasting problems," Sci. Rep. **9**, 1–16 (2019).
[15] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," arXiv:1409.1259 (2014).
[16] F. M. Bianchi, E. Maiorino, M. C. Kampffmeyer, A. Rizzi, and R. Jenssen, "An overview and comparative analysis of recurrent neural networks for short term load forecasting," arXiv:1705.04378 (2017).
[17] M. Tabor, *Chaos and Integrability in Nonlinear Dynamics: An Introduction* (Wiley-Interscience, 1989).
[18] K. Gajamannage and R. Paffenroth, "Reconstruction of agents' corrupted trajectories of collective motion using low-rank matrix completion," in *Proceedings of the 2019 IEEE International Conference on Big Data* (IEEE, 2019), pp. 2826–2834.
[19] T. Vicsek, A. Czirók, E. Ben-Jacob, I. Cohen, and O. Shochet, "Novel type of phase transition in a system of self-driven particles," Phys. Rev. Lett. **75**, 1226–1229 (1995).

[20]K. Gajamannage, Y. Park, R. Paffenroth, and A. P. Jayasumana, "Reconstruction of fragmented trajectories of collective motion using Hadamard deep autoencoders," Pattern Recognit. **131**, 108891 (2022).
[21]R. Paffenroth, P. Du Toit, R. Nong, L. Scharf, A. P. Jayasumana, and V. Bandara, "Space-time signal processing for distributed pattern detection in sensor networks," IEEE J. Sel. Top. Signal Process. **7**, 38–49 (2013).
[22]A. P. Jayasumana, R. Paffenroth, G. Mahindre, S. Ramasamy, and K. Gajamannage, "Network topology mapping from partial virtual coordinates and graph geodesics," IEEE/ACM Trans. Networking **27**, 2405–2417 (2019).
[23]G. Mahindre, A. P. Jayasumana, K. Gajamannage, and R. Paffenroth, "On sampling and recovery of topology of directed social networks - a low-rank matrix completion based approach," in *Proceedings of the Conference on Local Computer Networks, LCN* (IEEE Computer Society, 2019), pp. 324–331.
[24]B. C. Csáji *et al.*, *Approximation with Artificial Neural Networks* (Faculty of Sciences, Eötvös Lorand University, 2001), Vol. 24, p. 7.
[25]S. Zhang, Y. Wu, T. Che, Z. Lin, R. Memisevic, R. R. Salakhutdinov, and Y. Bengio, "Architectural complexity measures of recurrent neural networks," Adv. Neural Inf. Process. Syst. **29**, 1822–1830 (2016).
[26]Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," IEEE Trans. Neural Networks **5**, 157–166 (1994).
[27]F. Chollet, *Deep Learning with Python* (Manning Publications Co. LLC, 2017).
[28]H. T. Siegelmann and E. D. Sontag, "Turing computability with neural nets," Appl. Math. Lett. **4**, 77–80 (1991).
[29]J. Schmidhuber, "Learning to control fast-weight memories: An alternative to dynamic recurrent networks," Neural Comput. **4**, 131–139 (1992).
[30]A. M. Schäfer and H.-G. Zimmermann, "Recurrent neural networks are universal approximators," Int. J. Neural Syst. **17**, 253–263 (2007).
[31]P. J. Werbos, "Backpropagation through time: What it does and how to do it," Proc. IEEE **78**, 1550–1560 (1990).
[32]A. Gruslys, R. Munos, I. Danihelka, M. Lanctot, and A. Graves, "Memory-efficient backpropagation through time," Adv. Neural Inf. Process. Syst. **29**, 4132–4140 (2016).
[33]L. Manneschi and E. Vasilaki, "An alternative to backpropagation through time," Nat. Mach. Intell. **2**, 155–156 (2020).
[34]D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015—Conference Track Proceedings* (San Diego, CA, 2015), a preprint is available on https://arxiv.org/abs/1412.6980.
[35]S. Boyd, "Distributed optimization and statistical learning via the alternating direction method of multipliers," Found. Trends Mach. Learn. **3**, 1–122 (2010).
[36]J. Vaze, P. Jordan, R. Beecham, A. Frost, and G. Summerell, *Guidelines for Rainfall-Runoff Modelling Towards Best Practice Model Application*, Interim Version 1 (eWater Cooperative Research Center, 2011).
[37]A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," Adv. Neural Inf. Process. Syst. **32**, 8026–8037 (2019).
[38]R. L. Devaney, *A First Course in Chaotic Dynamical Systems: Theory and Experiment* (CRC Press, 2018).
[39]E. N. Lorenz, "Deterministic nonperiodic flow," J. Atmos. Sci. **20**, 130–141 (1963).
[40]K. Gajamannage, S. Butail, M. Porfiri, and E. M. Bollt, "Dimensionality reduction of collective motion by principal manifolds," Physica D **291**, 62–73 (2015).
[41]K. Gajamannage, S. Butail, M. Porfiri, and E. M. Bollt, "Identifying manifolds underlying group motion in Vicsek agents," Eur. Phys. J.: Spec. Top. **224**, 3245–3256 (2015).
[42]R. Koizumi, T. Turiv, M. M. Genkin, R. J. Lastowski, H. Yu, I. Chaganava, Q.-H. Wei, I. S. Aranson, and O. D. Lavrentovich, "Control of microswimmers by spiral nematic vortices: Transition from individual to collective motion and contraction, expansion, and stable circulation of bacterial swirls," Phys. Rev. Res. **2**, 33060 (2020).
[43]S. L. Parnovsky and A. S. Parnowski, "Large-scale collective motion of RFGC galaxies," Astrophys. Space Sci. **325**, 163–175 (2010).
[44]D. I. Jayathilake and T. Smith, "Assessing the impact of PET estimation methods on hydrologic model performance," Hydrol. Res. **52**, 373–388 (2021).
[45]J. B. Kollat, P. M. Reed, and T. Wagener, "When are multiobjective calibration trade-offs in hydrologic models meaningful?," Water Resour. Res. **48**, https://doi.org/10.1029/2011WR011534 W03520 (2012).
[46]Q. Duan, J. Schaake, V. Andréassian, S. Franks, G. Goteti, H. V. Gupta, Y. M. Gusev, F. Habets, A. Hall, L. Hay, T. Hogue, M. Huang, G. Leavesley, X. Liang, O. N. Nasonova, J. Noilhan, L. Oudin, S. Sorooshian, T. Wagener, and E. F. Wood, "Model parameter estimation experiment (MOPEX): An overview of science strategy and major results from the second and third workshops," J. Hydrol. **320**, 3–17 (2006).
[47]D. I. Jayathilake and T. Smith, "Understanding the role of hydrologic model structures on evapotranspiration-driven sensitivity," Hydrol. Sci. J. **65**, 1474–1489 (2020).
[48]K. Gajamannage, R. Paffenroth, and A. P. Jayasumana, "A patch-based image denoising method using eigenvectors of the geodesics' Gramian matrix," arXiv:2010.07769 (2020).
[49]K. Gajamannage and E. M. Bollt, "Detecting phase transitions in collective behavior using manifold's curvature," Math. Biosci. Eng. **14**, 437–453 (2017).
[50]S. Bergström *et al.*, "The HBV model," in *Computer Models of Watershed Hydrology*, edited by V.P. Singh (Water Resources Publications, Highlands Ranch, CO, 1995), pp. 443–476.
[51]F. H. S. Chiew, M. C. Peel, A. W. Western *et al.*, "Application and testing of the simple rainfall-runoff model SIMHYD," *I Mathematical Models of Small Watershed Hydrology and Applications*, Water Resources Publications, Highlands Ranch, CO, 2002), pp. 335–367.
[52]K. Gajamannage, R. Paffenroth, and E. M. Bollt, "A nonlinear dimensionality reduction framework using smooth geodesics," Pattern Recognit. **87**, 226–236 (2019).
[53]K. Gajamannage and R. Paffenroth, "Bounded manifold completion," Pattern Recognit. **111**, 107661 (2021).
[54]S. Lee, M. Kooshbaghi, K. Spiliotis, C. I. Siettos, and I.G. Kevrekidis, "Coarse-scale PDEs from fine-scale observations via machine learning," Chaos **30**(1), 013141 (2020).
[55]E. Galaris, G. Fabiani, I Gallos, I. Kevrekidis, and C. Siettos, "Numerical bifurcation analysis of pdes from lattice Boltzmann model simulations: a parsimonious machine learning approach" J. Sci. Comput. **92**(2), 1–30 (2022).