Learning Monotone Dynamics by Neural Networks

Yu Wang, Qitong Gao, and Miroslav Pajic

***Abstract*— Feed-forward neural networks (FNNs) work as standard building blocks in applying artificial intelligence (AI) to the physical world. They allow learning the dynamics of unknown physical systems (e.g., biological and chemical) to predict their future behavior. However, they are likely to violate the physical constraints of those systems without proper treatment. This work focuses on imposing two important physical constraints: monotonicity (i.e., a partial order of system states is preserved over time) and stability (i.e., the system states converge over time) when using FNNs to learn physical dynamics. For monotonicity constraints, we propose to use nonnegative neural networks and batch normalization. For both monotonicity and stability constraints, we propose to learn the system dynamics and corresponding Lyapunov function simultaneously. As demonstrated by case studies, our methods can preserve the stability and monotonicity of FNNs and significantly reduce their prediction errors.**

I. Introduction

Artificial intelligence (AI) is rapidly advancing in the cyber world, especially in computer vision and natural language processing [1]. Recently, there has been a growing interest in building AI that can learn to interact with the physical world [2]. To this end, feedforward neural networks (FNNs) can serve as building blocks to learn unknown physical system dynamics [3], [4], [5] to predict their future behavior. Such systems usually obey physical constraints such as monotonicity and stability [6].

Monotonicity and stability naturally arise from applications in biology and chemistry. For instance, in an ecological model, the population of several cooperative species can be monotone. If a species' population increases at some time (e.g., by bringing in new ones from outside), then other species' populations will also be higher later, as illustrated in Figure 1. Besides, monotone systems can also be stable, i.e., the populations converge to given values over time. Examples include traffic networks [7], chemical reactions [8], and bio-ecological models [9].

However, without proper treatments, monotonicity and stability are likely to be violated by FNNs in learning, and consequently, the learned dynamics will not correctly reflect the dynamics of the real systems. In this work, we propose a new method to impose monotonicity on FNNs in learning without reducing their expressiveness by fusing nonnegative neural networks and batch normalization. In addition, for monotone and stable systems, we propose another method to impose both constraints by simultaneously learning the system dynamics and the corresponding Lyapunov function, as illustrated in Figure 2.

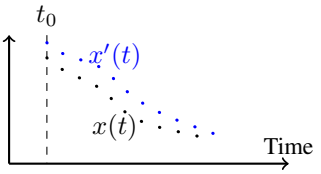

Fig. 1: Monotone system paths. If $x'(t_0) \geq x(t_0)$, then $x'(t) \geq x(t)$ for all $t \geq t_0$.

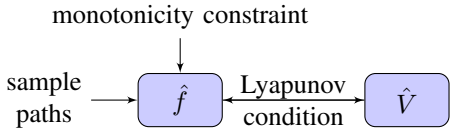

Fig. 2: Diagram of learning setup.

We implement our methods in a window-based fashion on two case studies: the Lotka-Volterra model of two cooperative species that can migrate between multiple patches and the biochemical control circuit of the translation from DNA to mRNA. The results show that our methods ensure the stability and monotonicity of the learned FNNs at most test points. In addition, by imposing the constraints and the window-based implementation, the prediction errors of the FNN to the system are significantly reduced, especially for long time horizons.

Imposing monotonicity constraints in learning starts from classification and regression of data whose labels increase (or decrease) with the features (e.g., the dependence of the value of a used car on its mileage). To handle such data, monotonicity constraints were imposed for kernel machines [10] and trees [11], [12]. Recently, monotonicity were studied for learning probabilistic dynamical models [13], [14], [15], [16]. Our work differs due to the use of the window method and learning with both monotonicity and stability constraints.

A common approach is to impose monotonicity as a penalty to the training loss, computed for a set of samples [17] or the average of some pre-defined distribution [18]. For this approach, the derived NNs are only monotonic for that set of samples or distribution, and require further certification for global monotonicity [19]. Alternatively, we can indirectly learn a monotone function from its derivative using an NN that only provides nonnegative outputs [20]. However, recovering the monotone function from the trained NN

Yu Wang is with the Department of Mechanical and Aerospace Engineering at the University of Florida, Gainesville, FL 32611, USA. Email: yuwang1@duke.edu. Qitong Gao and Miroslav Pajic are with the Department of Electrical and Computer Engineering at Duke University, Durham, NC 27708, USA. Emails: {qitong.gao, miroslav.pajic}@duke.edu

This work is sponsored in part by the ONR under agreements N00014-17-1-2504 and N00014-20-1-2745, AFOSR under award number FA9550-19-1-0169, as well as the NSF CNS-1652544 and CNS-2112562 awards.

would require integration; thus, any learning errors would accumulate over the integration/time, effectively resulting in large approximation errors.

To avoid the above issues, we impose monotonicity through the structure and weights of the NNs in a correct-by-construction way. For example, it is proposed to set single weights to be positive [21] or introduce constraints between multiple weights [22], [12], [23]. Examples range from a simple three-layer NN [24] to a more complex structure combining linear calibrators and lattices [25]. Our work proposes to use two-layer NNs with both min-ReLU and max-ReLU activation functions with nonnegative weights that can capture general nonlinear functions. To avoid sub-optimal outcomes [26] caused by the hard constraints in training, we propose to use batch normalization to "soften" the constraints. The case studies show that our approach can accurately approximate system dynamics without significantly affecting monotonicity conditions.

Inspired by the idea of using piecewise linear dynamics to approximate (known) nonlinear dynamics in non-learning context [27], we consider a NN with ReLU activation functions (instead of sigmoid). Similar to [24], [22], our NN can be viewed as a piecewise linear approximation of a nonlinear function, where the monotonicity is achieved by enforcing positive weights. However, our ReLU NN is more versatile and can have any number of layers, which is needed for learning complex nonlinear dynamics beyond classification and regression [28].

Our approach to ensuring stability is based on existing work on learning for Lyapunov functions [3], [4]. Specifically, we simultaneously learn the unknown dynamics and its Lyapunov function. This is similar in spirit to the idea from [5]. However, our update rule for training is different, as the method from [5] does not apply to window-based prediction. Specifically, instead of projecting the dynamics against the learned Lyapunov function [5] to keep them consistent with the Lyapunov condition, we propose to penalize the inconsistency between the learned dynamics and Lyapunov function in training.

II. Preliminaries

We consider an unknown discrete-time system

$$x(t+1) = f(x(t)), \tag{1}$$

where $t \in \mathbb{N}$ is the time, $x \in \mathbb{R}^n$ is the system state, and $f(\cdot)$ is a Lipschitz continuous nonlinear function. For a given initial state, we refer to the corresponding solution $x(t)$ of (1) as a *trajectory* of the system. The system (1) is a *monotone* system, if it preserves some partial order $\preceq$ on $\mathbb{R}^n$. Here we consider a common partial order [6] defined as

$$x \preceq y \iff x_i \leq y_i \text{ for all } i \in [n], \tag{2}$$

where $[n] = \{1, \ldots, n\}$, $x, y \in \mathbb{R}^n$, and $x_i, y_i \in \mathbb{R}$ denote their i-th entry.

The system (1) is *monotone* on domain $D \subseteq \mathbb{R}^n$ if for any two trajectories $x_1(t), x_2(t) \in D$, it holds that

$$x_1(0) \preceq x_2(0) \implies x_1(t) \preceq x_2(t) \text{ for all } t \in \mathbb{N}. \tag{3}$$

The system (1) is *monotone* if and only if the function f is monotonically non-decreasing in the common sense – i.e., if for any two inputs $x_1, x_2 \in \mathbb{R}^n$, it holds that $x_1 \preceq x_2 \implies f(x_1) \preceq f(x_2)$.

Example 1: A scalar linear system $x(t+1) = ax(t)$ is monotone on the domain $[0, +\infty)$ for any $a \geq 0$, since for any two initial states $x_1(0) \leq x_2(0)$, the two corresponding trajectories satisfy $x_1(t) = a^t x_1(0) \leq a^t x_2(0) = x_2(t)$. Therefore, the ordering between the initial states is preserved during the evolution of two trajectories for all times $t \in \mathbb{N}$.

It is important to highlight that the monotonicity is defined for the (initial) state not for the time, as illustrated in Figure 1. By Example 1, when $a \in (0, 1)$, the trajectory $x(t) = a^t x(0)$ decreases with the time t; yet, the system is still monotone with respect to the initial state $x(0)$. In addition, the monotonicity of the system (1) can be equivalently characterized by the gradient of function $f(x)$, as captured in the following lemma from.

Lemma 1: [6] The system (1) is monotone if and only if $\frac{\partial f}{\partial x_i} \geq 0$ for each entry $x_i, (i = 1, ..., n)$ of $x \in \mathbb{R}^n$. Specially, if the system (1) is linear – i.e., $f(x) = Ax$ for some $A \in \mathbb{R}^{n \times n}$, then it is monotone if and only if each entry $A_{ij}, (i, j \in [n])$ of the matrix A satisfies that $A_{ij} \geq 0$.

The system (1) is *globally asymptotically stable* (or *stable* for short), if it has a (discrete-time) Lyapunov function $V(x)$ [29]. Suppose that $x = 0$ is the stable point of the system (i.e., $f(0) = 0$) in general, a stable point x_0 can be moved to 0 by substituting x with $x - x_0$ in the system (1). Then $V(x)$ should satisfy the *Lyapunov condition* that

$$V(0) = 0 \text{ and } \forall x \neq 0, V(x) > 0 \text{ and } V(f(x)) - V(x) < 0. \tag{4}$$

The Lyapunov function can be viewed as a 'potential' with zero value at the stable point and positive values elsewhere. For stable systems, since the *discrete Lie derivative* $V(f(x)) - V(x)$ is negative, the (positive) value of the Lyapunov function should decrease along the system path so that it finally converges to zero at the stable point. For monotone stable systems, the Lyapunov function can always be written as the maximum of scalar functions [6]. That is, there exist scalar functions $V_i : \mathbb{R} \to \mathbb{R}$ such that

$$V(x) = \max_{i \in [n]} V_i(x_i), \quad \text{for} \quad [n] = \{1, \ldots, n\}. \tag{5}$$

This provides a foundation for our technique to efficiently learn the Lyapunov function of $\hat{f}$ by FNNs.

III. Monotone Neural Network

We introduce a window-based method to utilize FNNs to learn the dynamics of the system (1). We show that this method can reduce the learning error in general (Section III-A), as well as how to impose the monotonicity and stability constraints (Sections III-B and III-C, respectively). The proofs are available in Appendix of [30].

A. Window-Based Learning Method

Conventionally, to learn the dynamics f of the system (1), an FNN $\hat{f}_\theta$ parametrized by weights θ is trained to predict

the next state $x(t+1)$ from the current state $x(t)$ [3], [4], [5], i.e.,

$$\hat{x}(t+1) = \hat{f}_\theta(x(t)) \approx x(t+1) = f(x(t)).$$

To improve the prediction accuracy, we propose a window-based method. Specifically, the FNN $\hat{f}_\theta$ uses a q-window of past states to predict the next state, i.e.,

$$\begin{aligned}\hat{x}(t+1) &= \hat{f}_\theta(x(t), x(t-1), \dots, x(t-q+1)) \\ &\approx x(t+1) = f(x(t)).\end{aligned}$$

Accordingly, the training loss of $\hat{f}_\theta$ for f is given by

$$\mathbb{E}_{x(\cdot)\sim\rho}[\|x(t+1) - \hat{f}_\theta(x(t), x(t-1), \dots, x(t-q+1))\|^2], \tag{6}$$

where ρ is the state visitation distribution from a random initial state $x(0)$. In practice, the expectation in loss (6) is substituted by the empirical training loss for a batch of N given sample paths of the time horizon H, i.e.,

$$\begin{aligned}J(\hat{f}_\theta) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{H-q-1}\sum_{t=q-1}^{H-1}\big\|x_i(t+1) - \\ \hat{f}_\theta(x_i(t), x_i(t-1), \dots, x_i(t-q+1))\big\|^2,\end{aligned} \tag{7}$$

where N is the batch size and $x_i(\cdot)$ represents the state obtained from the i^{th} sample path for all $i \in \{1, ..., N\}$.

Admittedly by the dynamics f, the next state $x(t+1)$ only depends on the current state $x(t)$. However, using past states can still help training. The past state $x(t-i)$ is related to the next state $x(t+1)$ by $x(t+1) = f^{(i+1)}x(t-i)$. Suppose we start from training $\hat{f}_\theta$ by only using the dependency of $x(t+1)$ on $x(t)$. By adding $x(t-1)$ to training, the FNN $\hat{f}_\theta$ not only needs to fit the dependency of $x(t+1)$ on $x(t)$ but also $x(t+1)$ on $x(t-1)$. This generally reduces the prediction error when $\hat{f}_\theta$ is not exactly equal to f. By using the window method, we force the FNN $\hat{f}_\theta$ not only fit with 1-step dependencies of states by also multi-step dependencies, and hence improves the utility of sample trajectories.

B. Imposing the Monotonicity Constraints

Our method forces the input-output relation of each neuron to be monotone so that the overall FNN is monotone by setting the weights in the FNN to be nonnegative. We achieve this by resetting the negative weights to zero, or to relatively small random numbers that are close to zero, after each backpropagation operation, as in the Dropout [31] method that prevents deep neural networks from overfitting. We refer to such NNs as *nonnegative NNs*.

We use the following rectified linear unit (ReLU) activation functions

$$\begin{aligned}\varphi(x_1, \dots, x_d) &= \max\Big\{\sum_{i\in[d]}\theta_i x_i + \theta_0, 0\Big\} \\ \text{or } \min&\Big\{\sum_{i\in[d]}\theta_i x_i + \theta_0, 0\Big\}, \quad \theta_1, \dots, \theta_d \geq 0,\end{aligned} \tag{8}$$

where $x_1, ..., x_d \in \mathbb{R}$ are the inputs to the neurons, θ_0 is the bias, and $\theta_1, \dots \theta_d$ are the weights of the inputs. This is inspired by the use, in non-learning context, of piecewise linear dynamics to approximate known nonlinear dynamics [27]. Specifically, as φ is a piecewise linear function, the FNN $\hat{f}_\theta$ using such activation functions is also piecewise linear. Thus, it can serve as a piecewise linear approximation, if trained to approximate the dynamics (1). Finally, the min-ReLU activations in (8) are needed to allow for capturing general nonlinear dynamics due to the following claim.

Claim 1: If an FNN $\hat{f}_\theta$ only has the max-ReLU activations from (8), then $\hat{f}_\theta$ is *convex*.

Since the activations from (8) are monotone, the following holds.

Theorem 1: An FNN using the activations from (8) is monotone.

We note that the inverse of Theorem 1 may not be necessarily true; i.e., a monotone NN can have negative weights. For example, consider an NN with two hidden layers, each containing a single-neuron. The output of the first hidden layer is $y_1 = \max\{x_1, 1\}$ given its input x_1. The ReLU activation in the second hidden layer with a negative weight, computes $z_1 = \max\{-y_1, 1\}$ from the output of the first neuron y_1. Yet, the NN is still monotone as the output is always 1. In addition, the dynamics represented by a monotone NN may decrease over time. For example, consider the single-neuron network that computes $x_1(t+1) = \max\{0.5x_1(t), 1\}$ from the input $x_1(t)$. For an initial state $x_1(0) = 10$, the corresponding trajectory is decreasing with time $t \in \mathbb{N}$.

Batch Normalization.: Imposing hard constraints on the weights can lead to undesirable sub-optimal results in training, as observed in [26]. Hence, instead of straightly imposing the positive weight constraint $\theta_0, \dots, \theta_d \geq 0$ for the activations (8), we propose to use batch normalization (BN) [32] to soften the constraints and ensure the representation power of $\hat{f}_\theta$. This is because the BN parameters are allowed to converge to optima defined in a broader search space if necessary but can be trained to satisfy the weight constraints as well if it is optimal to do so, although this may lead to tolerable (minor) violations of the hard constraints as we will show in the applications in Section IV.

C. Imposing Stability Constraints

When the system (1) of interest is stable, we introduce the following learning method based on an optimization framework [33] that learns $\hat{f}_\theta$ and $\hat{V}_\xi$ iteratively. Recall that the system is stable if and only if it has a Lyapunov function $V(x)$ in the form of (5). Here, we train an FNN $\hat{V}_\xi(x)$ of the form (5) to represent $V(x)$. For a given $\hat{f}_\theta$, we train $\hat{V}_\xi(x)$ by imposing the Lyapunov condition (4) via the following expected loss

$$\begin{aligned}\min_\xi \mathbb{E}_{x(\cdot)\sim\rho}\Big(&\hat{V}_\xi(0)^2 + \big[-\hat{V}_\xi\big(x(t)\big)\big]^+ \\ &+ \big[\hat{V}_\xi\big(\hat{f}_\theta\big(x(t:t-q+1)\big)\big) - \hat{V}_\xi\big(x(t)\big)\big]^+\Big),\end{aligned} \tag{9}$$

where the expectation $\mathbb{E}_{x(\cdot)\sim\rho}$ follows from (6). In (9), the first term penalizes the non-zero value of $\hat{V}_\xi(0)$, the second term penalizes the negative values of $\hat{V}_\xi(x)$, and the third term

penalizes the positive values of the discrete Lie derivative of $\hat{V}_\xi$ for $\hat{f}_\theta$, as discussed in Section II. Effectively, our approach can be viewed as a discrete-time version of the training loss for the Lyapunov function from [4].

In practice, the expected loss of (9) is approximated by the average of N sample paths of length $H \gg q$ – i.e., to impose the Lyapunov condition, while training FNN $\hat{V}_\xi$, we utilize the loss function

$$\min_{\xi} \frac{1}{N}\sum_{i=1}^{N} \frac{1}{H-q-1}\sum_{t=q-1}^{H-1} \Big(\hat{V}_\xi(0)^2 + \big[-\hat{V}_\xi\big(x_i(t)\big)\big]^+ + \big[\hat{V}_\xi\big(\hat{f}_\theta\big(x_i(t:t-q+1)\big)\big) - \hat{V}_\xi\big(x_i(t)\big)\big]^+\Big). \quad (10)$$

Similarly, for a given $\hat{V}_\xi$, we train $\hat{f}_\theta$ by incorporating the Lyapunov condition (4) into the training loss of (6)

$$\min_{\theta} \mathbb{E}_{x(\cdot)\sim\rho}\Big(\big\|f(x(t)) - \hat{f}_\theta(x(t:t-q+1))\big\|^2 + \Big[\hat{V}_\xi\big(\hat{f}_\theta\big(x_i(t:t-q+1)\big)\big) - \hat{V}_\xi(x)\Big]^+\Big). \quad (11)$$

Here, the first term penalizes the difference in predicting the next state between $\hat{f}_\theta$ and f from (1); the second term penalizes the positive values of the discrete Lie derivative of $\hat{V}_\xi$ for $\hat{f}_\theta$, equivalent to the third term of (10). The first two terms of (10) are not included, as they are independent of $\hat{f}_\theta$.

As done for (10), in practice we approximate the expected loss in (11) by using the sample average

$$\min_{\theta} \frac{1}{N}\sum_{i=1}^{N} \frac{1}{H-q-1}\sum_{t=q-1}^{H-1} \Big(\big\|f(x_i(t)) - \hat{f}_\theta(x_i(t:t-q+1))\big\|^2 + \Big[\hat{V}_\xi\big(\hat{f}_\theta\big(x_i(t:t-q+1)\big)\big) - \hat{V}_\xi\big(x_i(t)\big)\Big]^+\Big). \quad (12)$$

To train $\hat{f}_\theta$ by (12), we also impose the monotonicity constraints using the method from Sec. III-B.

IV. Case Studies

To evaluate the effectiveness of our techniques, we consider two high-dimensional complex nonlinear dynamical systems that are monotone due to their physical properties.

A. Lotka-Volterra (LV) Model.

We start with the LV model that describes the interaction of two cooperative groups (e.g., the males and the females of the same specie) occupying an environment of n discrete patches. For the group $k \in \{0,1\}$, let $x_{ik}(t) \geq 0$ be the populations of the group k in the i^{th} patch at time $t \in \mathbb{N}$. The rate of migration from the j^{th} patch to the i^{th} patch is $a_{jik}x_{ik}(t)$ with $a_{jik} \geq 0$. At the patch i, the death rate is $b_{ik}x_{ik}(t)$, with $b_{ik} \geq 0$; and the reproduction rate is $c_{ik}x_{ik}(t)x_{i\bar{k}}(t)$, with $c_{ik} \geq 0$, where we make the convention that $\bar{k} = (k+1) \bmod 2$. Thus, for a discrete-time step $\tau > 0$, the change of the populations of the two groups at the i^{th} patch is given by

$$x_{ik}(t+1) = x_{ik}(t) + \tau\Big(c_{ik}x_{ik}(t)x_{i\bar{k}}(t) - b_{ik}x_{ik}(t) + \sum\nolimits_{j\in[n]\setminus\{i\}} a_{jik}\big(x_{jk}(t) - x_{ik}(t)\big)\Big). \quad (13)$$

The LV model is monotone on the positive orthant – if the population $x_{ik}(t)$ suddenly increases at time t for some i and k (e.g., adding new individuals from the outside), then the growth rate of all other populations will not decrease; thus, their populations only increase from the increment of $x_{ik}(t)$. To ensure that the time discretization in (13) faithfully captures this monotonicity property, the time step τ should satisfy $\tau < 1/\max_{k\in\{1,2\}}\max_{i\in[n]}\big(b_{ik} + \sum_{j\in[n]\setminus\{i\}} a_{jik}\big)$ for the system (13) to be monotone.

B. Biochemical Control Circuit (BCC) Model.

We also consider the BCC model describing the process of synthesizing a protein from segments of mRNA E_0 in a cell, through a chain of enzymes $E_1, ..., E_n$, where E_n is the end product. Let $x_0(t) \geq 0$ be the cellular concentration of mRNA, $x_i(t) \geq 0$ be the concentration of the enzyme i for $i \in [n]$ at time t. For each $i \in [n]$, the chemical reaction $\alpha_i E_{i-1} \to E_i$ is assumed to happen with unit rate, where $\alpha_i > 0$. In addition, the end product stimulates the creation of the mRNA by the rate $(x_n^p(t)+1)/(x_n^p(t)+K)$ for some $K > 1$, and $p \in \mathbb{N}$. For a discrete-time step $\tau > 0$, the change of the concentration of the enzyme i is given by

$$\begin{aligned} x_0(t+1) &= x_0(t) + \tau\Big(\frac{x_n^p(t)+1}{x_n^p(t)+K} - \alpha_1 x_1(t)\Big); \\ x_i(t+1) &= x_i(t) + \tau\big(x_{i-1}(t) - \alpha_i x_i(t)\big). \end{aligned} \quad (14)$$

The BCC model is monotone in the positive orthant since if the concentration of the enzyme i increases with the concentration of the enzyme $i+1$. To ensure that the time discretization in (14) does not violate this monotonicity property, the time step τ should satisfy $\tau < 1/\max_{i\in[n]}\alpha_i$ for the system (14) to satisfy Lemma 1, i.e., to be monotone.

C. Evaluation.

We set $n = 10$ in (13) for the LV model and $n = 20$ in (14) for the BCC model, so the dimensions of the system states are 20 and 21, respectively. The training data are drawn from the system with a random set of initial states $x(0)$. More details on the selection of system constants, FNN architectures, and training hyper-parameters are provided in Appendix B of [30].

We compare the performance when training the FNNs with (I) the proposed loss (12) that enforces both monotonicity and stability conditions, against (II) monotonicity loss (7) only (which does not ensure stability), and (III) mean-square loss only (i.e., neither monotonicity nor Lyapunov conditions are considered). We test the FNNs by iteratively predicting the system state T time steps (specifically $T = 1500, 2500, 3500$) after a given initial q-window of states that are not contained in the training data and then compare with the ground truth. In all cases, the FNN is trained for different windows sizes (specifically $q = 1, 100$). The normalized ℓ^2-norm errors, defined by the ratio of the ℓ^2-norm of the prediction error to the ℓ^2-norm of the ground truth, are summarized in Table I. The column headers "monotone and Lyapunov", "monotone only" and "baseline" refer to training with the methods (I), (II), and (III) specified above.

TABLE I: Normalized ℓ^2-norm of Errors in Approximated Trajectories

	Monotone & Lyapunov		Monotone Only		Baseline	
LV Model						
Total Steps\Window	100	1	100	1	100	1
1500	0.1063	**0.0514**	**0.1184**	0.5114	**0.1578**	0.5700
2500	0.1070	**0.0886**	0.1262	**0.9383**	**0.1628**	1.0302
3500	0.1070	**0.0966**	**0.1616**	1.2983	**0.1970**	1.4143
BCC Model						
Total Steps\Window	100	1	100	1	100	1
1500	**0.0359**	0.2169	**0.0397**	0.2663	**0.0376**	1.6004
2500	**0.0334**	0.3878	**0.0856**	0.4514	**0.0349**	1.9314
3500	**0.0330**	0.5543	**0.1746**	0.6290	**0.0377**	2.2409

The predicted trajectories for a subset of the states for the two case studies are shown in Figures 3(a) and 4(a), whereas the results for all states are provided in Appendix D of [30]. Imposing either the monotonicity or stability constraints reduces the prediction errors, and imposing both brings down the error even further. In addition, when the stability constraint is imposed (method (I)), the trained FNN becomes much more stable, which significantly reduces the prediction errors for long time horizons. Also, Table I shows using a longer window results in more accurate predictions of future states – the normalized ℓ^2-norm of the prediction errors for the window size $q = 100$ is generally much smaller than that for the window size $q = 1$. In addition, the prediction errors ramp up much slower for $q = 100$ than $q = 1$ over long time horizons.

To validate the monotonicity of the trained FNNs, we show in Figures 3(b) and 4(b) the $x(t + 1)$ against $x(t)$ relations for the first 250 steps for a selection of the dimensions of the states. The figures for all dimensions are given in Appendix D of [30]. The monotonicity condition is better satisfied when the monotonicity constraint is imposed in the training loss (in method (I) and (II)), despite the occasional violations due to the batch normalization. Besides, imposing the stability constraint (method (I)) generally does not worsen the violation of monotonicity.

V. Conclusion

We introduced a window-based method to learn the dynamics of unknown nonlinear monotone and stable dynamical systems. We employed feedforward neural networks (FNNs) and captured the system's physical properties by imposing the corresponding monotonicity and stability constraints during training. On two high-dimensional complex nonlinear systems (biological and chemical), we showed that the combination of the monotonicity and stability constraints enforces both properties on the learned dynamics while significantly reducing learning errors.

References

[1] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*, 2016.

[2] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford *et al.*, "The limits and potentials of deep learning for robotics," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 405–420, 2018.

[3] S. M. Richards, F. Berkenkamp, and A. Krause, "The Lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems," in *Proceedings of The 2nd Conference on Robot Learning*, 2018, pp. 466–476.

[4] Y.-C. Chang, N. Roohi, and S. Gao, "Neural Lyapunov control," in *Advances in Neural Information Processing Systems 32*, 2019, pp. 3240–3249.

[5] J. Z. Kolter and G. Manek, "Learning stable deep dynamics models," in *Advances in Neural Information Processing Systems*, 2019, pp. 11 126–11 134.

[6] H. Smith, *Monotone Dynamical Systems: An Introduction to the Theory of Competitive and Cooperative Systems*, 2008, vol. 41.

[7] S. Coogan and M. Arcak, "Efficient finite abstraction of mixed monotone systems," in *Proceedings of the 18th International Conference on Hybrid Systems Computation and Control - HSCC '15*, 2015, pp. 58–67.

[8] P. D. Leenheer, D. Angeli, and E. D. Sontag, "Monotone chemical reaction networks," *Journal of Mathematical Chemistry*, vol. 41, no. 3, pp. 295–314, 2007.

[9] R. F. Costantino, J. M. Cushing, B. Dennis, and R. A. Desharnais, "Experimentally induced transitions in the dynamic behaviour of insect populations," *Nature*, vol. 375, no. 6528, pp. 227–230, 1995.

[10] H. Mukarjee and S. Stern, "Feasible nonparametric estimation of multiargument monotone functions," *Journal of the American Statistical Association*, vol. 89, no. 425, pp. 77–80, 1994.

[11] A. Ben-David, "Monotonicity maintenance in information-theoretic machine learning algorithms," *Machine Learning*, vol. 19, no. 1, pp. 29–43, 1995.

[12] K. Neumann, M. Rolf, and J. J. Steil, "Reliable integration of continuous constraints into extreme learning machines," *International Journal of Uncertainty, Fuzziness and Knowledge Based Systems*, vol. 21, no. supp02, pp. 35–50, 2013.

[13] F. Dondelinger, D. Husmeier, S. Rogers, and M. Filippone, "ODE parameter inference using adaptive gradient matching with Gaussian processes," in *Artificial Intelligence and Statistics*, 2013, pp. 216–228.

[14] B. Calderhead, M. Girolami, and N. Lawrence, "Accelerating bayesian inference over nonlinear differential equations with Gaussian processes," *Advances in neural information processing systems*, vol. 21, pp. 217–224, 2008.

[15] J. Riihimäki and A. Vehtari, "Gaussian processes with monotonicity information," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010, pp. 645–652.

[16] M. Lorenzi and M. Filippone, "Constraining the dynamics of deep probabilistic models," *arXiv preprint arXiv:1802.05680*, 2018.

[17] A. Gupta, N. Shukla, L. Marla, A. Kolbeinsson, and K. Yellepeddi, "How to incorporate monotonicity in deep networks while preserving flexibility?" *arXiv:1909.10662 [cs]*, 2019.

[18] J. Sill and Y. S. Abu-Mostafa, "Monotonicity hints," in *Advances in Neural Information Processing Systems 9*, 1997, pp. 634–640.

[19] X. Liu, X. Han, N. Zhang, and Q. Liu, "Certified monotonic neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 15 427–15 438, 2020.

[20] A. Wehenkel and G. Louppe, "Unconstrained monotonic neural networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 1543–1553.

[21] N. P. Archer and S. Wang, "Application of the back propagation neural network algorithm with monotonicity constraints for two-group classification problems," *Decision Sciences*, vol. 24, no. 1, pp. 60–75, 1993.

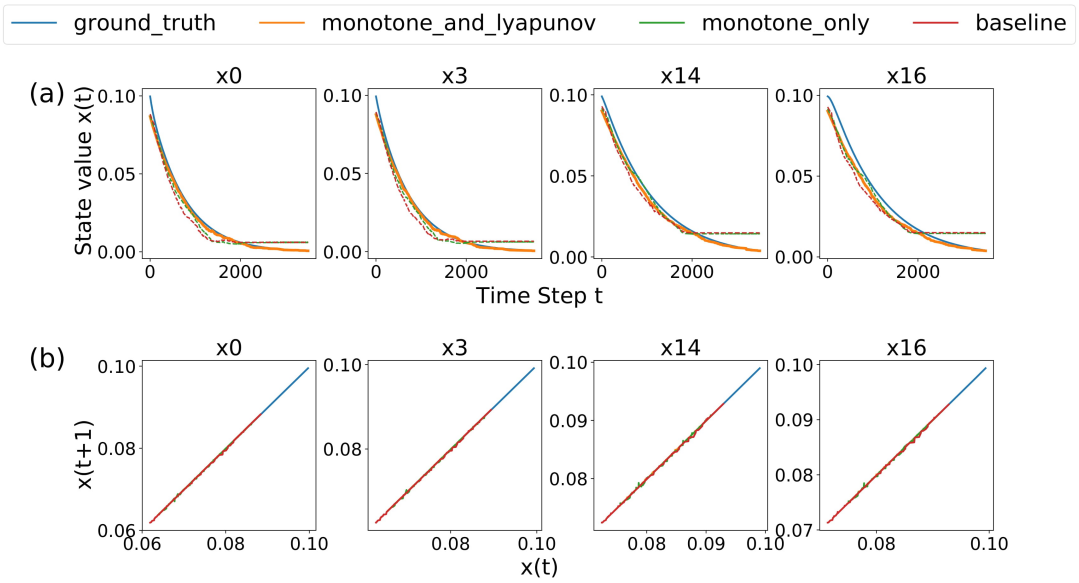

Fig. 3: (a) Predicted trajectories of the LV model using 100-window up to 3500 time steps; (b) The $x(t+1)$-$x(t)$ relation of the predicted LV model trajectory up to 250 time steps.

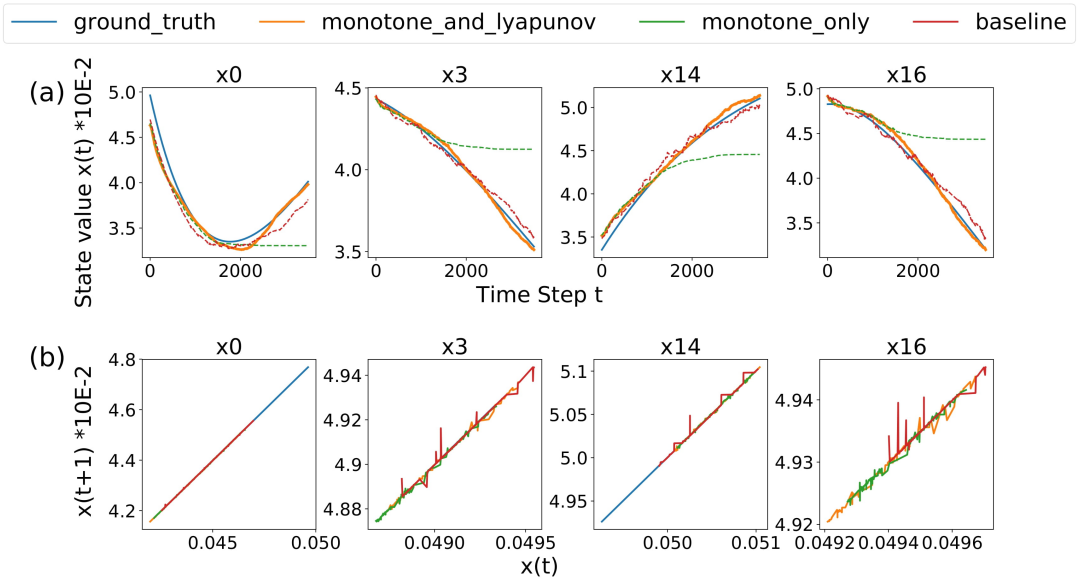

Fig. 4: (a) Predicted trajectories of the LV model using 100-window up to 3500 time steps; (b) The $x(t+1)$-$x(t)$ relation of the predicted LV model trajectory up to 250 time steps.

[22] H. Daniels and M. Velikova, "Monotone and partially monotone neural networks," *IEEE Transactions on Neural Networks*, vol. 21, no. 6, pp. 906–917, 2010.

[23] M. Gupta, A. Cotter, J. Pfeifer, K. Voevodski, K. Canini, A. Mangylov, W. Moczydlowski, and A. van Esbroeck, "Monotonic calibrated interpolated look-up tables," p. 47, 2016.

[24] J. Sill, "Monotonic networks," in *Advances in Neural Information Processing Systems*, 1998, pp. 661–667.

[25] S. You, D. Ding, K. Canini, J. Pfeifer, and M. Gupta, "Deep lattice networks and partial monotonic functions," in *Advances in Neural Information Processing Systems 30*, 2017, pp. 2981–2989.

[26] P. Márquez-Neila, M. Salzmann, and P. Fua, "Imposing hard constraints on deep networks: Promises and limitations," *arXiv preprint arXiv:1706.02025*, 2017.

[27] X. Chen, E. Ábrahám, and S. Sankaranarayanan, "Taylor model flowpipe construction for non-linear hybrid systems," in *2012 IEEE 33rd Real-Time Systems Symposium*, 2012, pp. 183–192.

[28] S. Liang and R. Srikant, "Why deep neural networks for function approximation?" in *The 5th International Conference on Learning Representations*, 2017.

[29] H. K. Khalil, *Nonlinear Systems*, 3rd ed., 2002.

[30] Y. Wang, Q. Gao, and M. Pajic, "Deep learning for stable monotone dynamical systems," Tech. Rep., 2021.

[31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[32] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[33] J. C. Bezdek and R. J. Hathaway, "Convergence of alternating optimization," *Neural, Parallel & Scientific Computations*, vol. 11, no. 4, pp. 351–368, 2003.