Nonlinear Dyn (2024) 112:3505–3518
https://doi.org/10.1007/s11071-023-09206-y

ORIGINAL PAPER

Learn from one and predict all: single trajectory learning for time delay systems

Xunbi A. Ji · **Gábor Orosz**

Received: 31 July 2023 / Accepted: 20 November 2023 / Published online: 9 January 2024

Abstract This paper focuses on learning the dynamics of time delay systems from trajectory data and proposes the use of the maximal Lyapunov exponent (MLE) as an indicator to describe the richness of the training data. Neural networks with trainable time delays are utilized to construct neural delay differential equations, and a learning algorithm based on network simulation loss is proposed to learn the delays and the nonlinearities in the right-hand side of these equations. We demonstrate that with the proposed networks and learning algorithm, the delay and the nonlinearity in the Mackey–Glass system can be learned from a single trajectory of sufficient richness. We also show that having larger MLE results in better generalizability of the trained networks. Namely, a network trained on a single chaotic trajectory obtained for a given delay value can be used to predict the dynamics for other delay values, where the system possesses stable/unstable equilibria, limit cycle oscillations, and/or chaotic behavior. We also test the learning algorithm on other systems including a climate model with multiple delays and external forcing. In this case, again, a time delay neural network can correctly learn the delays and the nonlinearities from a single chaotic trajectory, and the corresponding neural delay differential equation gives good short-term predictions.

Keywords Time delay systems · Delay learning · Neural delay differential equations

X. A. Ji (✉)
Department of Mechanical Engineering, University of Michigan, Ann Arbor, MI 48109, USA
e-mail: xunbij@umich.edu

G. Orosz
Department of Mechanical Engineering and Department of Civil and Environmental Engineering, University of Michigan, Ann Arbor, MI 48109, USA

1 Introduction

In recent years, data-driven methods have been applied to dynamical systems, and among those methods, neural networks have shown their power in function approximation [1,2]. However, they are also among the most criticized methods since they involve tremendous number of parameters, which makes their analysis difficult [3]. Moreover, generalizability of the trained network can be an issue if "big data" are not available. That is, neural networks can have poor performance outside the domain of the training data. Recent works made improvement by embedding prior knowledge—gained from first principle models—into the network design; see [4–8] for instance.

Different from the classification and regression problems, which normally aim to obtain maps between individual samples, learning dynamical systems involves sequential data referred to as trajectories. In order to learn the underlying dynamics, neural networks should consider the time dependency in the data. In discrete time, recurrent neural networks such as long

short-term memory (LSTM) networks [9] or residual neural networks (ResNet) [10], which consist of many layers, can be used to represent the information flow going through time. In continuous time, neural ordinary differential equations (NODEs) [11–13] directly model the underlying dynamics using a neural network. These networks capture the right-hand side of differential equations, that is, their input is typically the state of a dynamical system, while their output is the time derivative of the state.

Adding time delays to neural networks as additional parameters can increase their capability of approximating the dynamics while keeping the network structure simple. Neural delay differential equations (NDDEs) [14,15] were originally introduced to address the problem of intersecting trajectories, as the corresponding dynamical system has infinite-dimensional state space. It was also shown that NDDEs can be used to capture the latent dynamics of complex systems [16,17]. The theoretical background of time delay systems has a rich history [18–24]. Also, delays appear in many real-world applications including cryptography [25,26], epidemiology [27,28], ecology [29], climate models [30], machining processes [31], vehicle dynamics [32,33], and transportation systems [34,35]. Incorporating time delays into the data-driven models can help to interpret the models [36,37] and generalize for multiple scenarios without requiring additional data [38].

In this work, we utilize time delay neural networks with trainable delay to represent neural delay differential equations. We also put forward a new delay learning algorithm, which can simultaneously learn the time delay and the nonlinearity from trajectory data. We demonstrate that using such framework, complex dynamics can be learned via neural networks of simple structure. Additionally, we show that those learned delays are meaningful and they help neural networks to generalize better, even when the training data is limited. In particular, the dynamics can be learned from a single trajectory generated for one delay value, such that the network gives accurate predictions for a wide range of delay values.

Apart from showcasing successful implementation of our learning algorithm, we show that the maximal Lyapunov exponent (MLE) [39] can be utilized to quantify the richness of the data. Time delay systems often exhibit qualitative changes in behavior as the delay is varied.For instance, the system may have a stable equilibrium (MLE < 0) for small delay values and a stable limit cycle (MLE $= 0$) for larger delay values. One may study such qualitative changes using bifurcation analysis [40]. For some systems, changing the delay leads to chaotic behavior (MLE > 0) with trajectories covering a large part of the state space [41]. Figure 1 illustrates the MLE of such a system as a function of the delay τ (blue curve). At each delay value, we have a single trajectory as the training data and we learn the system dynamics from that trajectory. The MLE of learned network is shown by the orange curve. With the rich dynamics indicated by the high MLE, the trained network could also reproduce the system dynamics better in terms of delays, nonlinearities, and MLE. The key idea is to train network on rich data (with positive MLE), so that it can predict the behavior for other delay values (for which the MLE may be positive, negative or zero) without retraining.

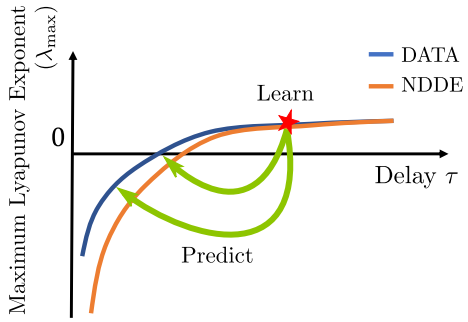

Fig. 1 Conceptual diagram on how the maximal Lyapunov exponent of the data used for learning affects the generalizability of the neural delay differential equation (NDDE)

The rest of the paper is organized as follows. In Sect. 2, we introduce time delay systems and the tools to compute Lyapunov exponents for such systems. In Sect. 3, we introduce neural delay differential equations with trainable delays, while in Sect. 4, we construct the loss function and introduce the training algorithm. In Sect. 5, we apply the algorithm to learn the dynamics of a classical example, namely the Mackey–Glass system, generate the bifurcation diagram for the learned system, and compute the maximal Lyapunov exponent to evaluate the generalizability of the network. We also show that the algorithm is applicable for a climate model which contains multiple delays as well as external forcing in Sect. 6. We summarize the results and provide future research directions in Sect. 7.

2 Lyapunov exponents for time delay systems

Lyapunov exponents represent the rate of separation of infinitesimally close trajectories. Positive maximal Lyapunov exponent (MLE) $\lambda_{\max} > 0$ is used to characterize the sensitivity to initial conditions in chaotic dynamical systems [39]. Thus, we use the MLE to characterize the richness of the dynamics. In this section, we introduce how to calculate the MLE for time delay systems. For more details about the numerical algorithms; see [42].

Consider a nonlinear autonomous time delay system whose time evolution is determined by

$$\dot{x}(t) = \mathcal{G}(x_t), \tag{1}$$

where $x \in \mathbb{R}^n$ and $x_t(\vartheta) = x(t+\vartheta)$, $\vartheta \in [-\tau_{\max}, 0]$ with maximum delay $\tau_{\max} > 0$. That is, $x_t \in \mathcal{X}$ where $\mathcal{X}$ denotes the space of continuous functions $\mathcal{X} = C([-\tau_{\max}, 0], \mathbb{R}^n)$ and $\mathcal{G} : \mathcal{X} \to \mathbb{R}^n$ is a functional. For initial history x_0, let us denote the solution as $\varphi(x_0, t)$, that is, $x_t(\vartheta) = \varphi(x_0(\vartheta), t)$. Then for initial history $x_0 + \delta x_0$, which is in the close vicinity of x_0 for small $\|\delta x_0\|$, we have the solution $\varphi(x_0 + \delta x_0, t)$. The maximal Lyapunov exponent for the time delay system (1) is defined as in [41]:

$$\lambda_{\max} = \lim_{t\to\infty} \lim_{\|\delta x_0\|\to 0} \frac{1}{t} \ln \frac{\|\varphi(x_0 + \delta x_0, t) - \varphi(x_0, t)\|}{\|\delta x_0\|}, \tag{2}$$

where $\|\delta x_0\| = \sup_{\vartheta\in[-\tau_{\max},0]} \|\delta x_0(\vartheta)\|_2$ and the condition for the existence of the limit is also given in [41].

By linearizing the autonomous nonlinear time delay system (1) about a time-dependent solution, one may obtain the linear nonautonomous system

$$\dot{x}_t = \mathcal{A}(t)x_t, \tag{3}$$

with the time-dependent linear operator $\mathcal{A} : \mathbb{R}\times\mathcal{X} \to \mathcal{X}$. Then, by discretizing time, this infinite-dimensional system can be reduced to a system of $M+1$ nonautonomous linear ordinary differential equations (ODEs):

$$\begin{bmatrix} \dot{x}_t(\vartheta_0) \\ \dot{x}_t(\vartheta_1) \\ \vdots \\ \dot{x}_t(\vartheta_M) \end{bmatrix} = \mathcal{A}_M(t) \begin{bmatrix} x_t(\vartheta_0) \\ x_t(\vartheta_1) \\ \vdots \\ x_t(\vartheta_M) \end{bmatrix}, \tag{4}$$

with mesh $-\tau_{\max} = \vartheta_M < \ldots < \vartheta_1 < \vartheta_0 = 0$. The simplest, uniform-mesh discretization uses $\vartheta_i = ih$ with distance $h = \tau_{\max}/M$ between the mesh points; see [38]. In this work, we use the Chebyshev mesh

$$\vartheta_i = \frac{\tau_{\max}}{2}\left(\cos\left(\frac{i\pi}{M}\right) - 1\right), \tag{5}$$

which provides higher accuracy [24].

The Lyapunov exponents can then be calculated using discrete QR method [42]. One can factorize the fundamental matrix $X(t)$ of equation (4), at a strictly increasing sequence of time instances $\{t_k\}$ with $t_0 = 0$:

$$X(t_k) = Q_k R_k, \tag{6}$$

where Q_k is an orthogonal matrix and R_k is an upper triangular matrix. The matrix R_k can be expressed by the product of a series upper triangular matrices $R_{j,j-1}$:

$$R_k = \left(\prod_{j=1}^{k} R_{j,j-1}\right) R_0. \tag{7}$$

The matrix $R_{j,j-1}$ is computed from factorizing the solution $\psi_{t_{j-1}}(t)$ of the system

$$\dot{\psi}_{t_{j-1}}(t) = \mathcal{A}_M(t)\psi_{t_{j-1}}(t), \quad t \in [t_{j-1}, t_j], \tag{8}$$

with initial condition $\psi_{t_{j-1}}(t_{j-1}) = Q_{j-1}$, yielding

$$\psi_{t_{j-1}}(t_j) = Q_j R_{j,j-1}. \tag{9}$$

at time $t = t_j$.

The Lyapunov exponents are then recovered as

$$\lambda_i = \limsup_{k\to\infty} \frac{1}{t_k} \sum_{j=1}^{k} \ln[R_{j,j-1}]_{i,i}, \tag{10}$$

where $[R_{j,j-1}]_{i,i}, i = 1, \ldots, M+1$ is the i-th diagonal entry of $R_{j,j-1}$ [43]. When computing (10), one should truncate the time at a large enough value $t_k = T$ and choose a sufficiently large mesh size M in (4) to obtain good approximation of λ_i for the nonlinear time delay system. With larger T and larger M, the maximal Lyapunov exponent $\lambda_{\max} = \lambda_1$ can be represented with higher accuracy; see the evaluation in [42]. In this work, the MLE $\lambda_{\max}$ is obtained with $T = 10^4$ and $M = 10$. These two parameters were chosen based on a balance between approximation accuracy and computation efficiency. Such computation of Lyapunov exponents is also applicable for time delay systems with multiple discrete delays and even distributed delays, as long as the system can be linearized around a given solution.

3 Neural delay differential equations

System (1) with d discrete delays, $0 \le \tau_1, \ldots, \tau_d \le \tau_{\max}$, can be written as

$$\dot{x}(t) = \mathbf{g}(\mathbf{x}_t), \tag{11}$$

where $\mathbf{x}_t = \begin{bmatrix} x(t-\tau_1) \\ \vdots \\ x(t-\tau_d) \end{bmatrix} \in \mathbb{R}^{nd}$ and $\mathbf{g} : \mathbb{R}^{nd} \to \mathbb{R}^n$. In order to capture the dynamics in equation (11), we construct the neural delay differential equation analogously,

$$\dot{\hat{x}}(t) = \text{net}(\hat{\mathbf{x}}_t), \tag{12}$$

with d delays, $0 \le \hat{\tau}_1, \ldots, \hat{\tau}_d \le \tau_{\max}$, and $\text{net} : \mathbb{R}^{nd} \to \mathbb{R}^n$ representing a time delay neural network. Suppose the network has L layers with nonlinearity $f_l(\cdot)$ in each layer $l = 1, \ldots, L$. Let us denote the input as $\mathbf{z}_0^t$ and the output as z_L^t at time t. Then, we have

$$z_L^t = \text{net}(\mathbf{z}_0^t), \tag{13}$$

where

$$\begin{aligned} z_1^t &= f_1(W_1 \mathbf{z}_0^t + b_1), \\ z_l^t &= f_l\big(W_l z_{l-1}^t + b_l\big), \quad l \ge 2, \ldots, L. \end{aligned} \tag{14}$$

This network can be used to approximate the state derivative of the delayed dynamical system by defining $z_L^t = \dot{\hat{x}}(t)$ and $\mathbf{z}_0^t = \hat{\mathbf{x}}_t$, cf. (12) and (13).

Note that some of the delays can potentially have the same value and they do not have to be in an ascending/descending order. Therefore, when the NDDE (12) is used to approximate the dynamics of the time delay system (11), knowing the number of delays in the system is beneficial but not required [38]. Figure 2a illustrates a time delay neural network used to capture the right-hand side of an NDDE with discrete delays, while panel (b) shows a block diagram illustrating the corresponding NDDE simulation. Below we consider the time delay neural networks with trainable delays. That is, apart from the weights W_l and biases b_l, $l = 1, \ldots, L$ in (14), we also learn the delays $\hat{\tau}_k$, $k = 1, \ldots, d$.

For example, an autonomous delay dynamical system with one delay ($d = 1$) can be described by

$$\dot{x}(t) = g(x(t), x(t-\tau)), \tag{15}$$

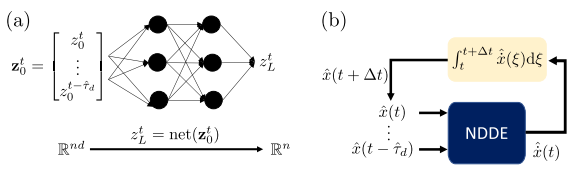

Fig. 2 **a** Illustration of a time delay neural network constructed to capture the dynamics of a neural delay differential equation (NDDE) with a single delay. **b** Illustration of the simulation of an NDDE

whose nonlinearity $g(\cdot)$ and delay τ may be unknown. Our goal is to learn $g(\cdot)$ as well as τ from the data using the NDDE

$$\dot{\hat{x}}(t) = \text{net}(\hat{x}(t), \hat{x}(t-\hat{\tau})), \tag{16}$$

with trainable delay $\hat{\tau}$. Control systems with state feedback can also be viewed as autonomous systems. For example, the dynamical system $\dot{x}(t) = f(x(t), u(t))$ with delayed state feedback $u(t) = k(x(t-\tau))$ yields

$$\dot{x}(t) = f\big(x(t), k(x(t-\tau))\big). \tag{17}$$

When the nonlinear dynamics $f(\cdot,\cdot)$ of the system, the feedback law $k(\cdot)$, and the delay τ are not known, we need to learn these from trajectory data.

Due to the trainable delays in the networks and due to the recurrent feature of network simulation, the time delay neural network is able to take the time dependency into consideration, while keeping the number of parameters relatively low. Moreover, the neural network (13) captures the dynamics of the delay differential equation (12), and it can be analyzed using tools from the dynamical systems and control literature. Additionally, with explicit delay parameters in the input layer, the network is more interpretable and generalizable. The trained delays and nonlinearities are independent from each other, that is, the network does not need to be retrained once the delays is changed.

4 Learning algorithm

In this section, we construct the loss function for network training and illustrate the training algorithm based on discrete delays as demonstrated in Fig. 2a. For a given initial history, which is obtained by linear interpolation from sampled data, one can solve the NDDE with a chosen DDE solver. Then, the DDE solver can be used to predict the state at time t as

$$\hat{x}(t) = \text{DDEsolver}(\text{net}, x_{t_0}, t), \tag{18}$$

where the history function $x_{t_0}(\vartheta)$, $\vartheta \in [t_0 - \tau_{\max}, t_0]$ is approximated from data $x(t_0), x(t_0 - \Delta t), \ldots, x(t_0 - \tau_{\max})$ and Δt denotes the sampling time.

In this work, we use the 4-step Adams–Bashforth scheme with fixed simulation time step δt to obtain the solution (18); see [44] for ODE implementation, and Appendix A for DDE implementation. Note that the time step δt can be smaller than the sampling time Δt to achieve better accuracy. One may also discretize the

DDE along the history and obtain simulation from the resulting set of ordinary differential equations (ODEs); see [38] for more details.

In order to measure the error between the simulation and the data, we construct the simulation loss function

$$\mathcal{L}_{\text{sim}} = \frac{1}{nH}\sum_{j=1}^{H}\sum_{i=1}^{n}\left(\hat{x}^{(i)}(t_0+j\Delta t)-x^{(i)}(t_0+j\Delta t)\right)^2, \tag{19}$$

where $H\Delta t$ is the simulation horizon and $x^{(i)}$ denotes the i-th component of the vector $x \in \mathbb{R}^n$. Since the data is sampled at $t = t_0 + j\Delta t$, the network simulation is also evaluated at the same time moments.

In order to train the network parameter θ, which incorporates the weights W_l and biases $b_l, l = 1, \ldots, L$, and the delays $\hat{\tau}_k, k = 1, \ldots, d$ in the time delay neural network (13), we use adaptive moment estimation [45]. This requires the calculation of the gradient $\frac{\partial \mathcal{L}_{\text{sim}}}{\partial \theta}$, that is, $\frac{\partial \mathcal{L}_{\text{sim}}}{\partial W_l}$, $\frac{\partial \mathcal{L}_{\text{sim}}}{\partial b_l}$, and $\frac{\partial \mathcal{L}_{\text{sim}}}{\partial \hat{\tau}_k}$. The gradient of the loss with respect to delay $\hat{\tau}_k$ is given by

$$\frac{\partial \mathcal{L}_{\text{sim}}}{\partial \hat{\tau}_k} = \frac{\partial \mathcal{L}_{\text{sim}}}{\partial x_{t_0}(-\hat{\tau}_k)}\frac{\partial x_{t_0}(-\hat{\tau}_k)}{\partial \hat{\tau}_k} = \frac{\partial \mathcal{L}_{\text{sim}}}{\partial x_{t_0}(-\hat{\tau}_k)}\dot{x}_{t_0}(-\hat{\tau}_k). \tag{20}$$

Here, $\hat{\tau}_k$ is a continuous variable, so $x_{t_0}(-\hat{\tau}_k)$ and $\dot{x}_{t_0}(-\hat{\tau}_k)$ are approximated from the linear interpolation of the history samples. Namely, one can express any delay value as $\tau = (j+\alpha)\Delta t$ with $j \in \mathbb{N}$ and $\alpha \in [0, 1)$, and approximate the state at $t - \tau$ as

$$\begin{aligned} x(t-\tau) &= x(t-(j+\alpha)\Delta t) \\ &\approx (1-\alpha)x(t-j\Delta t) + \alpha x(t-(j+1)\Delta t). \end{aligned} \tag{21}$$

The state derivatives can be obtained as follow

$$\begin{aligned} \dot{x}(t-\tau) &\approx \frac{x(t-j\Delta t)-x(t-(j+\alpha)\Delta t)}{\alpha\Delta t} \\ &\overset{(21)}{\approx} \frac{x(t-j\Delta t)-x(t-(j+1)\Delta t)}{\Delta t}, \end{aligned} \tag{22}$$

where Euler's method is used in the first approximation.

The gradients $\frac{\partial \mathcal{L}_{\text{sim}}}{\partial W_l}$, $\frac{\partial \mathcal{L}_{\text{sim}}}{\partial b_l}$, and $\frac{\partial \mathcal{L}_{\text{sim}}}{\partial \hat{\tau}_k}$ are calculated via backpropagation, utilizing the automatic differentiation through the DDE solver. Although the training dataset may consist only one trajectory, it can be separated into shorter segments using the horizon $H\Delta t$ in (19) and the length of history $\tau_{\max}$. The loss function (19) measures the difference between the data and the trajectory generated by the neural delay differential equation for a particular initial history. When using multiple segments, we define the cost function

$$\mathcal{L} = \frac{1}{N}\sum_{m=1}^{N}\mathcal{L}_{\text{sim}}(x_{t_0^m}), \quad x_{t_0^m} \in \mathcal{S}, \tag{23}$$

where N is the number of segments (batch size) used for one update and the set $\mathcal{S}$ contain the potential initial histories in the training dataset.

In this work, we implement the training method using the 4-step Adams–Bashforth scheme, together with `dlgradient` and `adamupdate` from MATLAB Deep Learning Toolbox. The sample code of the NDDE solver with multiple delays is provided in Appendix A and other essential codes for implementation of learning are provided in Appendices B and C. In the adaptive moment estimation (`adamupdate`), different learning rates were chosen for delays and for the other parameters. In addition to the parameter updates with the gradient information, we restrict the delay value to be between 0 and $\tau_{\max}$ considering the positivity of the delay and the length of memory. The algorithm for multiple-delay learning is illustrated in Algorithm 1.

5 Mackey–Glass dynamics

Now we examine the NDDE approach and the proposed learning algorithm when learning the dynamics of the Mackey–Glass system [47], a scalar autonomous time delay system modeled by the equation

$$\dot{x}(t) = \frac{\beta x(t-\tau)}{1+(x(t-\tau))^{\delta}} - \gamma x(t), \tag{24}$$

with $\beta = 4$, $\gamma = 2$, $\delta = 9.65$. This mathematical model is used to describe the complex behavior in physiological systems with delayed effects, for example, the control of arterial CO_2 concentration and the regulation of hematopoiesis. This is a classical time delay system, known for the simplicity of its structure and for the complexity of its behavior. It exhibits different qualitative behaviors as the time delay τ is varied.

For a small delay, for instance, $\tau = 0.2$, the system has a stable equilibrium at $x(t) \equiv 1$, as shown in Fig. 3a. As the delay increases, the equilibrium loses the stability and a stable limit cycle rises, whose amplitude grows with the delay. Such a stable limit cycle is shown in Fig. 3b for $\tau = 0.5$. Increasing the delay further, the system becomes chaotic, see panels (c) and (d) for

Goal: Learn $\theta = \{W_l, b_l, \hat{\tau}_k\}, l = 1, \ldots, L, k = 1, \ldots, d$ from data.

- Choose the simulation horizon H, the simulation time step δt, and the learning rates η_τ and η for delays and other parameters.
- Set the maximum iteration number $q_{\max}$ and the maximum allowed delay $\tau_{\max}$.
- Initialize θ_0, i.e., initialize $\hat{\tau}_k$ uniformly between $[0, \tau_{\max}]$, b_l as zeros and W_l using Glorot initialization [46]; see Appendix B for an example of initialization code.

for $q = 1, \ldots, q_{\max}$ **do**

- Randomly take N segments from the training data such that the length of each segment depends on horizon H.
- Simulate the NDDE with current weights, biases and delays using `dl_ddeab4`.
- Calculate the cost $\mathcal{L}$ using (19) and (23).
- Back-propagate to get the gradients with respect to parameters using `dlgradient` given in Appendix A.
- Update θ_q from θ_{q-1} using `adamupdate` and impose positivity constraint to delays $\hat{\tau}$; see Appendix C for example.

if $q > 1$ ***and*** $L_q < L_{q-1}$ **then**
  $\theta_{\text{best}} = \theta_q$
end

end

$\theta = \theta_{\text{best}}$, evaluate the NDDE on testing datasets

Algorithm 1: Training algorithm for the time delay neural network with simulation loss

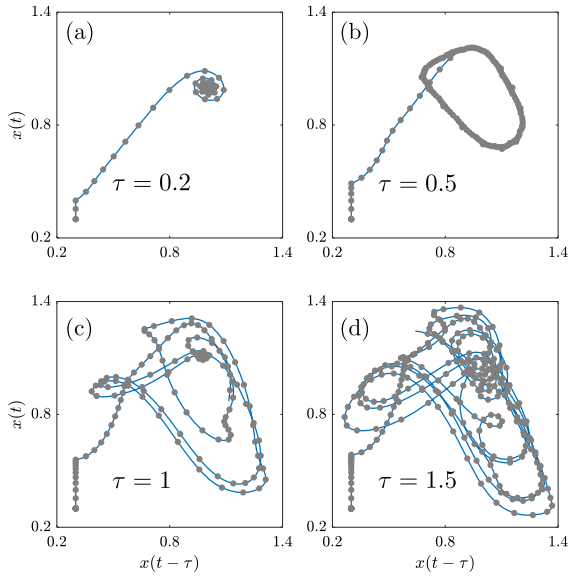

Fig. 3 Training datasets generated by the Mackey–Glass system: each dataset contains one trajectory generated for a different delay value using the initial history $x_0(\vartheta) \equiv 0.3, \vartheta \in [-\tau_{\max}, 0]$ sampled with time step $\Delta t = 0.1$. **a–c** Has the same length $T_{\text{tr}} = 15$, while **d** has the length $T_{\text{tr}} = 30$ since the richer dynamics takes more time to present

$\tau = 1$ and $\tau = 1.5$. The qualitative changes in the system behavior with the change of parameters are studied using bifurcation analysis. The appearance and stability change of equilibria and periodic orbits are usually summarized in bifurcation diagrams; see already Fig. 7.

We use a time delay neural network with two hidden layers, five neurons in each layer, tanh nonlinearity, and one trainable delay to learn from four different datasets, shown in Fig. 3. That is, the data generated by (24) is captured by the NDDE

$$\dot{\hat{x}}(t) = W_3 \tanh\Big(W_2 \tanh\Big(W_1 \begin{bmatrix} \hat{x}(t) \\ \hat{x}(t-\hat{\tau}) \end{bmatrix} + b_1\Big) + b_2\Big), \tag{25}$$

where $W_1 \in \mathbb{R}^{5\times 2}$, $W_2 \in \mathbb{R}^{5\times 5}$, $W_3 \in \mathbb{R}^{1\times 5}$, b_1, $b_2 \in \mathbb{R}^{5\times 1}$.

Note that the datasets shown in Fig. 3 are not combined during training, i.e., only a single trajectory is used in each case. By doing so, we examine the effect of datasets in the training procedure as well as the generalizability. The training horizon is chosen to be $H = 10$, $H\Delta t = 1$ and the simulation time step in the DDEsolver is $\delta t = 0.01$. The learning rate for weights and biases is chosen to be 0.01. The learning rate for delay is 0.01 for the limit cycle dataset (generated at $\tau = 0.5$) and for the chaotic datasets (generated at $\tau = 1, 1.5$), but it is 0.001 for the stable equilibrium dataset (generated at $\tau = 0.2$), because the target delay value is smaller. The batch size $N = 10$ is used for training, while the maximum number of iterations is set to be 2000 for the datasets generated at $\tau = 0.5, 1$ and 5000 for the datasets generated at $\tau = 0.2, 1.5$. The networks with smallest training loss along the iterations are used as final trained network for prediction and evaluation.

The training algorithm performs well on the all four datasets. The trained neural network captures the trajectory for the training data and accurately predicts the future trajectory for a short time period. With a larger simulation horizon $H\Delta t$, the network can be used to predict the chaotic oscillations more accurately for a longer period. However, using larger horizon significantly increases the training time, and the prediction error still increases in the longer term due to the sensi-

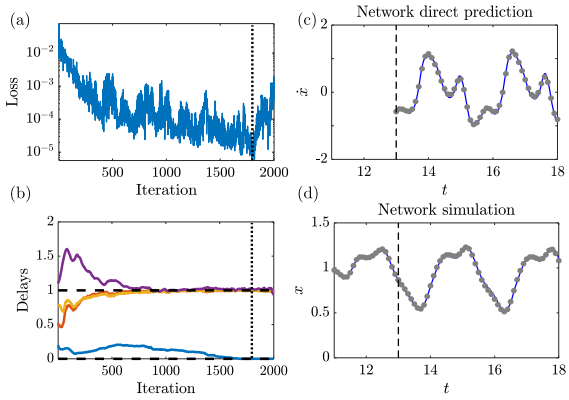

Fig. 4 Learning the Mackey–Glass dynamics. **a** Training loss along the iterations. **b** Learned delays along the iterations. **c** Derivative prediction on test data. **d** Simulation prediction on test data

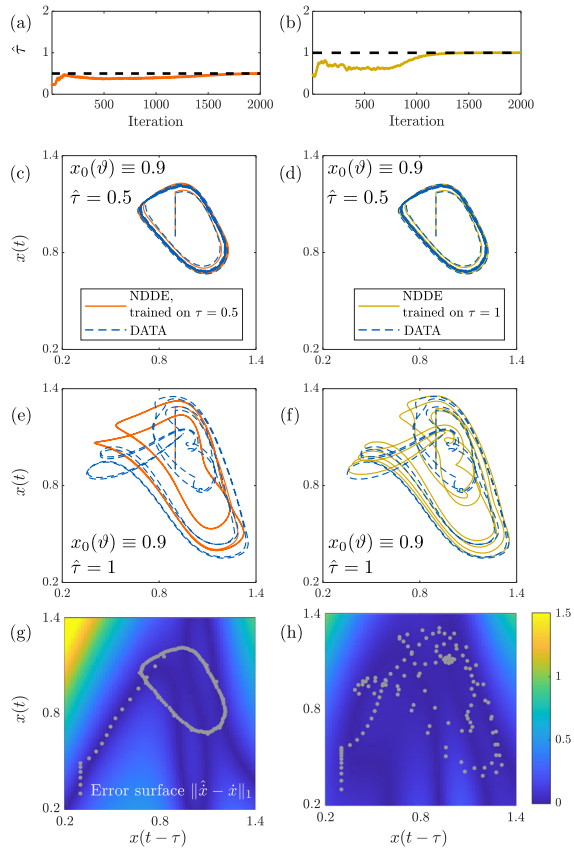

Fig. 5 Results when the networks are trained on the limit cycle dataset $\tau = 0.5$ (left) and the chaotic dataset $\tau = 1$ (right). **a**, **b** Learning the delays from the training data. **c**–**f** Comparison of network performances on test data. **g**, **h** Absolute error between $\dot{x}$ and $\dot{\hat{x}}$ in state space where the gray dots indicate training data

tivity to initial conditions in chaotic systems. To examine the performance of the trained networks, we test them for different initial histories and compare the MLE of the data as with the MLE of the trajectories given by the NDDE. We use bifurcation diagrams to demonstrate that rich data (with positive MLE) yields good generalizability. When the bifurcation diagram of the approximated system matches the bifurcation diagram of the original system, one may use the trained NDDE outside the domain of training data.

There are cases when the number of delays in the system is unknown. A demonstration of the developed algorithm under such a scenario is shown in Fig. 4. Here, we initially overestimate the number of delays to be four while there is only one delay in the system. The corresponding NDDE is now given by

$$\dot{\hat{x}}(t) = W_3 \tanh\Big(W_2 \tanh\Big(W_1 \begin{bmatrix} \hat{x}(t-\hat{\tau}_1) \\ \hat{x}(t-\hat{\tau}_2) \\ \hat{x}(t-\hat{\tau}_3) \\ \hat{x}(t-\hat{\tau}_4) \end{bmatrix} + b_1\Big) + b_2\Big), \tag{26}$$

where $W_1 \in \mathbb{R}^{5\times 4}$ and the dimensions of the other weights and biases are the same as in (25). As a result, some of the delays converge to the true value $\tau = 1$, while others converge to zero (representing the non-delayed terms). The training loss and learned delays are shown in panels (a) and (b), respectively. The dotted line indicates the minimum training error where the parameters used in the NDDE (25) are taken. The direct network prediction of the derivative $\dot{\hat{x}}$ and the simulated state $\hat{x}$ are shown as a function of time in panels (c) and (d) as blue curves, in comparison with the test data plotted as gray dots.

We show the learning path of the delay and the performance of the trained networks (25) on test data in Fig. 5. The performance of the network trained on limit cycle data (generated at $\tau = 0.5$) is presented in orange on the left, and the performance of the network trained on chaotic data (generated at $\tau = 1$) is presented in yellow on the right. Panels (a)–(b) show how the delay value is learned along the training iterations. In both cases, the delay starts from initial value $\hat{\tau} = 0.5\tau$ and approaches the correct delay value τ through the iterations. Both networks are tested on initial history $x_0(\vartheta) \equiv 0.9$ that differs from the initial history $x_0(\vartheta) \equiv 0.3$ used for generating the training data; see panels (c)–(f). One may observe from the phase

Table 1 Delays and maximal Lyapunov exponents of the original Mackey–Glass system and the trained NDDEs for the four datasets in Fig. 3 generated for initial history $x_0 \equiv 0.3$, while using mesh size $M = 10$ and truncation time $T = 10^4$

τ (DATA)	0.2	0.5	1	1.5
$\hat{\tau}$ (NDDE)	0.18	0.5	1	1.5
$\lambda_{\max}$ (DATA)	-0.66	-7.3×10^{-4}	0.143	0.155
$\lambda_{\max}$ (NDDE)	-0.77	-4.7×10^{-5}	0.140	0.153

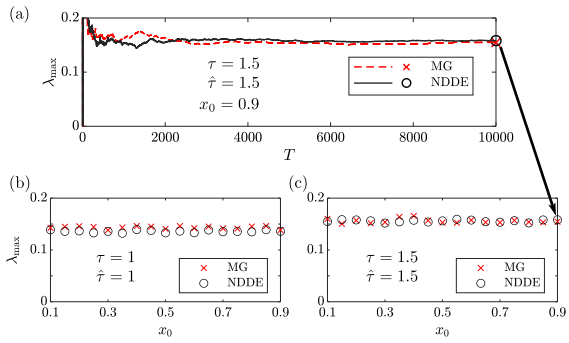

Fig. 6 Maximal Lyapunov exponents computed for the Mackey–Glass system and the trained NDDE. Panel **a** shows the MLEs as a function of the truncation time T for delay $\tau = 1$ and history $x_0 \equiv 0.9$. Panels **b** and **c** show the exponents at $T = 10^4$ for delays $\tau = 1$ and $\tau = 1.5$, respectively, while considering the initial histories $x_0 \equiv c$, $c \in [0.1, 0.9]$

portraits in panels (c) and (f) that both networks give good predictions for the delay value they are trained at.

The trained NDDEs are also tested for another delay value, without retraining, i.e., by keeping the same trained weights and biases; see the phase portraits in panels (d) and (e). The network trained on chaotic data at $\tau = 1$ can predict the limit cycle observed for $\tau = 0.5$, but the network trained on limit cycle data fails to predict the chaotic behavior observed for $\tau = 1$. This indicates that the networks trained on datasets with richer dynamics generalize better. Moreover, we present the difference of the state derivative between the original system and the NDDE in panels (g) and (f). Observe that the network trained on chaotic data $\tau = 1$ has a better approximation of the nonlinearity, as indicated by the blue color.

The richness of the training data affects the learning performance. To evaluate the richness, we calculate the MLE of the original data and compare it to the MLE of the trajectories generated by the trained networks. For the four networks trained on the datasets shown in Fig. 3, the delays and the maximal Lyapunov exponents are collected in Table 1. All networks fit their corresponding training data; thus, the delays and the MLEs are recovered by the NDDEs. The MLE and the learned delay for the network trained on stable equilibrium data are less accurate as most of the stable trajectory is around the equilibrium, which makes the learning difficult.

The computation of the MLE requires the simulation of the system with a given initial history [42]. The computed MLEs are shown in Fig. 6 for different time delays and for different initial histories. The approximate MLEs of the Mackey–Glass system and the learned NDDE are shown in the panel (a), as a function of the truncation time T for $\tau = 1.5$ and history $x_0 = 0.9$. Both systems require a large truncation time to obtain good estimates (convergence) on MLEs. The MLEs are compared at $T = 10^4$ for different initial histories and different delays in panels (b) and (c). Observe that the NDDEs trained on chaotic data estimate well the MLE of the true system regardless of the initial history of the testing trajectory. One may also observe slightly larger MLE in the case $\tau = 1.5$ compared to the case $\tau = 1$.

To illustrate the generalizability of NDDEs, we generate the bifurcation diagrams using DDE-BIFTOOL [48,49] and compare those to the bifurcation diagram of the original Mackey–Glass system in Fig. 7. We plot the peak-to-peak amplitude of the limit cycle $(x_{\max} - x_{\min})$ as a function of delay and indicate the stability/instability of limit cycles by green/red color. The equilibrium $x(t) \equiv 1$ loses its stability via Hopf bifurcation and a stable limit cycle appears at $\tau = 0.24$. The limit cycle undergoes a period doubling cascade with the first two bifurcations located at $\tau = 0.61$ and $\tau = 0.84$. Beyond $\tau = 0.87$, the system has chaotic attractor, which is indicated by gray shading in Fig. 7. From panels (a) and (b), we can see that the NDDEs with $\lambda_{\max} \leq 0$ have poor generalization for delay values greater than the delay in the training data. Contrarily, panels (c) and (d) demonstrate that the NDDEs with $\lambda_{\max} > 0$ predict the appearance of Hopf and period doubling bifurcations and capture the amplitude and stability of the limit cycles for other delay values without retraining. The network trained on data with larger MLE ($\tau = 1.5$ case) has better generalization, i.e., better approximation of the bifurcation diagram.

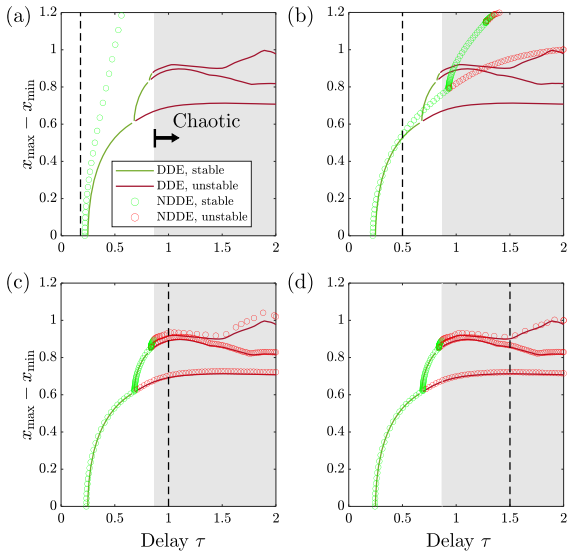

Fig. 7 Bifurcation diagrams of the NDDE learned from data at $\tau = 0.2, 0.5, 1, 1.5$ compared to the ground truth. The dashed line indicates the underlying delay in the training data, and the shaded area indicates where the original Mackey–Glass system exhibits chaotic behavior

Thus, the MLE can be used as a metric to evaluate the richness of the training data as well as the generalizability of the trained NDDE. Introducing time delays into the systems may help to improve the richness of the data. Other evaluation metrics of the data quality include the measurement noise, the sampling frequency, the span of initial conditions, and the ratio of the transients and steady states. We also learned the dynamics of the same systems from noisy data and from data without transient dynamics. NDDEs trained on data which include initial history deliver the best performance, since the transient data, which cover larger part of the state space, have richer dynamics. When noise is introduced into measurements, the delay learning remains robust while the nonlinearity learning degrades.

We also tested the NDDE approach on an Ikeda type system [7,50] which describes the dynamics of passive optical resonators. This is another classical time delay system whose dynamics change qualitatively when the time delay is varied. The details of the training process and the evaluation of the trained NDDE are provided in Appendix D. Again, the time delay is learned correctly from a single chaotic trajectory and the trained NDDE is able to reproduce the first Hopf bifurcation and the first period doubling bifurcation. However, the bifurcation diagram is not accurately reproduced for larger values of the time delay. The transition between limit cycles and chaotic attractors in the Ikeda system is fast compared to the Mackey–Glass system, which makes the numerical bifurcation analysis more challenging.

6 Climate model

Besides nonlinear autonomous systems with single delay, such as Mackey–Glass system and the Ikeda system, we can extend the training algorithms to systems with multiple delays and external forcing. In this section, we use a time delay neural network to learn the dynamics of the climate model

$$\dot{x} = aA(\kappa, x(t-\tau_1)) - bA(\kappa, x(t-\tau_2)) + cu(t), \tag{27}$$

$$A(\kappa, x) = \begin{cases} d_u \tanh\left(\frac{\kappa}{d_u} x\right), & \text{if } x \geq 0, \\ d_l \tanh\left(\frac{\kappa}{d_l} x\right), & \text{if } x < 0, \end{cases} \tag{28}$$

from a single trajectory where the input $u(t) = \cos(2\pi t)$ represents the seasonal forcing.

This model describes the deviation of thermocline depth from its long-term mean at the eastern boundary of the equatorial Pacific Ocean. The chaotic behavior of this simple delay differential equation is used to demonstrate the irregular phenomenon of the El Niño Southern Oscillation (ENSO). More detailed explanations of this model can be found in [51,52]. In our example, the positive and negative feedbacks are associated with two different fixed delays $\tau_1 = 0.0958$ and $\tau_2 = 0.4792$, while other parameters are $a = 2.02$, $b = 3.03$, $c = 2.6377$, $d_u = 2.0$, $d_l = -0.4$. With these parameters, we generate a trajectory using the constant history $x(t) \equiv 1$ as illustrated in Fig. 8a. The data are sampled with $\Delta t = 0.05$, indicated by gray dots. The simulation time step is $\delta t = 0.01$ and simulation horizon is chosen to be $H = 10$.

For this learning task, we use a time delay neural network which takes the forcing $u(t)$ and state history x_t as input and gives state derivative $\dot{x}(t)$ as output. The network consists of two hidden layers, five neurons in each layer, and three delays in the state. Note that the true number of delays in the system (27) is two (and there is no nondelayed state). That is, we moderately

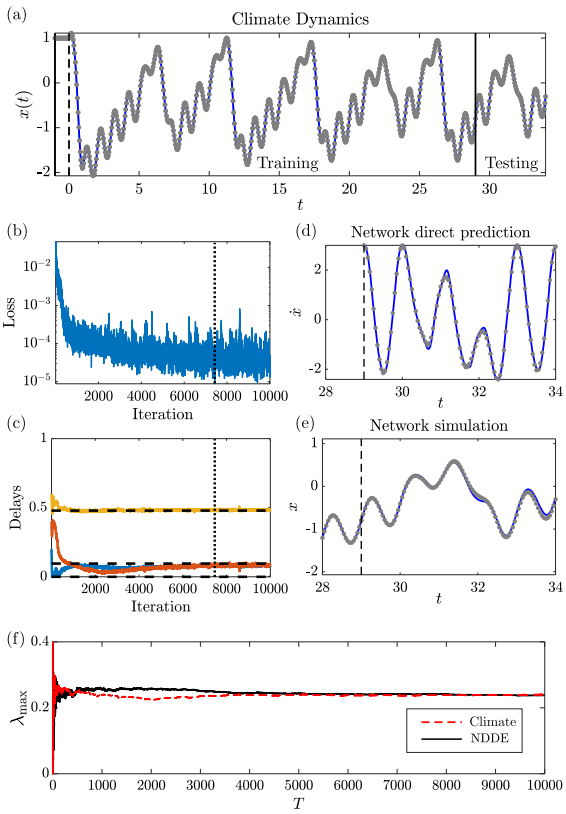

Fig. 8 Training and testing performance of the climate model. **a** Training and testing datasets obtained from a single chaotic trajectory. **b** Training loss along the iterations. **c** Learned delays along the iterations. **d** Derivative prediction on test data. **e** Simulation prediction on test data. **f** Maximal Lyapunov exponents of the climate model and the learned NDDE while using $M = 10$

overestimate the number of delays. With three trainable delays, the dynamics of the NDDE takes the form

$$\hat{\dot{x}}(t) = W_3 \tanh\left(W_2 \tanh\left(W_1 \begin{bmatrix} \hat{x}(t-\hat{\tau}_1) \\ \hat{x}(t-\hat{\tau}_2) \\ \hat{x}(t-\hat{\tau}_3) \\ u(t) \end{bmatrix} + b_1\right) + b_2\right), \tag{29}$$

where $W_1 \in \mathbb{R}^{5\times 4}$, $W_2 \in \mathbb{R}^{5\times 5}$, $W_3 \in \mathbb{R}^{1\times 5}$, b_1, $b_2 \in \mathbb{R}^{5\times 1}$. Here, $u(t)$ is obtained from interpolating the recorded external forcing data. Thus, with a given initial history x_{t_0} and forcing $u(t)$, one can simulate the NDDE as (18) and learn the weights, biases and delays using the loss (19) and Algorithm 1. The learning rates for all parameters are set to 0.01, the batch size is chosen to be $N = 10$, and the maximum number of iterations is set to 10000.

We present the training results of the NDDE on single trajectory in Fig. 8. From panel (c), we can see that two of the three delays converge to the same value τ_1 and the third one converges to τ_2. The final parameters of the NDDE are determined by the minimum training loss which is indicated by the dotted lines in panels (b) and (c). The derivative prediction and closed-loop network simulation on the test dataset are shown in panels (d) and (e), respectively. This example shows that the developed algorithm can be extended to systems with external forcing where using a chaotic trajectory still provides good performance due to its rich dynamics. The MLE of the trained network is compared with that of the true system in panel (f). This shows that the network trained on one chaotic trajectory can reproduce the MLE of the original system. The bifurcation diagrams are not presented here as they are require more delicate analysis due to the periodic excitation and this is beyond the scope of this work. It is an intriguing problem for future study.

7 Conclusions and discussion

We proposed a framework which uses time delay neural networks with trainable delays and simulation loss to learn the dynamics of time delay systems from data and yields neural delay differential equations. We showed that by including time delays, one may reduce the structural complexity of the networks. As an example, the Mackey–Glass system was used to test the developed algorithm. We demonstrated that delay can be learned from limited data and the network generalizes well if the training data is chaotic. Thus, the maximal Lyapunov exponent can be used to evaluate the richness of the training data and the generalizability of trained networks. We found that the larger the MLE is, the richer the dynamics are and the better the trained network generalizes. We also showed that the learning algorithm worked well on learning the dynamics of a climate model with two delays and external forcing from a single trajectory. This again demonstrated the effectiveness of learning from chaotic trajectory data.

The algorithm requires one to select the maximum allowed delay for the network to prevent having the entire trajectory as history. Establishing an algorithmic selection of the maximum delay can be a potential future study. While the time step Δt depends on the sampling data, the effects of simulation time step δt in

DDE solver and the simulation horizon H may also be studied in the future. Intuitively, faster dynamics will require higher sampling frequency and smaller simulation time step.

We have also tested the algorithm on multiple-state time delay systems, such as delayed SIR model in epidemiology and delayed predator–prey model in biology. While the delay learning from the state derivatives is still robust in those systems, learning the nonlinearities is more difficult due to the positivity constraint imposed on the states. As a future direction, we will keep exploring multiple-state time delay systems to improve the learning algorithm and realize more complex NDDEs. We also plan to use the time delay neural networks with trainable delays to learn time delay systems from real data, e.g., in case of human posture control and vehicular traffic. Finally, we consider injecting delays into the feedback loops of control systems in order to obtain better training data.

Author contributions Xunbi A. Ji developed and coded the algorithm and performed the evaluations. Gabor Orosz supervised the research. The first draft of the manuscript was written by Xunbi A. Ji and both authors commented on previous versions of the manuscript. Both authors read and approved the final manuscript.

Funding The authors acknowledge the support from Integrative System and Design in University of Michigan.

Data availability statement The data used in this paper are all generated from provided equations, they are also available from the corresponding author on reasonable request. The essential codes of the algorithm are provided in the Appendices.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

Appendices

A MATLAB code for NDDE solver

```
%% 4-step Adams--Bashforth scheme to solve DDEs
%% with multiple delays
function xsolu=dl_ddeab4(fun,par,delay,hist,time)
dim=length(hist(time(1)));
h=time(2)-time(1);
Maxdelay = max(ceil(max(delay)/h)*h,h);
timehist=fliplr(time(1)-Maxdelay:h:time(1));
z0=dlarray(zeros(dim,length(timehist)));
for i=1:length(timehist)
    z0(:,i)=hist(timehist(i));
end
z0=z0(:);
xsolu=dlarray(zeros(dim,length(time)));
for kk=1:length(time)
    timestep
    tnew=time(kk);
    if kk==1
        znew=z0;
    else
        xold=zold(1:dim);
        Z = reshape(zold,[dim length(timehist)]);
        xdelay = interp1(fliplr(timehist),...
        fliplr(Z)', time(1)-delay)';
        xdelay = reshape(xdelay,[],1);
        rhs1=fun(told,xold,xdelay,par);
        % calculation of the solution
        if kk==2
            xnew = xold+h*rhs1;
        elseif kk==3
            xnew = xold+h*(3*rhs1-rhs2)/2;
        elseif kk==4
            xnew = xold+h*...
            (23*rhs1-16*rhs2+5*rhs3)/12;
        elseif kk>4
            xnew=xold+h*...
            (55*rhs1-59*rhs2+37*rhs3-9*rhs4)/24;
        end
        znew=[xnew;zold(1:end-dim)];
    end
    if kk>3
        rhs4=rhs3;
    end
    if kk>2
        rhs3=rhs2;
    end
    if kk>1
        rhs2=rhs1;
    end
    told=tnew;
    zold=znew;
    xsolu(:,kk)=znew(1:dim);
end
```

B Example code for the NDDE initialization

```
%% an example of function
%% as the input of DDE solver above
function dx=fun(t,x,xdelay,par)
dx = par.fc3.Weights*tanh(par.fc2.Weights...
    *tanh(par.fc1.Weights*...
    [x;xdelay]+par.fc1.Bias)...
    +par.fc2.Bias);
end

function bias = initializeZeros(sz)
bias = zeros(sz,'single');
bias = dlarray(bias);
end

function weights = initializeGlorot(sz)
```

```
Z = 2*rand(sz,'single') - 1;
bound = sqrt(6 / (sz(2)+ sz(1)));
weights = bound * Z;
weights = dlarray(weights);
end

%% initialization of the parameters
% guess the number of delay 'nd',
% initalize randomly as dlarray
% NDDE will be the par in the fun
tau = dlarray(tau_max*rand(1,nd));
% an example of network
NDDE = struct;
NDDE.fc1 = struct;
sz = [hiddenSize nx*(1+nd)];
NDDE.fc1.Weights = initializeGlorot(sz);
NDDE.fc1.Bias    = initializeZeros([sz(1) 1]);
NDDE.fc2 = struct;
sz = [hiddenSize hiddenSize];
NDDE.fc2.Weights = initializeGlorot(sz);
NDDE.fc2.Bias    = initializeZeros([sz(1) 1]);
NDDE.fc3 = struct;
sz = [nx hiddenSize];
NDDE.fc3.Weights = initializeGlorot(sz);
tau = min(max(1e-5,tau),tau_max-1e-5);
```

C Code snippets for training

```
% xsolu is from dde solver 'dl_ddeab4'
% x_target is the observed data
loss = l2loss(xsolu,x_tartget)
grad_par = dlgradient(loss,par);
grad_tau = dlgradient(loss,par);
% adamupdate as an example
[par,averageGrad_p,averageSqGrad_p] = ...
adamupdate(par,grad_par,averageGrad_p,...
averageSqGrad_p,iteration);
[tau,averageGrad_t,averageSqGrad_t] = ...
adamupdate(tau,grad_tau,averageGrad_t,...
averageSqGrad_t,iteration)
```

D Bifurcation analysis of the Ikeda system

We train the same neural network (25) on one simulation trajectory (with $T = 30$, $\Delta t = 0.1$) for the Ikeda system

$$\dot{x}(t) = \beta \sin^2(x(t-\tau)) - \gamma x(t), \tag{30}$$

with parameters $\beta = 8, \gamma = 2, \tau = 1$ and initial history $x(t) \equiv 1$. In the training process, the simulation time step for the DDE solver is set to $\delta t = 0.01$, while the horizon is $H = 10$, $H\Delta t = 1$. The learning rate for delay is 0.01 and for weights and biases is 0.1. Maximum number of iterations is set to 5000.

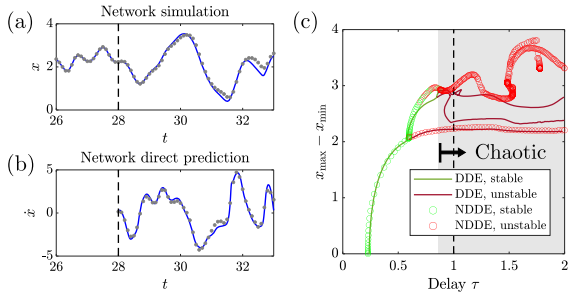

Fig. 9 Training and testing performance of the Ikeda system. **a** Derivative prediction on test data. **b** Simulation prediction on test data. **c** Bifurcation diagram

The network successfully learns the delay $\hat{\tau} = 1$ from the data and the predictions obtained by the trained network are given in Fig. 9a and b. The bifurcation diagrams of the original and the learned systems are compared in panel (c). The bifurcation diagrams match well for smaller delay values, while there are some differences for larger values of the delay (where even the bifurcation analysis of the original Ikeda system becomes numerically challenging). Note that the network is only trained on data generated at $\tau = 1$.

References

1. Pei, J.S., Mai, E.C., Wright, J.P., Masri, S.F.: Mapping some basic functions and operations to multilayer feedforward neural networks for modeling nonlinear dynamical systems and beyond. Nonlinear Dyn. **71**(1–2), 371 (2013)
2. Brunton, S.L., Kutz, J.N.: Data-driven Science and Engineering. Data-driven Science and Engineering Cambridge University Press (2019)
3. Canziani, A., Paszke, A., Culurciello, E.: An analysis of deep neural network models for practical applications. Preprint at arXiv:1605.07678 (2016)
4. Ji, X.A., Molnár, T.G., Avedisov, S.S., Orosz, G.: Feedforward neural networks with trainable delay. In: 2nd Conference on Learning for Dynamics and Control, vol. 120, pp. 127–136 PMLR (2020)
5. Wong, S., Jiang, L., Walters, R., Molnár, T.G., Orosz, G., Yu, R.: In: 3rd Conference on Learning for Dynamics and Control, pp. 917–929 PMLR (2021)
6. Koch, J., Maxner, T., Amatya, V., Ranjbari, A., Dowling, C.: Physics-informed machine learning of parameterized fundamental diagrams. Preprint at arXiv:2208.00880 (2022)
7. Goldmann, M., Mirasso, C.R., Fischer, I., Soriano, M.C.: Learn one size to infer all: exploiting translational symmetries in delay-dynamical and spatiotemporal systems using scalable neural networks. Phys. Rev. E **106**, 044211 (2022)
8. Levine, M., Stuart, A.: A framework for machine learning of model error in dynamical systems. Commun. Am. Math. Soc. **2**(07), 283 (2022)

9. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735 (1997)
10. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
11. Chen, R.T.Q., Rubanova, Y., Bettencourt, J., Duvenaud, D.: Neural ordinary differential equations. Advances in Neural Information Processing Systems (2018)
12. Turan, E.M., Jäschke, J.: Multiple shooting for training neural differential equations on time series. IEEE Control Syst. Lett. **6**, 1897 (2021)
13. Rahman, A., Drgoňa, J., Tuor, A., Strube, J.: Neural ordinary differential equations for nonlinear system identification. In: American Control Conference (ACC), pp. 3979–3984 IEEE (2022)
14. Zhu, Q., Guo, Y., Lin, W.: Neural delay differential equations. In: International Conference on Learning Representations (2021)
15. Zhu, Q., Guo, Y., Lin, W.: Neural delay differential equations: system reconstruction and image classification. Preprint at arXiv:2304.05310 (2023)
16. Gupta, A., Lermusiaux, P.F.: Neural closure models for dynamical systems. Proc. R. Soc. A **477**(2252), 20201004 (2021)
17. Gupta, A., Lermusiaux, P.F.: Generalized neural closure models with interpretability. Sci. Rep. **13**, 10634 (2023)
18. Diekmann, O., van Gils, S.A., Verduyn Lunel, S.M., Walther, H.O.: Stability of Time-Delay System. Delay Equations, Springer (1995)
19. Gu, K., Kharitonov, V.L., Chen, J.: Stability of Time-Delay system. Stability of Time-Delay System, Springer (2003)
20. Michiels, W., Niculescu, S.I.: Stability and stabilization of time-delay systems. Stability and Stabilization of Time-Delay Systems, SIAM (2007)
21. Krstic, M.: Delay compensation for nonlinear, adaptive, and PDE systems. In: Delay Compensation for Nonlinear, Adaptive, and PDE Systems, Birkhäuser (2003)
22. Insperger, T., Stépán, G.: Semi-discretization for time-delay systems. In: Semi-Discretization for Time-Delay Systems, Springer (2011)
23. Fridman, E.: Dynamics of vehicle stability control subjected to feedback delay. In: Introduction to Time-Delay Systems, Birkhäuser (2014)
24. Breda, D., Maset, S., Vermiglio, R.: Stability of linear delay differential equations. In: Stability of Linear Delay Differential Equations, Springer (2015)
25. Larger, L., Goedgebuer, J.P., Udaltsov, V.: Ikeda-based nonlinear delayed dynamics for application to secure optical transmission systems using chaos. Comptes Rendus Physique **5**(6), 669 (2004)
26. Rontani, D., Locquet, A., Sciamanna, M., Citrin, D.S., Ortin, S.: Time-delay identification in a chaotic semiconductor laser with optical feedback: a dynamical point of view. IEEE J. Quantum Electron. **45**(7), 879 (2009)
27. Zhang, J.Z., Jin, Z., Liu, Q.X., Zhang, Z.Y.: Analysis of a delayed SIR model with nonlinear incidence rate. Discrete Dyn. Nat. Soc. **2008** (2008)
28. Molnár, T.G., Singletary, A.W., Orosz, G., Ames, A.D.: Safety-critical control of compartmental epidemiological models with measurement delays. IEEE Control Syst. Lett. **5**(5), 1537 (2020)
29. Panday, P., Samanta, S., Pal, N., Chattopadhyay, J.: Delay induced multiple stability switch and chaos in a predator–prey model with fear effect. Math. Comput. Simul. **172**, 134 (2020)
30. Keane, A., Krauskopf, B., Dijkstra, H.A.: The effect of state dependence in a delay differential equation model for the El Niño Southern Oscillation. Philos. Trans. R. Soc. A **377**(2153), 20180121 (2019)
31. Molnár, T.G., Dombóvári, Z., Insperger, T., Stépán, G.: On the analysis of the double Hopf bifurcation in machining processes via centre manifold reduction. Proc. R. Soc. A **473**(2207), 20170502 (2017)
32. Lu, H., Stépán, G., Lu, J., Takács, D.: Dynamics of vehicle stability control subjected to feedback delay. Eur. J. Mech. A/Solids **96**, 104678 (2022)
33. Beregi, S., Avedisov, S.S., He, C.R., Takács, D., Orosz, G.: Connectivity-based delay-tolerant control of automated vehicles: theory and experiments. IEEE Trans. Intell. Veh. **8**(1), 275 (2023)
34. Orosz, G., Wilson, R.E., Szalai, R., Stépán, G.: Exciting traffic jams: nonlinear phenomena behind traffic jam formation on highways. Phys. Rev. E **80**(4), 046205 (2009)
35. Avedisov, S.S., Bansal, G., Orosz, G.: Impacts of connected automated vehicles on freeway traffic patterns at different penetration levels. IEEE Trans. Intell. Transp. Syst. **23**(5), 4305 (2020)
36. Ji, X.A., Molnár, T.G., Avedisov, S.S., Orosz, G.: Learning the dynamics of time delay systems with trainable delays. In: 3rd Conference on Learning for Dynamics and Control, vol. 144, pp. 930–942, PMLR (2021)
37. Ji, X.A., Molnár, T.G., Gorodetsky, A.A., Orosz, G.: Bayesian inference for time delay systems with application to connected automated vehicles. In: IEEE Conference on Intelligent Transportation Systems Conference (ITSC), pp. 3259–3264 (2021)
38. Ji, X.A., Orosz, G.: Learning time delay systems with neural ordinary differential equations. IFAC-PapersOnLine **55**(36), 79 (2022)
39. Tél, T., Gruiz, M.: Chaotic Dynamics. Chaotic Dynamics, Cambridge University Press (2006)
40. Stépán, G.: Retarded dynamical systems. In: Retarded Dynamical Systems, Longman (1989)
41. Wernecke, H., Sándor, B., Gros, C.: Chaos in time delay systems, an educational review. Phys. Rep. **824**, 1 (2019)
42. Breda, D., Della Schiava, S.: Pseudospectral reduction to compute Lyapunov exponents of delay differential equations. Discrete Contin. Dyn. Syst.-B **23**(7), 2727 (2018)
43. Dieci, L., Jolly, M.S., Van Vleck, E.S.: Numerical techniques for approximating Lyapunov exponents and their implementation. J. Comput. Nonlinear Dyn. **6**(1) (2011)
44. Butcher, J.C.: Numerical methods for ordinary differential equations. In: Numerical Methods for Ordinary Differential equations, Wiley (2016)
45. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. In: Bengio, Y., LeCun, Y. (eds) 3rd International Conference on Learning Representations (2015)
46. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: 13th International Conference on Artificial Intelligence and Statistics, JMLR Workshop and Conference Proceedings, pp. 249–256 (2010)

47. Mackey, M.C., Glass, L.: Oscillation and chaos in physiological control systems. Science **197**(4300), 287 (1977)
48. Engelborghs, K., Luzyanina, T., Roose, D.: Numerical bifurcation analysis of delay differential equations using DDE-BIFTOOL. ACM Trans. Math. Softw. **28**(1), 1 (2002)
49. Sieber, J., Engelborghs, K., Luzyanina, T., Samaey, G., Roose, D.: DDE-BIFTOOL manual-bifurcation analysis of delay differential equations. Preprint at arXiv:1406.7144 (2014)
50. Ikeda, K., Matsumoto, K.: High-dimensional chaotic behavior in systems with time-delayed feedback. Physica D **29**(1–2), 223 (1987)
51. Keane, A., Krauskopf, B., Postlethwaite, C.M.: Climate models with delay differential equations. Chaos **27**(11), 114309 (2017)
52. Saunders, A., Ghil, M.: A Boolean delay equation model of ENSO variability. Physica D **160**(1–2), 54 (2001)