Trainable Delays in Time Delay Neural Networks for Learning Delayed Dynamics

Xunbi A. Ji and Gábor Orosz, *Senior Member, IEEE*

***Abstract*— In this article, the connection between time delay systems and time delay neural networks (TDNNs) is presented from a continuous-time perspective. TDNNs are utilized to learn the nonlinear dynamics of time delay systems from trajectory data. The concept of TDNN with trainable delay (TrTDNN) is established, and training algorithms are constructed for learning the time delays and the nonlinearities simultaneously. The proposed techniques are tested on learning the dynamics of autonomous systems from simulation data and on learning the delayed longitudinal dynamics of a connected automated vehicle (CAV) from real experimental data.**

***Index Terms*— Dynamical systems, machine learning, time delay neural network (TDNN), time delay system.**

I. Introduction

OVER the past a few decades, the development of the neural networks and machine learning techniques has flourished in many fields. Researchers have been trying to bridge the gap between dynamical systems and neural networks in different ways. On the one hand, neural networks may be used to govern the time evolution of states in discrete-time and continuous-time dynamical systems. Thus, the behavior of the networks can be studied from the dynamical systems' point of view [1], [2]. On the other hand, neural networks have been widely used for system identification and control in recent years [3], [4], [5], [6]. In many studies, neural networks are used to approximate the solution of the underlying dynamical system in discrete time. For example, [7] used a feedforward neural network to learn the input–output map of the plant, and [8], [9] used recurrent neural networks to identify the unknown dynamics. The neural networks are also used as controllers [10], [11] and as predictors [12] in many research studies.

Time delays were incorporated in neural network architectures as "memory" to construct complex mappings, and neural networks with delays were utilized to represent and control nonlinear systems in discrete time [13], [14], [15]. However, those delays did not correspond to the physical time delays, which appear in the real-world applications, such as climate systems [16], epidemiology models [17], [18], and road transportation systems [19], [20]. State and input delays were considered in the design of control algorithms for continuous-time models [21], [22], [23], and stability analysis of time delay neural networks (TDNNs) was utilized to assist modeling and control [24], [25], [26], [27]. There also exists comprehensive literature on the dynamics and control of time delay systems [28], [29], [30], [31], [32], [33], [34], awaiting to be fully utilized in the area of neural networks. The dynamics of time delay systems can be potentially learned from data using continuous-time models, which requires one to utilize tools both from dynamical systems and from neural networks. This article intends to make a stride in the direction of relating TDNNs and time delay systems in continuous time.

Since neural networks are shown to be powerful function approximators, one may utilize them to approximate the right-hand side of the differential equations governing the time evolution of the states in dynamical systems, as illustrated on the left of Fig. 1. For example, one may construct neural ordinary differential equations (NODEs) [35], which are continuous-depth models that parameterize the right-hand side using neural networks. Similar ideas have also been put forward earlier via continuous-time recurrent neural networks (CTRNNs) [36]. Both approaches model the state derivative and consider the solution of the differential equations as output, while the hidden-state derivatives are represented in a slightly different way in the two cases. It was proved in [37] that the dynamical systems without input can be approximated by CTRNNs, and in [38], this was extended to a general class of continuous-time dynamical systems.

Learning the dynamics of time delay systems is, in general, more difficult than learning delay-free dynamics, since the trajectories are embedded in an infinite dimensional state space [32], [34]. We aim to extend the concept of learning the dynamics with neural networks to the time delay systems, as illustrated on the right of Fig. 1. The time delays can result in rich dynamics, including periodic motions [39], quasi-periodic motion [40], and even chaos [16], [41], [42], while introducing the delays into the networks increases the flexibility and interpretability of the networks. For example, due to the rich dynamics resulting from time delays, a complicated deep neural network can be represented with a simpler network with delays [43]. Extending the concept of NODEs, neural delay differential equations (NDDEs) can be constructed [44], [45], where the right-hand side is represented by a neural network, which contains delays. In [46] and [47], neural networks with explicit delays were used for learning the continuous-time time delay nonlinear systems. However, prior results approaches considered the delays to be known, which may not hold in

Manuscript received 18 September 2023; revised 27 January 2024; accepted 12 March 2024. Date of publication 28 March 2024; date of current version 1 March 2025. *(Corresponding author: Xunbi A. Ji.)*

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

Gábor Orosz is with the Department of Mechanical Engineering and the Department of Civil and Environmental Engineering, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: orosz@umich.edu).

Digital Object Identifier 10.1109/TNNLS.2024.3379020

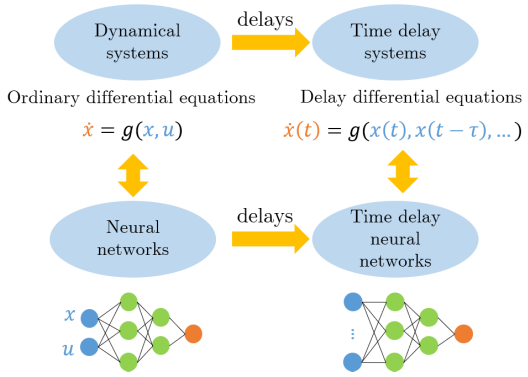

Fig. 1. Relationships between dynamical systems and neural networks without and with time delay.

many applications. In many practical problems, the existence and the sources of the delays are known, but their values are unknown. Our goal is to establish general gradient-based methods for learning the time delays from data.

In this article, we represent the right-hand side using TDNNs with trainable delays (TrTDNNs), which can capture the nonlinearities and the time delays in the system. The trained networks allow us to simulate continuous-time systems and predict the state of these systems at any future time moment. We aim to learn the direct mapping from the delayed values of states and control inputs to the derivative of the state (where the latter is obtained numerically from data), instead of learning the dynamics indirectly via the error between simulation output and the data as in [35], [43], [44], [45], and [48]. This reduces the computation cost in training and provides more flexibility in the choice of networks, e.g., allow the use of more general TDNNs with multiple input delays. We introduce a limited number of trainable time delays and construct network architecture, so that the learned delays have clear physical interpretations. Then, we use gradient-based training algorithms to enable the TrTDNNs to learn the delays and the nonlinearities in the system simultaneously. Some preliminary works have been presented in [49], where a TrTDNN was used to learn the control law of a connected automated vehicle (CAV), and in [50], where the method was extended to multiple delays.

The contributions of this article include the following.

1) We relate TDNNs and time delay system from a continuous-time perspective and construct TDNNs for learning the delayed dynamics from trajectory data.
2) We propose trainable delays in TrTDNNs and develop gradient-based training algorithms, which balance between global search and local search in parameter space.
3) The parameter update rules are derived analytically for two gradient-based methods. The implementation codes and data are provided on GitHub (link).
4) Examples with both simulation data and real experimental data are provided to demonstrate the robustness of developed methods.

The rest of this article is organized as follows. We specify the problem setting in Section II. We introduce TrTDNNs, the loss functions, and the gradient-based training algorithms in Section III, where we also derive the gradients of the loss with respect to delays and other parameters. In Section IV, we provide examples of using TrTDNNs to learn the dynamics of autonomous systems with multiple delays and the longitudinal dynamics of a CAV. We conclude our results and provide the future research directions in Section V.

II. Problem Statement

We consider time delay systems in continuous time, which can be cast in the general form

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

where $x \in \mathbb{R}^n$ and $x_t \in C([-\tau_{\max}, 0], \mathbb{R}^n)$ represent the space of continuous functions. Namely, we define $x_t(\vartheta) := x(t+\vartheta)$, $\vartheta \in [-\tau_{\max}, 0]$, where $\tau_{\max} > 0$ is the maximum value of the delay. Similarly, for the control input, we have $u \in \mathbb{R}^p$, $u_t \in C([-\tau_{\max}, 0], \mathbb{R}^p)$, and $u_t(\vartheta) := u(t+\vartheta)$, $\vartheta \in [-\tau_{\max}, 0]$. That is, the right-hand side is given by the functional $\mathcal{G} : C([-\tau_{\max}, 0], \mathbb{R}^n) \times C([-\tau_{\max}, 0], \mathbb{R}^p) \to \mathbb{R}^n$. Specifically, we restrict ourselves to point delays, that is, to cases when (1) can be written as follows:

$$\begin{aligned}\dot{x}(t) = g(&x(t-\tau_0), x(t-\tau_1), \ldots, u(t-\tau_d),\\ &u(t-\sigma_0), u(t-\sigma_1), \ldots, u\big(t-\sigma_q\big)\big)\end{aligned} \tag{2}$$

where $g : \mathbb{R}^n \times \mathbb{R}^n \times \cdots \times \mathbb{R}^n \times \mathbb{R}^p \times \mathbb{R}^p \times \cdots \times \mathbb{R}^p \to \mathbb{R}^n$ and $\tau_0 = \sigma_0 = 0$, while d and q represent the number of (nonzero) delays in the state and the input, respectively. Our goal here is to learn the nonlinear functional $\mathcal{G}$ or, equivalently, learn the nonlinear function g and the delays $\tau_0, \tau_1, \ldots, \tau_d$ and $\sigma_0, \sigma_1, \ldots, \sigma_q$, simultaneously.

Learning the functional $\mathcal{G}$ in (1) means finding a functional $\hat{\mathcal{G}}$, such that the norm $\|\mathcal{G}(x_t, u_t) - \hat{\mathcal{G}}(x_t, u_t)\|$ remains small while considering the set of states $\mathcal{X} \subset C([-\tau_{\max}, 0], \mathbb{R}^n)$ and set of inputs $\mathcal{U} \subset C([-\tau_{\max}, 0], \mathbb{R}^p)$. These sets will be specified with the available data as described further below. For the specific form (2), we shall find a function $\hat{g}$, such that the norm $\|g(\cdot) - \hat{g}(\cdot)\|$ remains small for the set of arguments specified by the data, and also find time delays $\hat{\tau}_0, \hat{\tau}_1, \ldots, \hat{\tau}_d$ and $\hat{\sigma}_0, \hat{\sigma}_1, \ldots, \hat{\sigma}_q$, which are close to the true delay values. In [51], a method was proposed to identify the time delay and the plant's parameters for a linear system using transfer function estimation. In this article, we will obtain the nonlinear function $\hat{g}$ and the delays $\hat{\tau}_0, \hat{\tau}_1, \ldots, \hat{\tau}_d$ and $\hat{\sigma}_0, \hat{\sigma}_1, \ldots, \hat{\sigma}_q$ directly from states and inputs.

In order to achieve this, we construct TrTDNNs. The inputs of such a network are the current and past values of the state and the control input, and the output of the network is the state derivative. Different from conventional TDNN, the time delays in the TrTDNN are continuous variables that evolve through the learning process. The proposed training algorithm is based on gradient information, which can utilize many advanced gradient-based methods. The neural networks, the loss function, and the training algorithm used will be introduced in Section III.

The learning process is executed while interacting with data. We assume that we observe the state and the input at time

moments $t = j\Delta t$, $j \in \mathbb{Z}$. In this case, the state derivative can also be approximated from data using finite differences. For example, using Euler method's, the state derivative can be obtained as follows:

$$\dot{x}(j\Delta t) \approx \frac{x((j+1)\Delta t) - x(j\Delta t)}{\Delta t} \tag{3}$$

but one may also use more sophisticated methods to obtain the derivative.

Since the delays vary continuously during the learning process, the value $x(t-\tau)$ may not be available directly. (Here, dropped the subscript of τ to simplify the notation.) Recall that the state is only observed at the time moments $t = j\Delta t$, $j \in \mathbb{Z}$. Considering that $\tau \in [l\Delta t, (l+1)\Delta t)$, we may define $\alpha \in [0, 1)$, such that $\tau = (l+\alpha)\Delta t$. Then, the delayed state can be approximated using linear interpolation

$$x(t-\tau) \approx (1-\alpha)x(t-l\Delta t) + \alpha x(t-(l+1)\Delta t) \tag{4}$$

and similar procedure can be followed for the input yielding

$$u(t-\sigma) \approx (1-\beta)u(t-l\Delta t) + \beta u(t-(l+1)\Delta t) \tag{5}$$

where $\sigma = (l+\beta)\Delta t$ and $\beta \in [0, 1)$.

After the training is complete, one may test the accuracy of the learned nonlinearity and delays by either comparing $\dot{\hat{x}}$ with $\dot{x}$ or comparing the solutions $\hat{x}(t)$ with $x(t)$, obtained via numerical simulation, along some time horizon. To further investigate how well the learned dynamics matches the ground truth, one may study the stability properties. For example, linearizing the dynamics (2) around an equilibrium, we obtain

$$\dot{x}(t) = \sum_{k=0}^{d} A_k x(t-\tau_k) + \sum_{k=0}^{q} B_k u(t-\sigma_k) \tag{6}$$

where $A_k \in \mathbb{R}^{n\times n}$ and $B_k \in \mathbb{R}^{n\times p}$, and we used $g(0, \ldots, 0) = 0$ to simplify the notation. This results in the characteristic equation

$$\det\left[\lambda I - \sum_{k=0}^{d} A_k e^{-\lambda\tau_k}\right] = 0 \tag{7}$$

see [30], which has infinitely many solution for the characteristic roots λ (see [32], [34]). Computing the rightmost characteristic roots allows us to compare the asymptotic behavior of the learned system with that of the ground truth.

III. TDNNs With Trainable Delays

In this section, we first review conventional TDNNs and point out their shortcomings when they are used to learn the delayed dynamics in continuous time. Then, we introduce novel TrTDNNs and highlight how they may overcome the shortcomings of conventional TDNNs. These are followed by the descriptions of the loss function and the training algorithms.

A. Time Delay Neural Networks

Time delays have been introduced into neural networks for speech recognition in [52]. The corresponding TDNNs are essentially feedforward neural networks with delayed information flow. Time delay networks with input delays can be expressed as follows:

$$\begin{aligned} z_1^t &= f_1\left(\sum_{k=0}^{r} w_1^k z_0^{t-k} + b_1\right) \\ z_l^t &= f_l\left(W_l z_{l-1}^t + b_l\right), \qquad l = 2, \ldots, L \end{aligned} \tag{8}$$

where z_l^t represents the (potentially vector valued) output of layer l at time t; $f_l(\cdot)$ is the activation function, which is applied elementwise and has well-defined derivative $f_l'(s) = (\partial f_L(s)/\partial s)$; and w_1^k is the weight matrix associated with delay index k, while W_l and b_l are the weight and bias used in layer l. Define the augmented vector

$$\mathbf{z}_0^t = \begin{bmatrix} z_0^t \\ z_0^{t-1} \\ \vdots \\ z_0^{t-r} \end{bmatrix}. \tag{9}$$

Equation (8) can be rewritten as follows:

$$\begin{aligned} s_1^t &= W_1 \mathbf{z}_0^t + b_1 \\ z_1^t &= f_1\left(s_1^t\right) \\ s_l^t &= W_l z_{l-1}^t + b_l \\ z_l^t &= f_l\left(s_l^t\right), \quad l = 2, \ldots, L \end{aligned} \tag{10}$$

with $W_1 = \left[w_1^0, w_1^1, \ldots, w_1^r\right]$. Here, we introduce the compact notation

$$z_L^t = \mathrm{net}\left(\mathbf{z}_0^t\right) \tag{11}$$

which captures the functional relationship between the input $\mathbf{z}_0^t$ and the output z_L^t of the network and can be utilized for both traditional TDNNs and the proposed TrTDNNs.

The time history of the state x and the control input u is often referred to as memory. These are usually available in discrete time, and the output can be very general, such as letters or words. Since our goal is to represent the dynamics of the dynamical system (2), we can discretize time and assume that the delays coincide with the discrete time steps $j\Delta t$, $j \in \mathbb{Z}$, where data are available. Then, one may construct

$$\mathbf{z}_0^t = \begin{bmatrix} x(t) \\ u(t) \\ x(t-\Delta t) \\ u(t-\Delta t) \\ \vdots \\ x(t-r\Delta t) \\ u(t-r\Delta t) \end{bmatrix} \tag{12}$$

where $\tau_{\max} = r\Delta t$. This approach, however, leads to computational issues. Namely, considering small Δt values results in a large weight matrix W_1, which may not be necessary for the system with discrete point delay, since many of these weights shall approach zero during training, which requires

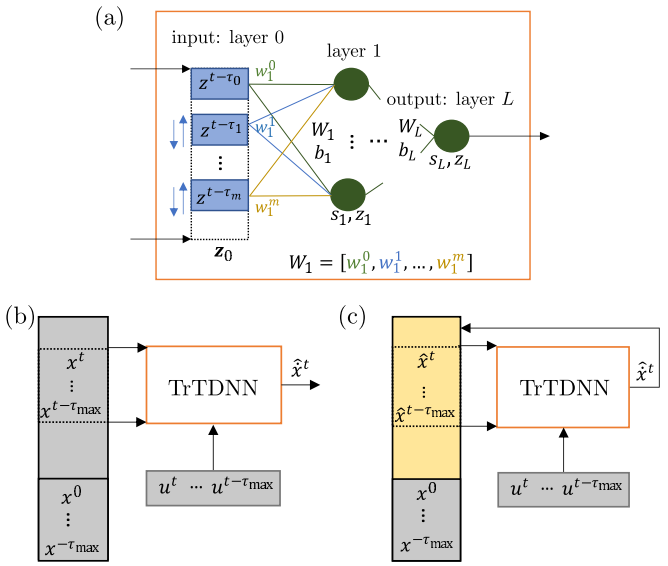

Fig. 2. (a) TrTDNN for predicting state derivative at time t. (b) Use of TrTDNN as feedforward mapping during training. (c) Use of TrTDNN for simulation: gray color means inputs from data, and the yellow color means the inputs obtained from simulation.

extra regularization. At the same time, simulating the system with the large number of delays is also computational expensive. Instead, we propose a new architecture, which can accommodate trainable time delays as laid out in Section III-B.

B. Constructing TrTDNNs

The concept of TrTDNN is shown in Fig. 2(a). There is still a window of sequential data available to the network at each time moment t, but the trainable delay parameters $\tau_0, \tau_1, \ldots \tau_m$ "select" the relevant ones as the network input. As these time delays evolve during the training, the inputs change as indicated by the blue arrows. After training, similar to the weights and biases, the time delays are fixed when using the network to predict future states of the dynamical system.

The TrTDNN also takes the form (10), but how the input $\mathbf{z}_0^t$ is constructed in the two cases differs. TDNNs utilize the whole memory (9), while for TrTDNNs, we construct the sparse memory

$$\mathbf{z}_0^t = \begin{bmatrix} z_0^{t-\tau_0} \\ z_0^{t-\tau_1} \\ \vdots \\ z_0^{t-\tau_m} \end{bmatrix} \tag{13}$$

with flexible delays, which will be learned from data. By constructing such flexible delayed inputs, the number of delays in TrTDNNs can be much smaller than the number of discretization points ($m \ll r$). This allows us to greatly simplify the network structure and customize using prior knowledge, which also improves the interpretability of the results. The arrangement of the delayed states and control inputs is also flexible, i.e., the delays do not have to be in an ascending/descending order, they can be assigned as zero, and their number can be overestimated. For instance, when learning the dynamics of (2), one may construct the input of the TrTDNN as follows:

$$\mathbf{z}_0^t = \begin{bmatrix} x\left(t-\hat{\tau}_0\right) \\ x\left(t-\hat{\tau}_1\right) \\ \vdots \\ x\left(t-\hat{\tau}_d\right) \\ u\left(t-\hat{\sigma}_0\right) \\ u\left(t-\hat{\sigma}_1\right) \\ \vdots \\ u\left(t-\hat{\sigma}_q\right) \end{bmatrix} \tag{14}$$

where the delays $\hat{\tau}_k$ and $\hat{\sigma}_k$ are continuous variables learned from the data. As discussed above, the state x and the control input u are typically available at time moments $j\Delta t$, $j \in \mathbb{Z}$. Thus, to construct the elements of (14), we will utilize (4) and (5).

We remark that the form (11) can be viewed as a recurrent neural network if the output of the network is used as input in the next time moment. Moreover, depending on whether the output is the state derivative at current time or the state itself at next time moment, the network can be used to capture the dynamics of continuous-time or discrete-time dynamical systems. In this article, we consider the former one, that is, the outputs of the TDNNs and TrTDNNs are used as the state derivatives to resemble the delay differential equations in the continuous time. The derivative of the state can then be predicted using the following equation:

$$\hat{\dot{x}}(t) = \text{net}\left(\mathbf{z}_0^t\right). \tag{15}$$

Then, learning the dynamics of the time delay system (2) is equivalent to sequentially adjusting the weights W_l, $l = 1, 2, \ldots, L$, and biases b_l, $l = 1, 2, \ldots, L$, in net as well as the delays $\hat{\tau}_k$, $k = 0, 1, \ldots, d$, and $\hat{\sigma}_k$, $k = 0, 1, \ldots, q$, such that the error $\|\hat{\dot{x}}(t) - \dot{x}(t)\|$ decreases.

C. Loss Function

The learning is performed by utilizing trajectory data generated by (2). Since the data are only available at discrete time steps $t = j\Delta t$, one can only evaluate the difference $\|\hat{\dot{x}}(t) - \dot{x}(t)\|$ at these time moments. Thus, we design the loss function using the mean-squared error between the predicted derivative $\hat{\dot{x}}(j\Delta t) = z_L^j$ and the derivative $\dot{x}(j\Delta t)$ obtained from data

$$\mathcal{L} = \frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{n}\left(\hat{\dot{x}}_i(t_0 + j\Delta t) - \dot{x}_i(t_0 + j\Delta t)\right)^2. \tag{16}$$

Observe that the prediction starts from $t_0 \geq \tau_{\max}$, because we need an initial segment of data of length $\tau_{\max}$. Here, x_i refers to the ith element of vector x, while the time derivative $\dot{x}(t_0 + j\Delta t)$ may be available as data or can be calculated from $x(t_0 + j\Delta t)$ via numerical differentiation; see (3).

We remark that one may alternatively define the simulation loss

$$\mathcal{L}_{\text{sim}} = \frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{n}\left(\hat{x}_i(t_0 + j\Delta t) - x_i(t_0 + j\Delta t)\right)^2 \tag{17}$$

which is used below to evaluate the performance of the trained networks. Different choices of the error measures and corresponding loss functions are studied in [48] and [53].

In order to demonstrate the performance of the proposed networks, we use a TrTDNN with sparse trainable input delays and a conventional TDNN with the dense input delays to carry out the same learning task. We train the networks using the loss function (16). While training the networks, the state x and control input u are taken from data, that is, the networks are trained using (15) in a feedforward fashion, as illustrated in Fig. 2(b). After training, we compare the performance of the two approaches using the simulation loss (17), as shown in Fig. 2(c). For the network simulation, only the initial history of the state x_0 and the control input $u(t)$ are given, while the state is generated using

$$\hat{x}(t) = \text{DDEsolver}(\text{net}, x_0, u, t). \tag{18}$$

In other words, in (15), the input of the net at each time t is constructed based on the ongoing simulation $\hat{x}(t)$ instead of the data $x(t)$.

D. Training Methods

Here, we illustrate the analytical derivation of the gradients of the loss function $\mathcal{L}$, so that the gradient-based methods, such as the gradient descent (GD) method or Levenberg–Marquardt (LM) method [54], can be applied. In our training algorithms, time delays are also treated as parameters, similar to weights and biases. Consequently, we need to calculate the gradients with respect to the delays too.

In the GD method, the updating formula of parameter $\theta \in \mathbb{R}^M$ at iteration $k+1$ uses the gradient of the loss function at iteration k

$$\theta_{k+1} = \theta_k - \eta_\theta \frac{\partial \mathcal{L}}{\partial \theta_k}. \tag{19}$$

The learning rate η_θ can be a constant or be adaptive during the training. In order to calculate the gradient $(\partial \mathcal{L}/\partial \theta_k)$, we reformulate the cost function (16) as follows:

$$\mathcal{L}(\theta) = \frac{1}{N} E^\top(\theta) E(\theta) \tag{20}$$

where

$$E(\theta) = \begin{bmatrix} e^1 \\ e^2 \\ \vdots \\ e^N \end{bmatrix} \in \mathbb{R}^{Nn} \tag{21}$$

and

$$e^j = \hat{\dot{x}}(t_0 + j\Delta t) - \dot{x}(t_0 + j\Delta t). \tag{22}$$

Then, the updating formula (19) becomes

$$\theta_{k+1} = \theta_k - \eta_\theta \frac{2}{N} \frac{\partial E^\top}{\partial \theta_k} E(\theta_k) \tag{23}$$

which contains the Jacobian matrix $(\partial E^\top/\partial \theta) \in \mathbb{R}^{M\times(Nn)}$.

For the LM method, we use the formula

$$\theta_{k+1} = \theta_k - \left(\frac{2}{N} \frac{\partial E^\top}{\partial \theta_k} \frac{\partial E}{\partial \theta_k} + \mu I \right)^{-1} \frac{2}{N} \frac{\partial E^\top}{\partial \theta_k} E(\theta_k) \tag{24}$$

where μ is the tuning parameter and I denotes the $M \times M$ identity matrix. For large μ, the LM method approximates the GD method with a small learning rate. For small μ, it approximates Newton's method with an approximate Hessian matrix $(\partial E^\top/\partial \theta)(\partial E/\partial \theta) \in \mathbb{R}^{M\times M}$.

Both algorithms (23) and (24) contain the Jacobian matrix

$$\frac{\partial E^\top}{\partial \theta} = \left[\left(\frac{\partial e^1}{\partial \theta} \right)^\top, \ldots, \left(\frac{\partial e^N}{\partial \theta} \right)^\top \right] \tag{25}$$

which requires the calculations of the gradients $(\partial e^j/\partial \theta)$, $j = 1, \ldots, N$. To simplify the notation, we drop the superscript j. Also, we vectorize the weight matrices $W_l \in \mathbb{R}^{n_l \times n_{l-1}}$ into the vectors

$$\overline{W}_l = \left[w_{1,1}, \ldots, w_{1,n_{l-1}}, w_{2,1}, \ldots, w_{2,n_{l-1}}, \cdots \right]^\top \in \mathbb{R}^{n_l n_{l-1}} \tag{26}$$

where n_l denotes the dimension of z_l. Then, the gradients with respect to weights $\overline{W}_l$ and biases b_l can be expressed as follows:

$$\frac{\partial e}{\partial \overline{W}_l} = \frac{\partial e}{\partial z_L} h_l \frac{\partial s_l}{\partial \overline{W}_l} = h_l \left(I_{n_l \times n_l} \otimes z_{l-1}^\top \right) \tag{27}$$

and

$$\frac{\partial e}{\partial b_l} = \frac{\partial e}{\partial z_L} h_l \frac{\partial s_l}{\partial b_l} = h_l \tag{28}$$

respectively, where

$$h_l = h_{l+1} \frac{\partial s_{l+1}}{\partial z_l} \frac{\partial z_l}{\partial s_l} = h_{l+1} W_{l+1} f_l'(s_l) \tag{29}$$

for $l = 1, \ldots, L-1$, and

$$h_L = f_L'(s_L) = I_{n_L \times n_L} \tag{30}$$

see (10). Note that $\otimes$ denotes the Kronecker product, so that $(I_{n_l \times n_l} \otimes z_{l-1}^\top) \in \mathbb{R}^{n_l \times n_l n_{l-1}}$, $h_l \in \mathbb{R}^{n_L \times n_l}$; therefore $(\partial e/\partial \overline{W}_l) \in \mathbb{R}^{n_L \times n_l n_{l-1}}$, and $(\partial e/\partial b_l) \in \mathbb{R}^{n_L \times n_l}$. In our case, the dimension of the last layer is equal to the dimension of the state derivative $\dot{x}$, i.e., $n_L = n$.

In case of the time delay parameters, we define the vector

$$T = \left[\ \hat{\tau}_0, \ldots, \hat{\tau}_d, \hat{\sigma}_0, \ldots, \hat{\sigma}_q \ \right]^\top \in \mathbb{R}^D \tag{31}$$

with $D = d + q + 2$, which is part of the parameter θ. The gradient with respect to the delay vector T can be calculated as follows:

$$\frac{\partial e}{\partial T} = \frac{\partial e}{\partial z_L} h_1 \frac{\partial s_1}{\partial \mathbf{z}_0} \frac{\partial \mathbf{z}_0}{\partial T} = h_1 W_1 \frac{\partial \mathbf{z}_0}{\partial T}. \tag{32}$$

Here, the network input $\mathbf{z}_0 = [\ z_{0,1}, \ldots, z_{0,n_0}\ ]^\top \in \mathbb{R}^{n_0}$ is an augmented vector of states and control inputs corresponding to each delay value, where $n_0 = D(n+p)$, and n and p are the dimension of the state x and the input u, respectively, as defined in Section II. The matrix $(\partial \mathbf{z}_0/\partial T) \in \mathbb{R}^{n_0 \times D}$ can be written as follows:

$$\frac{\partial \mathbf{z}_0}{\partial T} = \left[\frac{\partial \mathbf{z}_0}{\partial \tau_1}, \ldots, \frac{\partial \mathbf{z}_0}{\partial \tau_D} \right]. \tag{33}$$

Algorithm 1 Multi-Attempt Gradient-Based Training Algorithm for a TrTDNN

Data: training data and validation data
Result: learn $\theta = \{W_l, b_l, T\}, l = 1, \ldots, L$ from data
- normalize inputs and outputs between [0, 1] and obtain the derivatives of the inputs
- choose the updating formulas (23) or (24) and tune parameters η_θ or μ
- set maximum iteration number $k_{\max}$, maximum violation number $v_{\max}$, maximum allowed delay $\tau_{\max}$
- choose different initializations for W_l uniformly between $[-1, 1]$, set $b_l = 0$, and initialize the elements of T within interval $[0, \overline{\tau}]$ for $\overline{\tau} < \tau_{\max}$

For each initialization, set $v = 0$ and start training
for $k = 1, \ldots, k_{\max}$ **do**
  construct the input $\mathbf{z}_0$ for selected trajectories based on T_k
  calculate the losses $\mathcal{L}_{\text{tr}}$ and $\mathcal{L}_{\text{va}}$ for training and validation data
  obtain the gradient information and update parameters $\theta_{k+1} \leftarrow \theta_k$
  if $\mathcal{L}_{\text{va}}$ increases consecutively for $v_{\max}$ times **or** any element of T_{k+1} exceeds $\tau_{\max}$ **then**
    break loop
  end
end
After collecting all parameters from multiple attempts at the end of training, choose the set of parameters W_l, b_l, T which gives the smallest $\mathcal{L}_{\text{tr}}$ among them

Each element $z_{0,i}, i = 1, \ldots, n_0$, is a scalar associated with one specific delay $\tau_b \in T, b = 1, \ldots, D$. Thus, we can write

$$\frac{\partial z_{0,i}}{\partial \tau_b} = \frac{\partial z_{0,i}(t-\tau_b)}{\partial \tau_b} = -\frac{\partial z_{0,i}(t-\tau_b)}{\partial t} = -\dot{z}_{0,i}(t-\tau_b) \tag{34}$$

where $\dot{z}_{0,i}(t-\tau_b)$ is the time derivative of the network input at time $t-\tau_b$, which can be approximated from the data. For instance, the simplest approximation is given by using a zero-order-hold approximation and the forward Euler method

$$\dot{z}_{0,i}(t-\tau_b) \approx \big(z_{0,i}(t-(j-1)\Delta t) - z_{0,i}(t-j\Delta t)\big)/\Delta t \tag{35}$$

for $\tau_b \in [(j-1)\Delta t, \Delta t)$. We can set $(\partial z_{0,i}/\partial \tau_b) = 0$ if we regulate the network to have a delay-free input $x(t)$ or $u(t)$. This will be the case in Section IV-B. We also remark that to ensure the nonnegativeness of time delays, after using the updates (23) or (24), we apply $\max\{0, T_{k+1}\}$ elementwise; see (31).

Recall that in the formulas above, the parameter vector θ collects vectorized weights and biases and the time delays. Therefore, we obtain

$$\frac{\partial e}{\partial \theta} = \left[\frac{\partial e}{\partial \overline{W}_1}, \frac{\partial e}{\partial b_1}, \cdots, \frac{\partial e}{\partial \overline{W}_L}, \frac{\partial e}{\partial b_L}, \frac{\partial e}{\partial T}\right] \in \mathbb{R}^{n_L \times M} \tag{36}$$

and concatenate the e^j-s of different time steps as in (25) to construct the Jacobian $(\partial E^\top/\partial\theta) \in \mathbb{R}^{M\times(Nn_L)}$ with $n_L = n$.

We describe the training process in Algorithm 1 when using a dataset, which consists of multiple input–output trajectories. For autonomous systems, since there is no external excitation (control input), the validation dataset contains trajectories created by different histories, so that it is independent of the training dataset. The validation dataset is not used for updating parameters but for monitoring the training. In the case where only one trajectory is available, parts of the trajectory are selected as validation or testing, and those parts do not overlap with each other. In Algorithm 1, the input $\mathbf{z}_0$ of each trajectory is constructed based on learned delays T [see (31)] at each iteration, and the loss of multiple trajectories is combined for updating the parameters. One can also choose a trajectory randomly from training set and use that to update the parameters. Then, the GD algorithm becomes the stochastic GD method. In addition, we use multiple initializations to avoid undesired local minima.

IV. Applications

In this section, we demonstrate that TrTDNNs are capable of learning multiple time delays in autonomous systems from simulation data. We also show that they can also be used to identify the time delay from noisy field data.

A. Learning From Simulation Data

We consider the following nonlinear autonomous systems with multiple delays:

$$\dot{x}(t) = -x(t) + x(t-1)x(t-0.5) - x^3(t-0.5) \tag{37}$$

and

$$\dot{x}(t) = -x(t-1.5) + x^2(t-1) - x^3(t-0.5). \tag{38}$$

In system (37), there are two delays (not including the delay-free term) and a "cross term" containing both delays. In system (38), there are three delays and no delay-free term. These systems are constructed for testing the proposed algorithm while learning multiple time delays. In both cases, the same network architecture (one-hidden-layer TrTDNN with three input delays and only three neurons in the hidden layer) is used to learn the dynamics from simulation data. We use the sigmoid function

$$f(z) = \frac{1}{1+e^{-z}} \tag{39}$$

as the activation function in the hidden layer and a linear function in the output layer.

The training dataset contains seven trajectories created by numerical simulation using the constant histories $x(t) \equiv c$, $t \in [-\tau_{\max}, 0]$, where $c = \{1, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6\}$. The simulated trajectory with constant history $c = 1.25$ is used for validation, and we test the performance on the trajectory with constant history $c = 1.35$. The time step in all datasets is $\Delta t = 0.01$. The initial time delays are chosen between 0 and $\tau_{\max} = 2$. We implemented both the GD and LM methods, but the LM method converges much faster. Thus, the results are presented here for LM method with $\mu = 0.01$. During training, we start from ten different initializations and choose the best result from them as our final trained results. We show below that more attempts in training provide better convergence in the time delay. As a trade-off, having more attempts makes the training process slower.

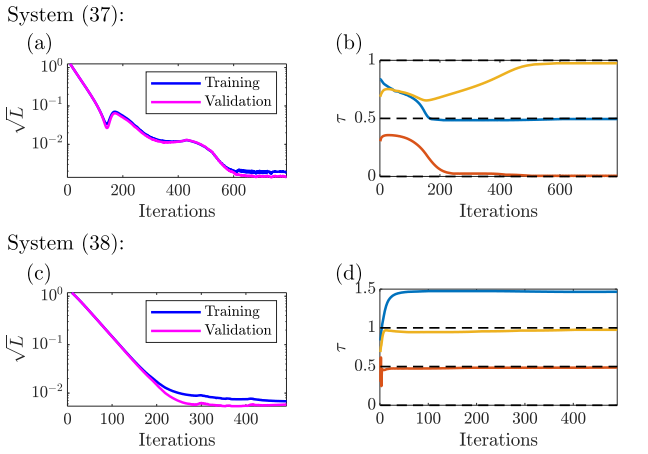

Fig. 3. Normalized RMSE and learned delays along training iterations for (a) and (b) system (37) and (c) and (d) system (38). While only first 700 iterations are shown in the figure, the actual training stops later based on the validation loss criterion.

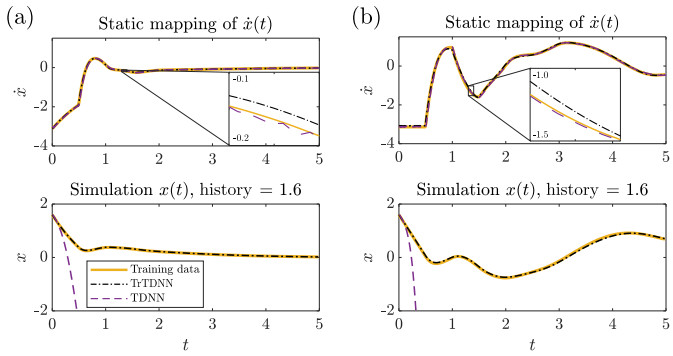

Fig. 4. Performance of TrTDNN and conventional TDNN on the training data for (a) system (37) and (b) system (38).

The training processes for systems (37) and (38) are shown in Fig. 3. Fig. 3(a) and (c) shows that the loss decreases rapidly within a few hundred iterations for both systems. Fig. 3(b) and (c) depicts the learning paths of the time delay parameters. Observe that the delays converge to the true values, i.e., to 0, 0.5, and 1 in case of (37) and to 0.5, 1, and 1.5 in case of (38). To highlight the performance of the trained TrTDNN, we display the simulation results of the trained networks in Figs. 4 and 5 for training data and testing data, respectively. Static mapping refers to the input–output mapping at each time step, where the inputs are always from the data; see (15) and Fig. 2(b). Simulation refers the network as a delay differential equation simulated with given initial history; see (18) and Fig. 2(c).

A conventional one-hidden-layer TDNN is also trained using MATLAB Deep Learning Toolbox with LM method for comparison. The conventional TDNN also has three neurons in the hidden layer and the same nonlinear activation function (39), but with 201 inputs, i.e., $[x(t),\ x(t-0.01),\ \ldots, x(t-2)]$, compared with the three inputs with three trainable delays in the TrTDNN. The quantitative analysis of the training and testing performances is collected in Table I. Since the conventional TDNN has a large number of inputs, the complexity of the network is much larger, even though the number of neurons in the hidden layer is the same as for the TrTDNN. Thus, the conventional TDNN can achieve

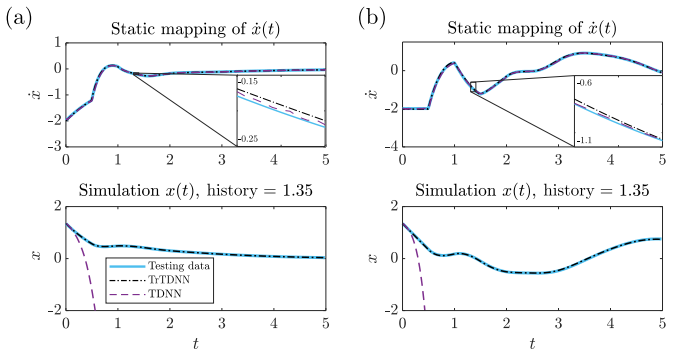

Fig. 5. Performance of TrTDNN and conventional TDNN on the test data for (a) system (37) and (b) system (38).

TABLE I
TRAINING AND TESTING RESULTS FOR THE TWO SYSTEMS

RMSE		System (37)		System (38)	
		TDNN	TrTDNN	TDNN	TrTDNN
Training ($c = 1.6$)	$\dot{x}$	**0.0081**	0.0177	**0.0176**	0.0381
	x	26.2978	**0.0069**	31.8429	**0.0266**
Testing ($c = 1.35$)	$\dot{x}$	**0.0046**	0.0071	**0.0115**	0.0183
	x	25.6948	**0.0029**	28.8596	**0.0095**

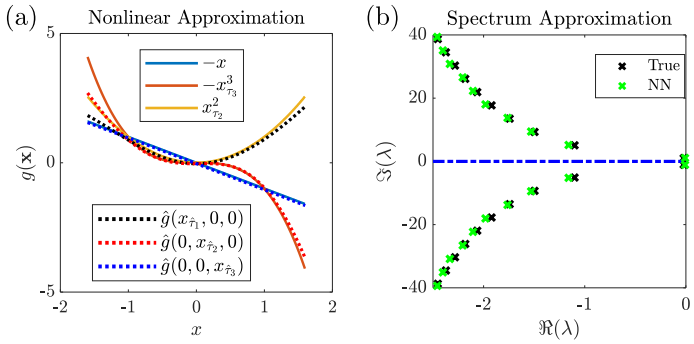

Fig. 6. (a) Approximation of the nonlinearities in (38). Solid curves represent the ground truth, while the dashed curves are extracted from the TrTDNN. (b) Approximation of the spectrum in (38). Black crosses show the ground truth, while the green crosses are from the trained network.

a better fitting in terms of static mapping, i.e., capture $\dot{x}$ better. However, when the networks are used for simulation, the conventional TDNN fails to capture x due to overfitting, while the TrTDNN is able to reproduce it with high accuracy. This highlights the advantages of the sparsity and trainability of the proposed TrTDNN.

To emphasize that the trained TrTDNN can capture the true dynamics, we also demonstrate that it reproduces the nonlinearities in the system as well as the spectrum near equilibria. Since the delayed components in system (38) are decoupled, we plot them separately as shown by the solid curves in Fig. 6(a). Also, by activating each delayed channel in the network inputs, we can plot the nonlinear approximation given by the trained network, as shown by the dashed curves. Thus, with sigmoid function (39) and only three neurons in the hidden layer, the TrTDNN can simultaneously learn the delays and nonlinearities. In order to check the transient dynamics around the equilibrium $x(t) \equiv 0$ in system (38), we compare the spectrum, i.e., the eigenvalues of the trained network and that of the original system in Fig. 6(b). The spectrum given by

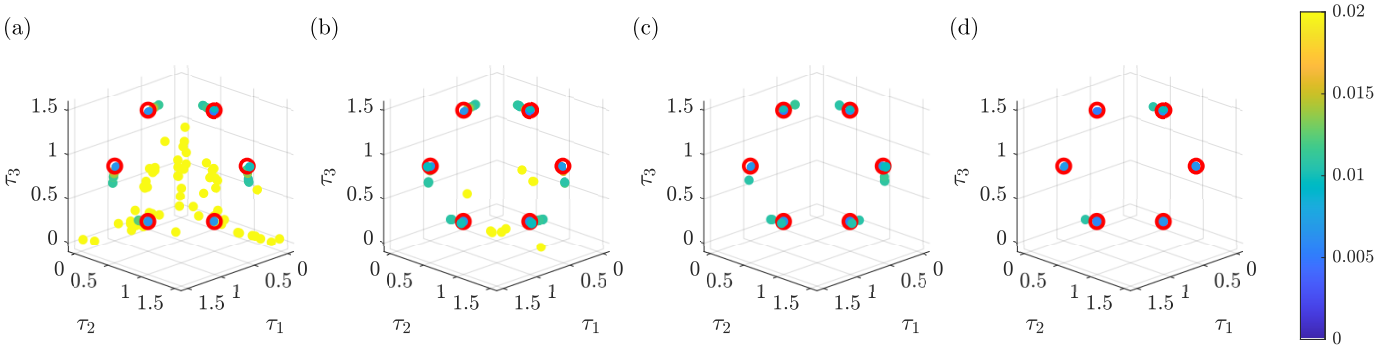

Fig. 7. Convergence of learned delays in 100 runs. (a) One attempt. (b) Five attempts. (c) Ten attempts. (d) 20 attempts.

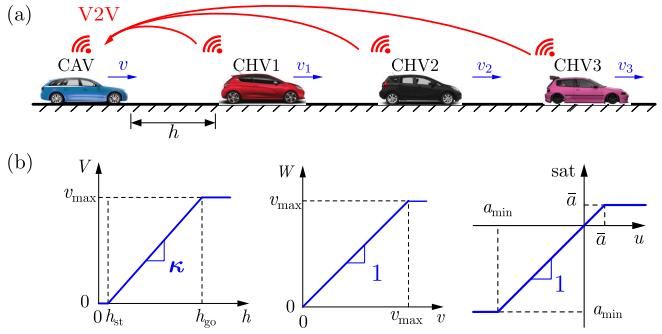

Fig. 8. (a) Experimental setup where a CAV follows CHVs based on the data transmitted through V2V connectivity. (b) Nonlinearities used in the model.

the trained network is very close to the ground truth, which are the solutions to $\lambda + e^{-1.5\lambda} = 0$. That is, the TrTDNNs approximates well the underlying transient dynamics.

To illustrate the necessity of using multiple attempts in the gradient-based training algorithms, we show the final learned time delays with different numbers of attempts in each training in Fig. 7 for system (38). The 100 runs of training are performed, and the final delays are shown in the 3-D plot, while the color bar shows the final training error. Fig. 7(a) shows that introducing the additional trainable delay parameters into the networks leads to many undesired local minima. As we increase the number of attempts, we explore more places in the parameter space, and some undesired local minima can be filtered out by choosing the best path. We can see that with ten attempts in the training process, the delays converge to the true values. Multiple initializations in the training help to search globally. At the same time, the gradient information obtained from approximate state derivatives helps in local exploitation.

B. Learning From Field Data

Here, we provide an example of learning the delay and nonlinearities from field data. In particular, we use a TrTDNN to learn the longitudinal dynamics of a CAV from experimental data. The scenario is shown in Fig. 8(a) where a CAV is inserted to a chain of connected human-driven vehicles (CHVs). Via vehicle-to-vehicle (V2V) communication, CAV is made aware of the velocities and positions of the vehicles ahead, and it also measures its own velocity and position.

The details of the controller design and experiments are given in [55]. Here, we provide a brief overview of the design procedure, while our main goal is to learn the dynamics form data. The longitudinal dynamics of the CAV can be modeled by the following equation:

$$\begin{aligned} \dot{h}(t) &= v_1(t) - v(t) \\ \dot{v}(t) &= -p(v(t)) + \text{sat}(u(t-\tau)) \end{aligned} \tag{40}$$

where h is the distance headway to the CHV1 immediately ahead, v is the speed of the CAV, and v_1 is the speed of the CHV1. Moreover, $p(v(t))$ represents the loss term, which depends on CAV's current velocity, while u represents the commanded acceleration that is limited by the saturation function

$$\text{sat}(u) = \begin{cases} a_{\min}, & \text{if} \quad u < a_{\min} \\ u, & \text{if} \quad a_{\min} \le u \le a_{\max} \\ a_{\max}, & \text{if} \quad u > a_{\max} \end{cases} \tag{41}$$

to keep it within the limits given by the road friction and driver comfort. Finally, τ represents the time delay in the control loop resulting from V2V communication, onboard computation, and actuator delays. Neither the resistance term $p(v)$ nor the time delay τ was known while designing the controller. Here, we will identify these from the data collected through the experiments.

The control law used in the experiments [55], [56], [57], [58] is given by the following equation:

$$u = \alpha(V(h) - v) + \sum_{k=1}^{3} \beta_k (W(v_k) - v) \tag{42}$$

where α is the control gain used to regulate the headway h, while β_1, β_2, and β_3 are the control gains used to respond to the velocities v_1, v_2, and v_3 of the CHVs ahead. The control law includes the range policy

$$V(h) = \begin{cases} 0, & \text{if} \quad h < h_{\text{st}} \\ \kappa(h - h_{\text{st}}), & \text{if} \quad h_{\text{st}} \le h \le h_{\text{go}} \\ v_{\max}, & \text{if} \quad h > h_{\text{go}} \end{cases} \tag{43}$$

which gives the desired speed as an affine function of the headway. It also contains the speed policy

$$W(v_k) = \begin{cases} v_k, & \text{if} \quad v_k \le v_{\max} \\ v_{\max}, & \text{if} \quad v_k > v_{\max} \end{cases} \tag{44}$$

which is introduced to avoid following leaders whose speed exceeds the maximum desired speed $v_{\max}$. The functions V, W, and sat are depicted in Fig. 8(b).

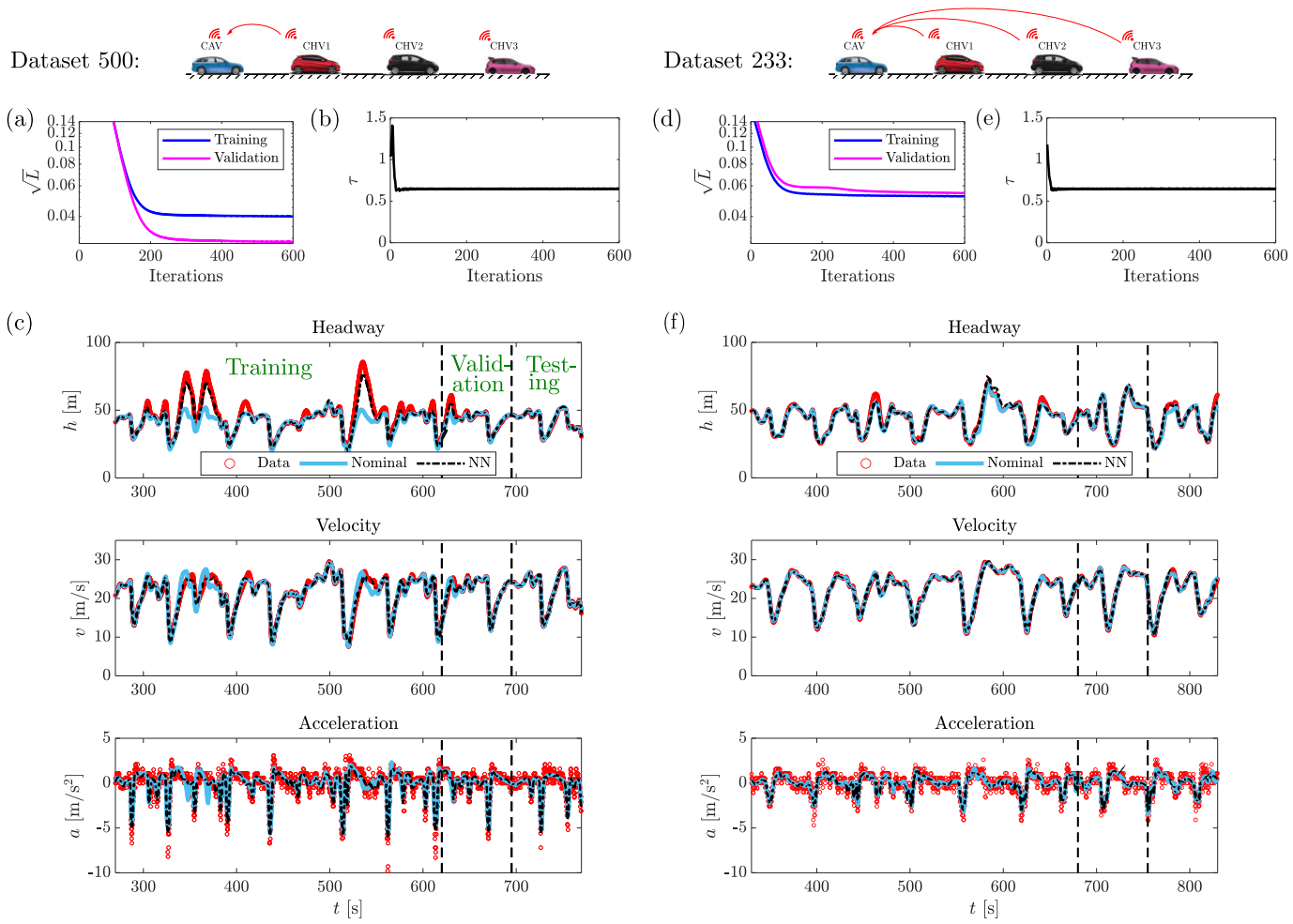

Fig. 9. TrTDNN training results on datasets 500 and 233 using LM algorithm. (a) and (d) Training and validation loss. (b) and (e) Evolution of the time delay. (c) and (f) Simulation of learned TrTDNN (black) together with the simulation given by the nominal model (light blue) and the experimental data (red). Note that (a), (b), (d), and (e) only show the first 600 iterations of training, but the actual training stops later based on the validation loss criterion. The RMSE of the quantities plotted in (c) and (f) is collected in Table II.

As shown in Fig. 9, we utilize two experimental datasets with different controller settings. In "dataset 500," the CAV only responded to the CHV immediately ahead using $(\beta_1, \beta_2, \beta_3) = (0.5, 0.0, 0.0)\ \mathrm{s}^{-1}$, while in "dataset 233," the CAV responded to all three CHVs ahead with $(\beta_1, \beta_2, \beta_3) = (0.2, 0.3, 0.3)\ \mathrm{s}^{-1}$. Both settings shared the same $\alpha = 0.4\,\mathrm{s}^{-1}$. When simulating the nominal model, we use $\tau = 0.6$ s and $p(v) = a + cv^2$ with $a = 0.0981\ \mathrm{m/s}^2$ and $c = 0.0003\ \mathrm{m}^{-1}$; see [50]. The delay and the loss term were not known in the experiments, and some unmodeled dynamics (e.g., powertrain) are not represented in the nominal model (40)–(42). This can be observed when comparing the experimental data (red) and the simulation of the nominal model (light blue) in Fig. 9(c) and (f). Our goal here is to learn the overall dynamics of the CAV and the time delay τ simultaneously from the experimental data.

A TrTDNN with three neurons in the single hidden layer is used when learning the dynamics. The number of the inputs depends on the availability of the data and the number of the time delays in the systems. We presume that one dominant time delay appears in the feedback loop through the inputs $h(t-\tau)$, $v(t-\tau)$, $v_1(t-\tau)$, $v_2(t-\tau)$, $v_3(t-\tau)$, and we also set a time delay to zero corresponding to the loss term containing $v(t)$. There are altogether six inputs to the TrTDNN, five of which share the same trainable delay. The output of the TrTDNN is the acceleration $\dot{v}(t)$, since the kinematic part of (40) does not require further learning. The TrTDNNs are trained with LM methods with $\mu = 0.01$ and ten attempts. The initial delays for those attempts are chosen uniformly between 0 and 2 s, and the maximum allowed delay is set to 3 s.

Approximately, 500-s data (with a time step of 0.1 s) are utilized from each experiment. The first 70% was used for training, 15% was used for validation, and 15% was used for testing. The training processes and delay learning path are shown in Fig. 9(a) and (b) for dataset 500 and in Fig. 9(d) and (e) for dataset 233. In both cases, the delay value converges to a number around 0.65 s. In Fig. 9(c) and (f), the simulations from the trained TrTDNN are shown as black dotted-dashed lines capturing the experimental data (red circles) well. The simulation results of the nominal model are plotted as light blue solid curves. From Fig. 9(c), we can see that the trained TrTDNN can capture the headway and velocity well in the training segment and gives good predictions in validation and testing segments. The difference between the model-based simulations and the learned dynamics is more

TABLE II
TESTING RESULTS FOR THE TWO DATASETS

RMSE	Dataset 500		Dataset 233	
	TrTDNN	Nominal	TrTDNN	Nominal
h [m]	**2.7682**	7.9383	**1.6971**	1.7633
v [m/s]	**0.5865**	1.1765	**0.3680**	0.3884
$\dot{v}$ [m/s^2]	**0.6057**	0.6933	**0.4383**	0.4423

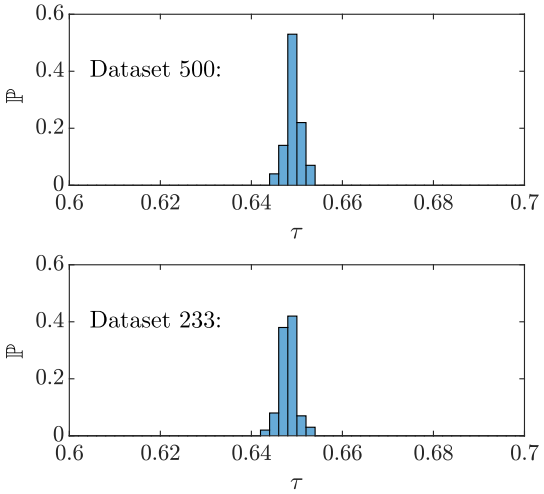

Fig. 10. Distributions of learned delay in 100 runs for the two datasets.

pronounced for dataset 500 than for dataset 233, since in the former case, the CAV performed more aggressive maneuvers. The simulation-based root-mean-square errors between prediction and data are collected in Table II for headway, velocity, and acceleration. To show the convergence in the time delay parameters, we run the training algorithm 100 times and plot the distribution of the final learned delay in Fig. 10. For both datasets, the delay converges to a distribution concentrated around 0.65 s.

V. CONCLUSION

In this article, we established a connection between time delay systems to the TDNNs in a continuous-time setting. We proposed the concept of a TrTDNN, which enables one to learn the nonlinearities and the time delays simultaneously. We also proposed gradient-based learning algorithms based on the derivative loss function, which led to fast training and accurate network simulation results when using the trained neural network as the right-hand side of the delay differential equation. We showed that using a conventional TDNN resulted in overfitting and inaccurate network simulation result. Utilizing sparse trainable delays in the TrTDNN helped to learn the underlying mechanism, that is, to extract the time delays and nonlinearities from data. The gradient-based learning algorithms with multiple initializations for delay training were demonstrated for two cases: learning from simulation data and learning from experimental data collected by CAVs. As future directions, we will consider the case where the states are not fully observed. Additional networks may also be employed together with TrTDNN to extract the hidden states and decode hidden-state derivatives.

REFERENCES

[1] W. Zhao, W. Lin, R. Liu, and J. Ruan, "Asymptotical stability in discrete-time neural networks," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 49, no. 10, pp. 1516–1520, Oct. 2002.

[2] H. Zhang, Z. Wang, and D. Liu, "A comprehensive review of stability analysis of continuous-time recurrent neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 7, pp. 1229–1262, Jul. 2014.

[3] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Netw.*, vol. 1, no. 1, pp. 4–27, Mar. 1990.

[4] X.-D. Li, J. K. L. Ho, and T. W. S. Chow, "Approximation of dynamical time-variant systems by continuous-time recurrent neural networks," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 52, no. 10, pp. 656–660, Oct. 2005.

[5] J.-S. Pei, E. C. Mai, J. P. Wright, and S. F. Masri, "Mapping some basic functions and operations to multilayer feedforward neural networks for modeling nonlinear dynamical systems and beyond," *Nonlinear Dyn.*, vol. 71, nos. 1–2, pp. 371–399, Jan. 2013.

[6] S. L. Brunton and J. N. Kutz, *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge, U.K.: Cambridge Univ. Press, 2019.

[7] J. G. Kuschewski, S. Hui, and S. H. Zak, "Application of feedforward neural networks to dynamical system identification and control," *IEEE Trans. Control Syst. Technol.*, vol. 1, no. 1, pp. 37–49, Mar. 1993.

[8] R. Kumar and S. Srivastava, "Externally recurrent neural network based identification of dynamic systems using Lyapunov stability analysis," *ISA Trans.*, vol. 98, pp. 292–308, Mar. 2020.

[9] R. Kumar, "Memory recurrent Elman neural network-based identification of time-delayed nonlinear dynamical system," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 53, no. 2, pp. 753–762, Feb. 2023.

[10] D. Psaltis, A. Sideris, and A. A. Yamamura, "A multilayered neural network controller," *IEEE Control Syst. Mag.*, vol. 8, no. 2, pp. 17–21, Apr. 1988.

[11] R. Kumar, S. Srivastava, and J. R. P. Gupta, "Modeling and adaptive control of nonlinear dynamical systems using radial basis function network," *Soft Comput.*, vol. 21, no. 15, pp. 4447–4463, Aug. 2017.

[12] S. Wong, L. Jiang, R. Walters, T. G. Molnár, G. Orosz, and R. Yu, "Traffic forecasting using vehicle-to-vehicle communication," in *Proc. 3rd Conf. Learn. Dyn. Control*, vol. 211, 2021, pp. 917–929.

[13] R. Kumar, "A Lyapunov-stability-based context-layered recurrent pi-sigma neural network for the identification of nonlinear systems," *Appl. Soft Comput.*, vol. 122, Jun. 2022, Art. no. 108836.

[14] R. Kumar, "Double internal loop higher-order recurrent neural network-based adaptive control of the nonlinear dynamical system," *Soft Comput.*, vol. 27, no. 22, pp. 17313–17331, Nov. 2023.

[15] J. Li, Z. Wang, H. Dong, and G. Ghinea, "Outlier-resistant remote state estimation for recurrent neural networks with mixed time-delays," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 5, pp. 2266–2273, May 2021.

[16] A. Keane, B. Krauskopf, and C. M. Postlethwaite, "Climate models with delay differential equations," *Chaos, Interdiscipl. J. Nonlinear Sci.*, vol. 27, no. 11, Nov. 2017, Art. no. 114309.

[17] F. Casella, "Can the COVID-19 epidemic be controlled on the basis of daily test reports?" *IEEE Control Syst. Lett.*, vol. 5, no. 3, pp. 1079–1084, Jul. 2021.

[18] A. D. Ames, T. G. Molnár, A. W. Singletary, and G. Orosz, "Safety-critical control of active interventions for COVID-19 mitigation," *IEEE Access*, vol. 8, pp. 188454–188474, 2020.

[19] G. Orosz, R. E. Wilson, R. Szalai, and G. Stépán, "Exciting traffic jams: Nonlinear phenomena behind traffic jam formation on highways," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 80, no. 4, Oct. 2009, Art. no. 046205.

[20] M. di Bernardo, A. Salvi, and S. Santini, "Distributed consensus strategy for platooning of vehicles in the presence of time-varying heterogeneous communication delays," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 1, pp. 102–112, Feb. 2015.

[21] Y. Chen, D. Huang, N. Qin, and Y. Zhang, "Adaptive iterative learning control for a class of nonlinear strict-feedback systems with unknown state delays," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, pp. 6416–6427, 2023.

[22] R. Moghadam and S. Jagannathan, "Optimal adaptive control of uncertain nonlinear continuous-time systems with input and state delays," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 6, pp. 3195–3204, Jun. 2023.

[23] T. G. Molnar and T. Insperger, "On the robust stabilizability of unstable systems with feedback delay by finite spectrum assignment," *J. Vib. Control*, vol. 22, no. 3, pp. 649–661, Feb. 2016.
[24] S. Arik, "New criteria for stability of neutral-type neural networks with multiple time delays," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 5, pp. 1504–1513, May 2020.
[25] X. Zhang, D. Wang, K. Ota, M. Dong, and H. Li, "Delay-dependent switching approaches for stability analysis of two additive time-varying delay neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 12, pp. 7545–7558, Dec. 2022.
[26] Y. He, G. P. Liu, D. Rees, and M. Wu, "Stability analysis for neural networks with time-varying interval delay," *IEEE Trans. Neural Netw.*, vol. 18, no. 6, pp. 1850–1854, Nov. 2007.
[27] Z. Li, Y. Bai, C. Huang, H. Yan, and S. Mu, "Improved stability analysis for delayed neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 9, pp. 4535–4541, Sep. 2018.
[28] O. Diekmann, S. A. van Gils, S. M. V. Lunel, and H.-O. Walther, *Delay Equations*. Cham, Switzerland: Springer, 1995.
[29] K. Gu, V. L. Kharitonov, and J. Chen, *Stability of Time-Delay System*. Cham, Switzerland: Springer, 2003.
[30] W. Michiels and S.-I. Niculescu, *Stability and Stabilization of Time-delay Systems*. Philadelphia, PA, USA: SIAM, 2007.
[31] M. Krstic, *Delay Compensation for Nonlinear, Adaptive, and PDE Systems*. Basel, Switzerland: Birkhäuser, 2003.
[32] T. Insperger and G. Stépán, *Semi-Discretization for Time-Delay Systems*. Cham, Switzerland: Springer, 2011.
[33] E. Fridman, *Introduction to Time-Delay Systems*. Basel, Switzerland: Birkhäuser, 2014.
[34] D. Breda, S. Maset, and R. Vermiglio, *Stability of Linear Delay Differential Equations*. Cham, Switzerland: Springer, 2015.
[35] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2018. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf
[36] M. A. Cohen and S. Grossberg, "Absolute stability of global pattern formation and parallel memory storage by competitive neural networks," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, no. 5, pp. 815–826, Sep. 1983.
[37] K. Funahashi and Y. Nakamura, "Approximation of dynamical systems by continuous time recurrent neural networks," *Neural Netw.*, vol. 6, no. 6, pp. 801–806, 1993.
[38] T. W. S. Chow and X.-D. Li, "Modeling of continuous time dynamical systems with input by recurrent neural networks," *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl.*, vol. 47, no. 4, pp. 575–578, Apr. 2000.
[39] G. Orosz and G. Stépán, "Subcritical Hopf bifurcations in a car-following model with reaction-time delay," *Proc. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 462, no. 2073, pp. 2643–2670, Sep. 2006.
[40] T. G. Molnar, Z. Dombovari, T. Insperger, and G. Stepan, "On the analysis of the double Hopf bifurcation in machining processes via centre manifold reduction," *Proc. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 473, no. 2207, Nov. 2017, Art. no. 20170502.
[41] M. C. Mackey and L. Glass, "Oscillation and chaos in physiological control systems," *Science*, vol. 197, no. 4300, pp. 287–289, Jul. 1977.
[42] M. Goldmann, C. R. Mirasso, I. Fischer, and M. C. Soriano, "Learn one size to infer all: Exploiting translational symmetries in delay-dynamical and spatiotemporal systems using scalable neural networks," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 106, no. 4, Oct. 2022, Art. no. 044211.
[43] F. Stelzer, A. Röhm, R. Vicente, I. Fischer, and S. Yanchuk, "Deep neural networks using a single neuron: Folded-in-time architecture using feedback-modulated delay loops," *Nature Commun.*, vol. 12, no. 1, pp. 1–10, Aug. 2021.
[44] Q. Zhu, Y. Guo, and W. Lin, "Neural delay differential equations," 2021, *arXiv:2102.10801*.
[45] Q. Zhu, Y. Guo, and W. Lin, "Neural delay differential equations: System reconstruction and image classification," 2023, *arXiv:2304.05310*.
[46] J. de Jesus Rubio, W. Yu, and X. Li, "Time-delay nonlinear system modelling via delayed neural networks," in *Proc. 6th World Congr. Intell. Control Autom.*, 2006, pp. 119–123.
[47] J. D. J. Rubio and W. Yu, "Stability analysis of nonlinear system identification via delayed neural networks," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 54, no. 2, pp. 161–165, Feb. 2007.
[48] X. A. Ji and G. Orosz, "Learn from one and predict all: Single trajectory learning for time delay systems," *Nonlinear Dyn.*, vol. 112, no. 5, pp. 3505–3518, Mar. 2024, doi: 10.1007/s11071-023-09206-y.
[49] X. A. Ji, T. G. Molnár, S. S. Avedisov, and G. Orosz, "Feed-forward neural networks with trainable delay," in *Proc. 2nd Conf. Learn. Dyn. Control*, vol. 120, 2020, pp. 127–136.
[50] X. A. Ji, T. G. Molnár, S. S. Avedisov, and G. Orosz, "Learning the dynamics of time delay systems with trainable delays," in *Proc. 3rd Conf. Learn. Dyn. Control*, vol. 144, 2021, pp. 930–942.
[51] F. Chen, H. Garnier, and M. Gilson, "Robust identification of continuous-time models with arbitrary time-delay from irregularly sampled data," *J. Process Control*, vol. 25, pp. 19–27, Jan. 2015.
[52] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 37, no. 3, pp. 328–339, Mar. 1989.
[53] X. Ji and G. Orosz, "Learning the dynamics of autonomous nonlinear delay systems," in *Proc. 5th Conf. Learn. Dyn. Control*, 2023, pp. 116–127.
[54] J. J. Moré, "The Levenberg–Marquardt algorithm: Implementation and theory," in *Numerical Analysis*. Cham, Switzerland: Springer, 1978, pp. 105–116.
[55] J. I. Ge, S. S. Avedisov, C. R. He, W. B. Qin, M. Sadeghpour, and G. Orosz, "Experimental validation of connected automated vehicle design among human-driven vehicles," *Transp. Res. C, Emerg. Technol.*, vol. 91, pp. 335–352, Jun. 2018.
[56] L. Zhang and G. Orosz, "Motif-based design for connected vehicle systems in presence of heterogeneous connectivity structures and time delays," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 6, pp. 1638–1651, Jun. 2016.
[57] D. Hajdu, J. I. Ge, T. Insperger, and G. Orosz, "Robust design of connected cruise control among human-driven vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 2, pp. 749–761, Feb. 2020.
[58] S. S. Avedisov, G. Bansal, and G. Orosz, "Impacts of connected automated vehicles on freeway traffic patterns at different penetration levels," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 5, pp. 4305–4318, May 2022.

Xunbi A. Ji received the B.E. degree in energy and power engineering from Southeast University, Nanjing, China, in 2018, and the MSE degree in mechanical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2020, where she is currently pursuing the Ph.D. degree in mechanical engineering.

Her research interests include modeling time delay systems, nonlinear dynamics, and applications to connected and automated vehicles.

Gábor Orosz (Senior Member, IEEE) received the M.Sc. degree in engineering physics from Budapest University of Technology, Budapest, Hungary, in 2002, and the Ph.D. degree in engineering mathematics from the University of Bristol, Bristol, U.K., in 2006.

He held post-doctoral positions at the University of Exeter, Exeter, U.K., and the University of California, Santa Barbara, CA, USA. In 2010, he joined the University of Michigan, Ann Arbor, MI, USA, where he is currently an Associate Professor of mechanical engineering, and civil and environmental engineering. From 2017 to 2018, he was a Visiting Professor of control and dynamical systems at California Institute of Technology, Pasadena, CA, USA. In 2022, he was a Visiting Professor of applied mechanics at Budapest University of Technology. His research interests include nonlinear dynamics and control, time delay systems, machine learning, and data-driven systems with applications to connected and automated vehicles, traffic flow, and biological networks.