Proceedings of Machine Learning Research vol 144:1–13, 2021

Learning the Dynamics of Time Delay Systems with Trainable Delays

Xunbi A. Ji XUNBIJ@UMICH.EDU
Department of Mechanical Engineering, University of Michigan, Ann Arbor, MI 48109, USA
Tamás G. Molnár TMOLNAR@CALTECH.EDU
Department of Mechanical Engineering, University of Michigan, Ann Arbor, MI 48109, USA
Department of Mechanical and Civil Engineering, California Institute of Technology, Pasadena, CA 91125, USA
Sergei S. Avedisov AVEDISKA@UMICH.EDU
Department of Mechanical Engineering, University of Michigan, Ann Arbor, MI 48109, USA
Toyota Motor North America R&D-Infotech Labs, Mountain View, CA 94043, USA
Gábor Orosz OROSZ@UMICH.EDU
Department of Mechanical Engineering, University of Michigan, Ann Arbor, MI 48109, USA
Department of Civil and Environmental Engineering, University of Michigan, Ann Arbor, MI 48109,USA

Abstract

In this paper, we propose a delay learning algorithm for time delay neural networks (TDNNs) based on mini-batch gradient descent. We show that the proposed algorithm is suitable for learning the dynamics of nonlinear time delay systems using TDNNs with trainable delays. The delays are introduced in the input layer and are learned with the same approach as weights and biases. The learned delays are easy to interpret and they are not restricted to discrete values. We demonstrate the method with an example of learning the dynamics of an autonomous time delay system. We show the performance of two proposed network architectures with trainable delays and compare it to a standard TDNN which has a large number of fixed (non-trainable) input delays. We demonstrate that the networks with trainable input delays achieve significantly better performance in neural network simulations compared to the standard TDNN. We also highlight that undesired local minima may appear due to the delays in the networks.

Keywords: time delay system, time delay neural network, delay learning

1. Introduction

Neural networks belong to one of the most important branches of machine learning. Since they were proposed by Rosenblatt (1958) and further developed with back-propagation algorithm (Rumelhart et al., 1986; Van Ooyen and Nienhuis, 1992; Zhang et al., 2007), a large variety of neural networks have been designed for classification, regression, system identification, etc. Starting from the 1990's, there have been many applications of feed-forward neural networks (FNNs) and recurrent neural networks (RNNs) in dynamical systems and control (Kumpati and Kannan, 1990; Kuschewski et al., 1993). Feed-forward neural networks are concise and straight-forward for learning the dynamics of time-invariant systems due to their relatively simple architectures and ability to approximate the nonlinear relation between inputs and outputs. Recurrent neural networks, on the other hand, contain a kind of concentrated "memory", which makes them popular in learning dynamical systems; see, for example, the continuous time recurrent neural networks (CTRNNs) (Kimura and Nakano, 2000; Chow and Li, 2000; Li et al., 2005), long-short term mem-

ory (LSTM) (Wang, 2017) and gated recurrent units (GRUs) (Jordan et al., 2019; De Brouwer et al., 2019).

In this paper, we focus on the applications of neural networks in learning the dynamics of time delay systems. In many engineering and physical systems the dynamics are affected by past states, which leads to the occurrence of time delays. For example, time delays show up in the areas of vehicular traffic (Burger et al., 2019; Molnár et al., 2021), tire models (Beregi et al., 2019), manufacturing processes (Quintana and Ciurana, 2011; Molnár et al., 2017), human balancing (Milton et al., 2015), biological networks (Chen and Aihara, 2002; Orosz et al., 2010) or even in the spreading of infectious diseases like COVID-19 (Casella, 2020; Ames et al., 2020). Time delays often lead to rich and complex dynamic behavior. They are rarely considered or studied in feed-forward neural networks, although the idea of introducing delays into FNNs was already proposed in the 1980's for speech recognition (Waibel et al., 1989). Here we aim to fill this gap and extend the usage of feed-forward neural networks in learning the dynamics of time delay systems.

The rest of the paper is organized as follows. In Section 2, we relate time delay systems and feed-forward neural networks and give a short literature overview on this topic. Then, we introduce a continuous delay learning algorithm together with weights learning in Section 3. To illustrate the algorithm, we give an example of learning two delays and nonlinearities from simulation data using two network architectures in Section 4. We also highlight a fundamental problem of learning the differential instead of the true dynamics when using FNNs with delayed inputs. In Section 5, we conclude our results and lay out the possible extensions of the algorithm.

2. Delays in Dynamical Systems and Neural Networks

Delays have gained popularity in learning dynamics in the recent years. For example, some researchers started to include delays into FNN architectures and considered the effect of delays in dynamical systems. This has led to so-called time delay neural networks (TDNNs), which are widely used in mapping one sequence of data to another (Wan, 1994). In general, delays can exist in each layer of a TDNN, however, delays are sometimes considered within the input layer only for a better interpretability. For example, as an application to capturing car-following behavior, Khodayari et al. (2012) added delay as another input to FNNs, Zheng et al. (2013) used a separate neural network to train an instantaneous input delay, Wang et al. (2018) compared the use of RNNs to FNNs with fixed delayed inputs, and Ji et al. (2020) constructed delayed FNNs based on first principle models.

In what follows, we use TDNNs to study autonomous nonlinear delayed dynamical systems written in the form

$$\dot{x}(t) = g(x(t), x(t-\tau_1), ..., x(t-\tau_K)), \tag{1}$$

$x(t) \in \mathbb{R}^n$. Our objective is to approximate the nonlinear mapping $g(\cdot)$ and at the same time learn the delay values $\tau_1, \ldots, \tau_K$ by including them as trainable parameters along with the weights and biases. We intend to learn delays in a continuous fashion, i.e., without restricting ourselves to discrete delays that are integer multiples of the time step in the data. To increase the interpretability of the network, we impose a restriction that delays are used in the input layer.

Chen et al. (2015) gave a method of learning a continuous delay from data, however, neural networks are not incorporated in that approach. The idea of using adaptive delays in TDNNs was first proposed by Lin et al. (1992, 1995). They included delays in hidden layer which may be hard to interpret and can make the networks hard to train or simulate. Learning the actual delay in the excitation through a linear combination of basis functions was realized by Ren and Rad

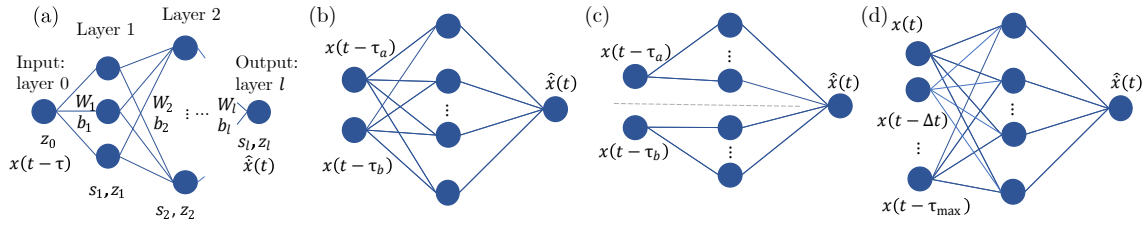

Figure 1: Schematic diagram of network architectures: (a) deep neural network with one delayed input; (b) fully-connected shallow network with two trainable delayed inputs; (c) decoupled shallow network with two trainable delayed inputs; (d) standard TDNN with a window of fixed delayed inputs.

(2007) while Ji et al. (2020) used special network architectures to learn the time delay. In Lin et al. (1992, 1995) and Ji et al. (2020), the learned delays were discrete, while Ren and Rad (2007) only considered a single delay in the excitation. In this paper, we propose a continuous delay learning algorithm with multiple delays using TDNNs. We implement the algorithm on an example and compare the simulations of the learned networks to those given by a standard TDNN with a window of fixed delayed inputs.

3. Feed-forward Neural Networks with Trainable Delays

In this section, we introduce feed-forward neural networks with trainable input delays; see the corresponding network architecture and notation in Fig. 1(a). For simplicity, the following equations describe a single input and single output example with one delay variable τ but the formulas can be generalized to multiple dimensions with multiple delays straightforwardly. We show an example of learning two delays in Section 4 with the network architectures illustrated in Fig. 1(b) and (c), and compare the performance with the standard TDNN architecture depicted in Fig. 1(d).

First, we consider a fully-connected neural network with input layer (layer 0), $l-1$ hidden layers and output layer (layer l). Activation functions $f_i(\cdot)$ are applied at each layer (except the input layer) with the derivative $f_i'(s_i) = \frac{\partial f_i(s_i)}{\partial s_i}$ well-defined for all $i = 1, 2, ..., l$. The equation of this network can be written as

$$\begin{aligned} s_i &= W_i z_{i-1} + b_i, \\ z_i &= f_i(s_i), \end{aligned} \tag{2}$$

where s_i is the linear combination of z_{i-1}, the outputs of layer $i-1$. The input of the network is defined as z_0 and the output of the network is $z_l = f_l(s_l)$ where we can choose $f_l(\cdot)$ to be a linear function.

In the context of learning the dynamics given by (1), the input z_0 corresponds to the signal $x(t-\tau)$ and the output z_l gives a prediction of the state derivative denoted by $\hat{\dot{x}}(t)$ that approximates $\dot{x}(t)$. Note that the input and output data are usually discretized with a time step Δt and are only available at the discrete time moments $t^j = j\Delta t$, $j \in \mathbb{Z}$. In what follows, we use superscript j to represent the data point corresponding to time t^j and we denote variables without superscript when referring to all available data. The input and output are therefore given by $z_0^j = x(t^j - \tau)$ and

$z_l^j = \hat{\dot{x}}(t^j)$. Note that $x(t^j - \tau)$ may not exist in the data if τ is not an integer multiple of Δt, hence we will approximate it later in this section as $\tilde{x}(t^j - \tau)$ based on the network's delay parameter τ and the data we have.

We define the loss function as

$$L = \frac{1}{N} \sum_{j=1}^{N} \left(z_l^j - y^j\right)^2, \tag{3}$$

where N is the number of data points we consider in each update and y^j is the output corresponding to time t^j, that is, $y^j = \dot{x}(t^j)$ when learning the dynamics of (1). We use the loss function (3) for parameter training with gradient descent method. Delays are viewed the same way as other parameters, that is, we iterate the delay values based on the same updating rule. The updating formula for parameter θ at iteration $n+1$ uses the gradient of the loss function at iteration n:

$$\theta(n+1) = \theta(n) - \eta_\theta(n) \frac{\partial L}{\partial \theta}(n). \tag{4}$$

The learning rate η_θ can be a tuned constant or be adaptive during the training. We can potentially choose different learning rates for delays and weights. A significant difference between delay parameters and weights is that we have a constraint over the delays: $0 \leq \tau \leq \tau_{\max}$, since the value of the delay cannot be negative and since time delay systems may have solutions which reappear for infinitely many different delay values (Yanchuk and Perlikowski, 2009).

Here, we also point out the difference between learning continuous delays and learning discrete delays. For learning continuous delays, we have the following updating formula:

$$\tau(n+1) = \max\left\{\tau(n) - \eta_\tau(n) \frac{dL}{d\tau}(n), 0\right\}, \tag{5}$$

where the time delay τ is a non-negative real number. Although the delayed states may not exist in the dataset in this case, we can still approximate them from adjacent data points. In discrete delay learning (Ji et al., 2020), we have

$$\tau(n+1) = \text{round}\left(\frac{\max\left\{\tau(n) - \eta_\tau(n)\frac{dL}{d\tau}(n), 0\right\}}{\Delta t}\right)\Delta t, \tag{6}$$

which is an integer multiple of the time step. Using the discrete values ensures that the delayed states exist in the dataset. However, the rounding in the updating formula leads to some problems. For example, it is difficult to find a good learning rate for the delay so that the delay can still change when the gradient is small. Since the delay cannot evolve smoothly, more local minima appears. Also, the learned delay is limited by the time step Δt. When Δt is relatively large compared to how fast the system evolves, learning the continuous delays is more robust.

The gradient information of weights and biases can be expressed by the following equations

$$\frac{\partial L}{\partial b_i} = \frac{\partial s_i}{\partial b_i} h_i \frac{\partial L}{\partial z_l} = \frac{2}{N} \sum_{j=1}^{N} h_i^j \left(z_l^j - y^j\right), \tag{7}$$

$$\frac{\partial L}{\partial W_i} = \frac{\partial s_i}{\partial W_i} h_i \frac{\partial L}{\partial z_l} = \frac{2}{N} \sum_{j=1}^{N} z_{i-1}^j h_i^j \left(z_l^j - y^j\right), \tag{8}$$

where

$$h_i = \frac{\partial z_i}{\partial s_i}\frac{\partial s_{i+1}}{\partial z_i}h_{i+1} = f_i'(s_i)W_{i+1}h_{i+1}, \tag{9}$$

for $i = 1, 2, ..., l-1$ while $h_l = f_l'(s_l)$.

Similarly, we define the gradient with respect to the delay τ as

$$\frac{\partial L}{\partial \tau} = \frac{\partial z_0}{\partial \tau}\frac{\partial s_1}{\partial z_0}h_1\frac{\partial L}{\partial z_l} = \frac{2}{N}\sum_{j=1}^{N}\frac{\partial z_0^j}{\partial \tau}W_1 h_1^j\big(z_l^j - y^j\big). \tag{10}$$

For multiple delays and multiple states, we can simply concatenate the delayed states together, by creating a large input vector and a vector τ from delay values. Note that the input z_0^j is the approximate delayed state $\tilde{x}(t^j - \tau)$, thus we can write

$$\frac{\partial z_0^j}{\partial \tau} = \frac{\partial \tilde{x}(t^j - \tau)}{\partial \tau} = -\frac{\partial \tilde{x}(t^j - \tau)}{\partial t} = -\tilde{\dot{x}}(t^j - \tau), \tag{11}$$

where $\tilde{\dot{x}}(t^j - \tau)$ is approximate time derivative of the delayed states. That is, learning a delay with gradient-based methods requires information about the derivative $\dot{x}$ in addition to the input x itself. This is a significant difference compared to learning weights and biases.

We consider the delay as continuous variable even though data are only available at the discrete time moments t^j. That is, $t^j - \tau$ may not coincide with any sampling time t^k. Thus, we approximate the delayed state $x(t^j - \tau)$ and the delayed state derivative $\dot{x}(t^j - \tau)$ by the closest data point using a round function in the time argument. Note that introducing the round function here does not affect the continuity in the delay learning (τ is not restricted to integer multiples of Δt). One can also use linear interpolation between neighboring data points or zero-order-hold approximation.

Algorithm 1 Mini-batch gradient descent with input delays training

Data: training data and validation data
Result: learn W_i, b_i and τ from data
normalize inputs and outputs between [0,1] and get derivatives of normalized inputs
choose the initial learning rate $\eta(1)$ and learning rate updating method
set maximum iteration number $n_{\max}$, maximum violation number $v_{\max}$ and maximum allowed delay $\tau_{\max}$
initialize $W_i(1)$ between [-1,1], $b_i(1) = 0$ and $\tau_i(1)$ within interval [0, $\overline{\tau}$], $\overline{\tau} < \tau_{\max}$, $v = 0$
for n = 1,..., $n_{\max}$ **do**
  randomly take one batch of data from the training set
  shift batch based on $\tau(n)$, calculate loss $L_{\mathrm{tr}}(n)$, $L_{\mathrm{va}}(n)$ for training data and validation data
  get gradient information and update parameters $W_i(n+1)$, $b_i(n+1)$ and $\tau(n+1)$
  if $n > 1$ *and* $L_{\mathrm{va}}(n) \geq L_{\mathrm{va}}(n-1)$ **then**
    $v = v + 1$
  else
    $v = 0$
  end
  if $\tau(n+1) > \tau_{\max}$ *or* $v > v_{\max}$ **then**
    break loop
  end
end
$W_i = W_i(n)$, $b_i = b_i(n)$, $\tau = \tau(n)$

We describe the training process in Algorithm 1. It is a mini-batch gradient descent algorithm for training dataset which consists of multiple input-output trajectories. The validation dataset contains different trajectories so that it is independent from the training dataset, and those data are not used for updating parameters but for monitoring the training. For each parameter update, one batch of data (corresponding to one trajectory) is taken for calculating the training loss L_{tr}, and the validation dataset is used for calculating the validation loss L_{va}. If the validation error increases consecutively for $v_{\max}$ iterations, we stop training to prevent overfitting. After training is done, we test the performance of the trained networks on a separate testing dataset.

4. Learning the Dynamics of Nonlinear Time Delay Systems

Consider an autonomous delayed dynamical system which has quadratic and cubic nonlinearities in two delayed terms:

$$\dot{x}(t) = a_2 x^2(t-\tau_2) + a_3 x^3(t-\tau_3). \tag{12}$$

We choose the parameters $a_2 = a_3 = -1$, $\tau_2 = 1$ and $\tau_3 = 0.5$ for which the system has two equilibria at $x = 0$ and $x = -1$. We are interested in how well shallow networks with input delays can learn the nonlinearity from a limited number of trajectories and whether such networks can learn the delays in the system.

We simulate (12) to obtain four batches of data for training by using constant values $c_{\mathrm{tr}} = \{0.8, 1.0, 1.2, 1.4\}$ as histories, i.e, $x(t) \equiv c_{\mathrm{tr}}$ for $t \in [-\tau_{\max}, 0]$. Simulation data with constant history $c_{\mathrm{va}} = 1.1$ is used for validation and we test the trained networks using the constant history $c_{\mathrm{ts}} = 1.3$. Each batch of data contains the state information $x(t^j)$ over $0 \le t^j \le 5$ with a time step $\Delta t = 0.01$. We obtain the time derivative from state through forward Euler method. The derivative is used both as output and for calculating the gradient with respect to delay in (11). We normalize the input and output between $[0, 1]$ and choose the learning rate $\eta_\theta(n) \equiv 0.1$ for all parameters including delays. In training, we also include the initial constant history in the data so that the shifting between inputs and outputs does not affect the length of prediction. Another reason for including the history into the training data is the fact that for autonomous delayed dynamical systems the solutions get smoother as time t increases (Michiels and Niculescu, 2007; Insperger and Stépán, 2011). Thus, the initial segments of the solution contain the "richest" dynamics for a stable trajectory without excitation. This also indicates that longer trajectories do not necessarily mean more beneficial datasets, because the significant initial segments have less weight.

We implement the delay training algorithm with a fully-connected network and a decoupled network shown in Fig. 1(b) and (c). Both are shallow neural networks and the difference is the structure of the hidden layer. The fully-connected network does not contain any prior knowledge about the form of (12), while the decoupled architecture implies that there is no cross term between delayed states. The number of neurons in the hidden layer is $M = 30$ for fully-connected network and $M = [20, 20]$ for the two parts of the decoupled network. Both networks use ReLU as activation function in the hidden layer, i.e., $f(z) = \max\{z, 0\}$, and we set $\tau_{\max} = 2$.

The evolution of the loss L and the learned delays τ_a and τ_b along the iterations of one training process is shown in Fig. 2(a,b) for the fully-connected architecture and in Fig. 2(d,e) for the decoupled architecture. We also show how the true nonlinearities are approximated by these two networks in Fig. 2(c) and (f). Note that the neural networks do not distinguish the order of the two delays, that is, τ_a and τ_b can approach τ_2 and τ_3 or, alternatively, τ_3 and τ_2, respectively. Nevertheless, the two delays converge to the true values 1 and 0.5 with both architectures. The delays in

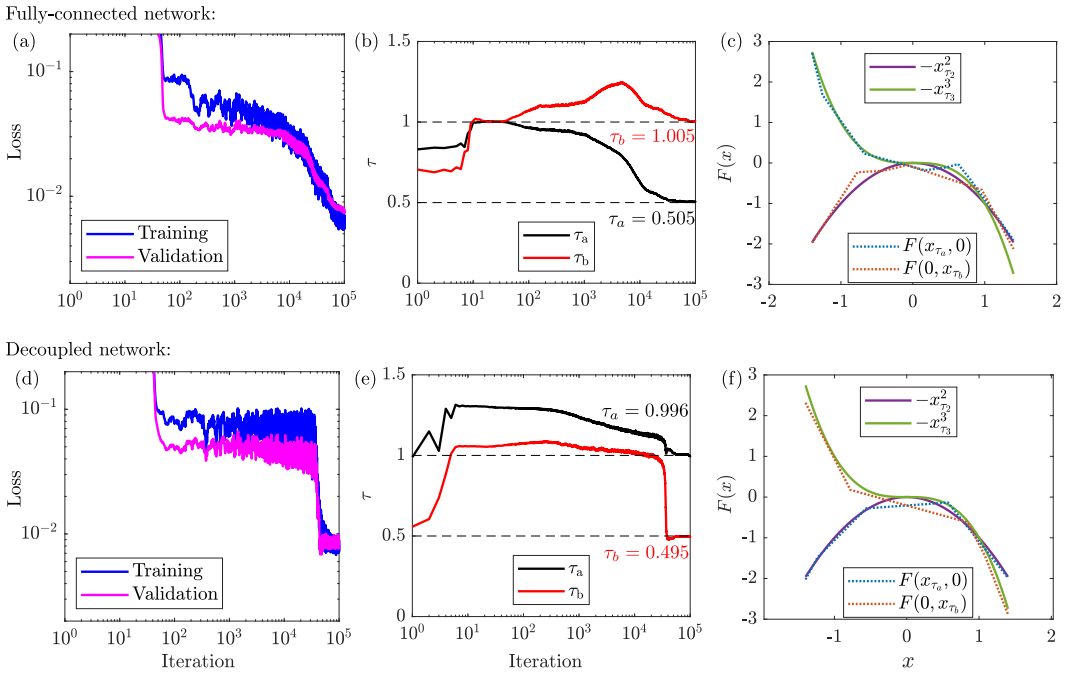

Figure 2: Evolution of the loss (left) and delays (middle) over the iterations during training, and the equivalent approximation F of the nonlinearities given by trained networks (right): (a)-(c) fully-connected network; (d)-(f) decoupled network.

the input layer can be learned and they can be interpreted clearly as the trained networks represent the delayed dynamical system. Moreover, the nonlinearities are also recovered from data with the meaningful learned delays. We can see the piece-wise property of the network approximation given by the ReLU activation function in the hidden layer.

To investigate the performance of the trained networks, we analyze the system

$$\dot{x}(t) = F(x(t-\tau_a), x(t-\tau_b)), \tag{13}$$

with histories $c_{\text{tr}} = 1.4$ and $c_{\text{ts}} = 1.3$, where F, τ_a and τ_b are the nonlinearity and delays learned by the networks. First, we inspect the input-output relationship between $x(t)$ and $\dot{x}(t)$, which we refer to as static mapping. Then, we simulate system (13) using the learned parameter values which we refer to as the neural network simulation results. Both of these results are compared to the those obtained from simulating the original system (12). For further comparison, we also consider a standard TDNN which uses $x(t), x(t-\Delta t), ..., x(t-\tau_{\max})$ as inputs and $\dot{x}(t)$ as output. This architecture is referred to as network with a window of delayed inputs. Since $\Delta t = 0.01$ and $\tau_{\max} = 2$, this TDNN has 201 inputs where 200 of them are delayed with different quantized delays. Here we use a shallow TDNN with 12 hidden neurons and ReLU activation function as benchmark to learn from the same data.

The three trained networks are compared to the data in Fig. 3(a) and (b) in terms of static mapping and simulation results of the trajectory with $c_{\text{tr}} = 1.4$ from training dataset. Figure 3(c) and (d) show the performance of the trained networks on testing data. All three networks have good static mapping performance on both training and testing dataset. In neural network simulation, the

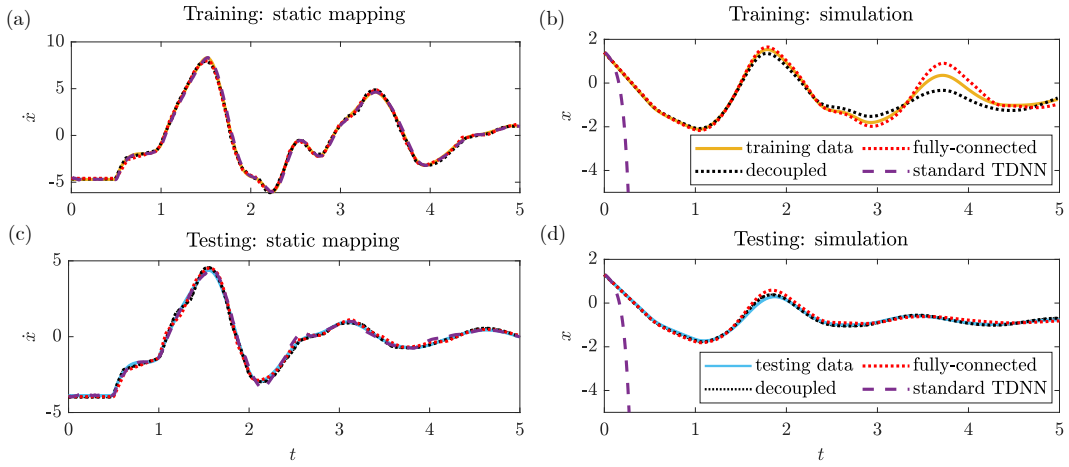

Figure 3: Static mapping and simulation results for the trained networks: (a), (b) performance on one of the trajectories with $c_{\text{tr}} = 1.4$ in the training dataset; (c), (d) performance on testing dataset with $c_{\text{ts}} = 1.3$.

Architecture		Standard TDNN	Fully-connected network	Decoupled network
Training loss $\sqrt{L_{\text{tr}}}$	Static mapping ($\dot{x}$)	0.0229	**0.0102**	0.0106
	Simulation (x)	∞	**0.0514**	0.0676
Testing loss $\sqrt{L_{\text{ts}}}$	Static mapping ($\dot{x}$)	0.1453	0.0096	**0.0085**
	Simulation (x)	∞	0.0305	**0.0143**
τ_a/τ_b		-	0.505/1.005	0.996/0.495

Table 1: Performances of different networks on training and testing trajectories. The normalized root-mean-square deviations are calculated based on $\dot{x}$ for the performance of static mapping and on x for the performance of neural network simulation.

networks with two trainable delays give reasonable simulations close to the data, while the standard TDNN with 200 delays fails (the simulation blows up). This shows that with trainable delays, the networks have the capability of approximating the true dynamics given by (12).

Remark 1 *The standard TDNN may have the overfitting problem due to its large number of distributed delays when it is used to learn a system with small number of point delays.*

Also, the simulation of the standard TDNN with 200 delayed inputs is more time-consuming compared to the other two networks. The root-mean-square deviations on training data $\sqrt{L_{\text{tr}}}$ and on testing data $\sqrt{L_{\text{ts}}}$ of all trained networks are collected in Table 1. Since we normalize all data before training, $\sqrt{L_{\text{tr}}}$ and $\sqrt{L_{\text{ts}}}$ are also normalized values according to the maximum and minimum values of $\dot{x}$ and x in the training set. From the table we notice that two networks with trainable delays outperform the standard TDNN on testing data in both static mapping and neural network simulation.

With multiple runs of training, we also discovered some undesired local minima introduced by delays in the networks, which led us to the following observation.

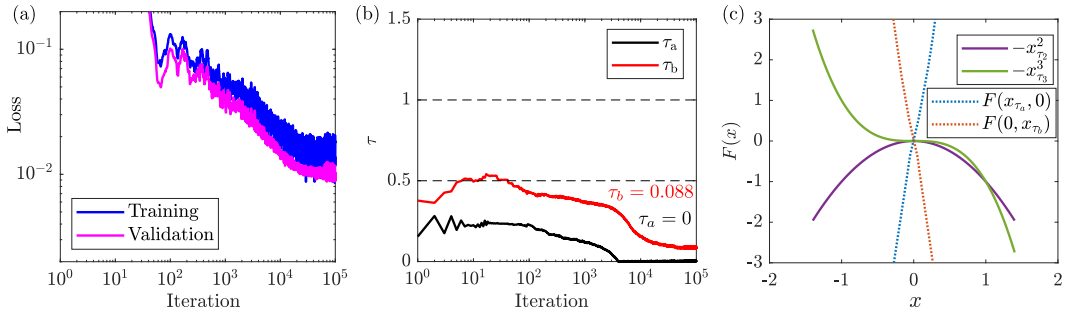

Figure 4: Example of the fully-connected network learning the differential: (a), (b) evolution of the loss and delays, respectively, over the iterations during training, (c) the approximation given by trained networks.

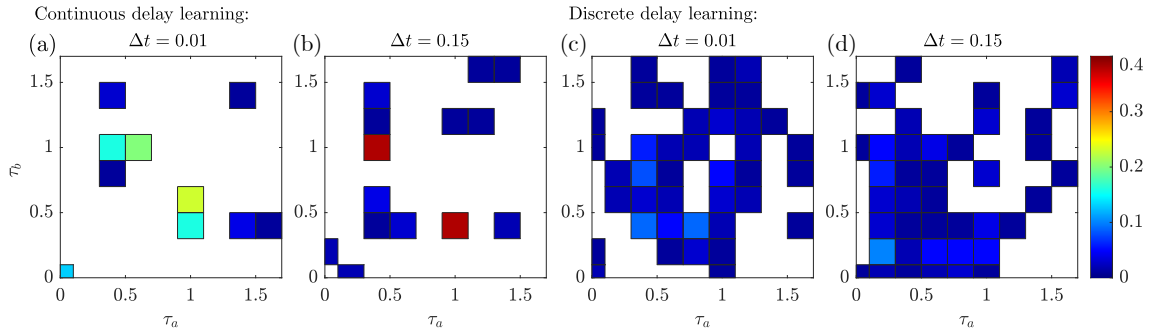

Figure 5: Histograms of final learned delays from 100 training runs with initial delays distributed uniformly over $[0, 1]$. The study is performed on the fully-connected network with different time steps and delay learning methods: (a), (b) continuous delay learning; (c), (d) discrete delay learning. Color indicates the frequency of the delays occurring in a region.

Remark 2 *If we use TDNNs with input delays for learning the dynamics of delay differential equations (mapping the delayed states to the time derivative), the TDNN has a possibility of approximating the differential instead of the right hand side of the equations.*

In other words, the training may end up learning to differentiate when iterating F, τ_a and τ_b. For example, a backward Euler method $\dot{x}(t) \approx (x(t) - x(t - \Delta\tau))/\Delta\tau$ may result from learning $\tau_a = 0$, a small value $\tau_b = \Delta\tau$ and $F(x(t - \tau_a), x(t - \tau_b)) = (x(t - \tau_a) - x(t - \tau_b))/\Delta\tau$. This problem can happen to all time delay neural networks used for mapping the states to the time derivative of the states, especially for the case where Δt is small. For instance, the results of the fully-connected network learning the backward Euler method are shown in Fig. 4. The training and validation errors still decrease to a small value in panel (a) and the delays in panel (b) converge to $\tau_a = 0$ and a small value $\tau_b = 0.088$. Panel (c) shows that the visualization of the learned nonlinearity F appears to be two lines with slopes around $\pm 1/\tau_b \approx \pm 11.4$, which indicates that the network is approximating the backward Euler method. Since the data is discretized with forward Euler method and $\Delta t = 0.01$, the error between two discretization methods depends on the time step.

We also increase Δt to test the robustness of the proposed continuous delay learning method (5) and compare it to the discrete delay learning method (6). We choose $\Delta t = 0.15$ to create low-resolution datasets for training. This larger time step not only introduces larger numerical differentiation error when generating the data for $\dot{x}$, but also affects the gradients with respect to delays (in (11) the time derivative $\tilde{\dot{x}}$ approximated by adjacent data points is inaccurate). We consider previous datasets with $\Delta t = 0.01$ as high-resolution datasets. We ran 100 training trials with initial delays distributed uniformly over $[0, 1]$ on both high-resolution and low-resolution datasets. The histogram of the final learned delays for the fully-connected network is plotted in Fig. 5.

Remark 3 *The continuous delay learning method is more robust compared to discrete delay learning method on both high-resolution data and low-resolution data.*

With discrete delay learning, the learning process is easily trapped in many local minima introduced by the discretization in the delay parameter space. Although the learning rate was tuned for discrete delay learning separately, the convergence of delays is not clear. Choosing a small time step does not resolve this issue and having a large time step makes the learning performance worse. However, with the continuous delay learning algorithm, most of the delays converge to the correct values or to the very small values related to the differential. Even when the time step is large, the continuous delay learning algorithm is able to approach the true delay values.

5. Summary

We proposed a delay learning algorithm with time delay neural networks and we showed its implementation for learning delays and nonlinearities in the dynamics of time delay systems. By comparing the simulations of networks with two trained delays to those of a standard TDNN with 200 fixed delays, we demonstrated the advantages of using trained delays over considering a window of delayed inputs. We discovered a generic problem in learning the dynamics of time delay systems: neural networks can potentially learn the differential instead of the true dynamics. We tested the robustness of our proposed continuous delay learning method by comparing the convergence in delays to the discrete delay learning with large time step (with larger numerical differentiation error in data). The continuous delay learning method is able to get the correct delay values for most of the cases. Although the example given in this paper is restricted to shallow networks, we also successfully applied the input delay training on fully-connected deep neural networks. The input delays can be learned through deep neural networks, however, the training process is more susceptible to running into undesired local minima compared to the shallow networks.

In the future, we plan to improve the robustness of the algorithms to avoid undesired local minima and speed up the delay learning. Potential directions can be: using batch normalization in each layer, applying more advanced optimization methods (e.g., Levenberg–Marquardt algorithm) for parameter training, or including the simulation error into the loss function.

References

Aaron D Ames, Tamás G Molnár, Andrew W Singletary, and Gábor Orosz. Safety-critical control of active interventions for COVID-19 mitigation. *IEEE Access*, 8:188454–188474, 2020. doi: 10.1109/ACCESS.2020.3029558.

Sándor Beregi, Dénes Takács, and Gábor Stépán. Bifurcation analysis of wheel shimmy with non-smooth effects and time delay in the tyre–ground contact. *Nonlinear Dynamics*, 98(1):841–858, 2019. ISSN 1573-269X. doi: 10.1007/s11071-019-05123-1.

Michael Burger, Simone Göttlich, and Thomas Jung. Derivation of second order traffic flow models with time delays. *Networks & Heterogeneous Media*, 14(2):265–288, 2019.

Francesco Casella. Can the COVID-19 epidemic be controlled on the basis of daily test reports? *arXiv preprint*, page 2003.06967, 2020.

Fengwei Chen, Hugues Garnier, and Marion Gilson. Robust identification of continuous-time models with arbitrary time-delay from irregularly sampled data. *Journal of Process Control*, 25: 19–27, 2015.

Luonan Chen and Kazuyuki Aihara. Stability of genetic regulatory networks with time delay. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 49(5):602–608, 2002.

Tommy WS Chow and Xiao-Dong Li. Modeling of continuous time dynamical systems with input by recurrent neural networks. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 47(4):575–578, 2000.

Edward De Brouwer, Jaak Simm, Adam Arany, and Yves Moreau. GRU-ODE-Bayes: Continuous modeling of sporadically-observed time series. In *Advances in Neural Information Processing Systems*, pages 7379–7390, 2019.

Tamás Insperger and Gábor Stépán. *Semi-Discretization for Time-Delay Systems*. Springer, 2011.

Xunbi A Ji, Tamás G Molnár, Sergei S Avedisov, and Gábor Orosz. Feed-forward neural networks with trainable delay. volume 120 of *Proceedings of Machine Learning Research*, pages 127–136, 10–11 Jun 2020.

Ian D Jordan, Piotr Aleksander Sokol, and Il Memming Park. Gated recurrent units viewed through the lens of continuous time dynamical systems. *arXiv preprint arXiv:1906.01005*, 2019.

Alireza Khodayari, Ali Ghaffari, Reza Kazemi, and Reinhard Braunstingl. A modified car-following model based on a neural network model of the human driver effects. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 42(6):1440–1449, 2012.

Masahiro Kimura and Ryohei Nakano. Dynamical systems produced by recurrent neural networks. *Systems and Computers in Japan*, 31(4):77–86, 2000.

S Narendra Kumpati and Parthasarathy Kannan. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27, 1990.

John G Kuschewski, Stefen Hui, and Stanislaw H Zak. Application of feedforward neural networks to dynamical system identification and control. *IEEE Transactions on Control Systems Technology*, 1(1):37–49, 1993.

Xiao-Dong Li, John KL Ho, and Tommy WS Chow. Approximation of dynamical time-variant systems by continuous-time recurrent neural networks. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 52(10):656–660, 2005.

Daw-Tung Lin, Judith E Dayhoff, and Panos A Ligomenides. Adaptive time-delay neural network for temporal correlation and prediction. In *Intelligent Robots and Computer Vision XI: Biological, Neural Net, and 3D Methods*, volume 1826, pages 170–181. International Society for Optics and Photonics, 1992.

Daw-Tung Lin, Judith E Dayhoff, and Pangs A Ligomenides. Trajectory production with the adaptive time-delay neural network. *Neural Networks*, 8(3):447–461, 1995.

Wim Michiels and Silviu-Iulian Niculescu. *Stability and stabilization of time-delay systems: An eigenvalue-based approach*. SIAM, 2007.

John Milton, Tamás Insperger, and Gábor Stépán. Human balance control: Dead zones, intermittency, and micro-chaos. In *Mathematical Approaches to Biological Systems*, pages 1–28. Springer, 2015.

Tamás G Molnár, Zoltán Dombovári, Tamás Insperger, and Gábor Stépán. On the analysis of the double hopf bifurcation in machining processes via centre manifold reduction. *Proceedings of the Royal Society A*, 473(2207):20170502, 2017.

Tamás G Molnár, Devesh Upadhyay, Michael Hopka, Michiel Van Nieuwstadt, and Gábor Orosz. Delayed lagrangian continuum models for on-board traffic predictions. *Transportation Research Part C*, 123:102991, 2021.

Gábor Orosz, Jeff Moehlis, and Richard M Murray. Controlling biological networks by time-delayed signals. *Philosophical Transactions of the Royal Society A*, 368(1911):439–454, 2010.

Guillem Quintana and Joaquim Ciurana. Chatter in machining processes: A review. *International Journal of Machine Tools and Manufacture*, 51(5):363–376, 2011.

Xue-Mei Ren and Ahmad B Rad. Identification of nonlinear systems with unknown time delay based on time-delay neural networks. *IEEE transactions on neural networks*, 18(5):1536–1541, 2007. doi: 10.1109/TNN.2007.899702.

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

Arjen Van Ooyen and Bernard Nienhuis. Improving the convergence of the back-propagation algorithm. *Neural Networks*, 5(3):465–471, 1992.

Alex Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J Lang. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(3):328–339, 1989.

Eric Wan. Time series prediction by using a connectionist network with internal delay lines. In *Time Series Prediction*, pages 195–217. Addison-Wesley, 1994.

Xiao Wang, Rui Jiang, Li Li, Yilun Lin, Xinhu Zheng, and Fei-Yue Wang. Capturing car-following behaviors by deep learning. *IEEE Transactions on Intelligent Transportation Systems*, 19(3): 910–920, 2018.

Yu Wang. A new concept using LSTM Neural Networks for dynamic system identification. In *2017 American Control Conference*, pages 5324–5329. IEEE, 2017.

Serhiy Yanchuk and Przemyslaw Perlikowski. Delay and periodicity. *Physical Review E*, 79(4): 046221, 2009.

Jing-Ru Zhang, Jun Zhang, Tat-Ming Lok, and Michael R Lyu. A hybrid particle swarm optimization–back-propagation algorithm for feedforward neural network training. *Applied Mathematics and Computation*, 185(2):1026–1037, 2007.

Jian Zheng, Koji Suzuki, and Motohiro Fujita. Car-following behavior with instantaneous driver–vehicle reaction delay: A neural-network-based methodology. *Transportation Research Part C: Emerging Technologies*, 36:339–351, 2013.