Proceedings of Machine Learning Research 120:1–10, 2020

Feed-forward Neural Networks with Trainable Delay

Xunbi A. Ji XUNBIJ@UMICH.EDU
Tamás G. Molnár MOLNART@UMICH.EDU
Sergei S. Avedisov* AVEDISKA@UMICH.EDU
Gábor Orosz OROSZ@UMICH.EDU
Department of Mechanical Engineering, University of Michigan, Ann Arbor, MI 48109, USA

Editors: A. Bayen, A. Jadbabaie, G. J. Pappas, P. Parrilo, B. Recht, C. Tomlin and M. Zeilinger

Abstract

In this paper we build a bridge between feed-forward neural networks and delayed dynamical systems. As an initial demonstration, we capture the car-following behavior of a connected automated vehicle that includes time delay by using both simulation data and experimental data. We construct a delayed feed-forward neural network (DFNN) and introduce a training algorithm in order to learn the delay. We demonstrate that this algorithm works well on the proposed structures.

Keywords: delayed feed-forward neural network, car-following, connected automated vehicle, time delay system

1. Introduction

Neural networks are widely used when models are hard to obtain through first principles. However, little effort has been made to incorporate time delays into such networks. This is in contrast with the dynamical systems literature, where many models have been augmented with time delays and it has been shown that even simple systems may exhibit rich dynamics Molnar et al. (2017); Orosz et al. (2009). This suggests that adding time delays to neural networks of simple structure may allow us to capture a large variety of behaviors. This may provide a viable alternative to increasing the complexity of the networks (number of layers and number of neurons). Delays have been incorporated in neural network structures to improve the performance of neural convolution models de Vries and Principe (1990) in applications such as word and speech recognition Tank and Hopfield (1987); Lang et al. (1990). The delays in those networks are used for concentrating information in time, while the structures and cost functions become relatively complicated. We consider neural networks with delays from different perspectives: learning the delay explicitly from data, utilizing simple network structures, and imposing a physics based model on the neural network.

As a first step we consider a simple delayed dynamical system and reformulate it as a delayed feed-forward neural network (DFNN). Then, we establish a training algorithm that allows us to learn the time delay in the network. As an example we consider the dynamics of a connected automated vehicle (CAV) following a connected human-driven vehicle (CHV) Zhang and Orosz (2016); Ge et al. (2018). In this case the longitudinal controller of the CAV is known and we have an estimate of the time delay that arises from the powertrain and from the wireless communication between the CAV and the CHV. We demonstrate that the DFNN reproduces the system dynamics when the delay is selected appropriately and that our training algorithm converges. We remark that there has been

* Now working with Toyota Motor North America R&D-Infotech Labs, Mountain View, CA 94043, USA

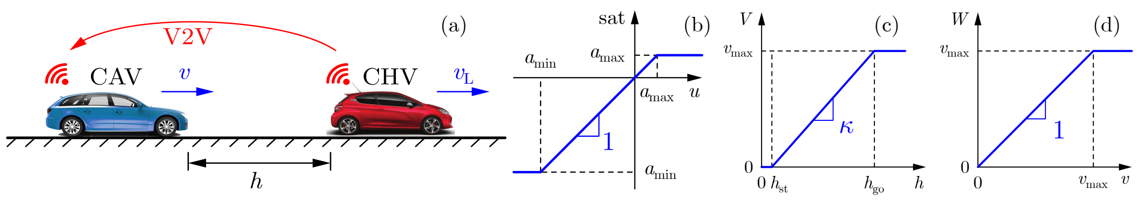

Figure 1: (a) Car-following scenario considered in the paper; (b, c, d) Nonlinearities in the control law of the CAV.

efforts in capturing and controlling car-following dynamics using neural networks Wu et al. (2017); Panwai and Dia (2007); Wang et al. (2018); Wu and Work (2018) but, to the best of our knowledge, our work is the first one that explicitly incorporates a trainable time delay in such systems.

The paper is constructed as follows. In Sec. 2 we describe the car-following dynamics and the corresponding datasets. In Sec. 3 we construct two delayed feed-forward neural networks that are equivalent to the physical system at the linear and nonlinear levels, respectively. Section 4 is devoted to the delay searching algorithm while Sec. 5 summarizes the key results.

2. Car-following Dynamics and Data

In this section, we introduce the car-following scenario for which we demonstrate our methods. We show the experimental data we use for training and describe a first principle model that can be used to reproduce the data.

The experimental data is described in detail in Avedisov et al. (2018) where a connected automated vehicle (CAV) interacted with connected human-driven vehicles (CHVs) on a virtual ring. By controlling the length of the virtual ring, the average spacing h^* between vehicles was set, which was related to traffic density. Here we only focus on the case where the CAV responds to a single CHV immediately ahead, see Fig. 1 (a) with the velocities v, v_L and the headway h indicated. We have ten runs with different average spacing from $h^* = 5\,\mathrm{m}$ to $h^* = 50\,\mathrm{m}$. The time is discretized as $t^j = j\Delta t$, with $\Delta t = 0.1\,\mathrm{s}$. The length of the data in the ten runs varies between $99\,\mathrm{s}$ and $200\,\mathrm{s}$.

The longitudinal dynamics can be modeled by

$$\begin{aligned} \dot{h}(t) &= v_\mathrm{L}(t) - v(t), \\ \dot{v}(t) &= \mathrm{sat}(u(t-\tau)), \end{aligned} \qquad \mathrm{sat}(u) = \begin{cases} a_\mathrm{min} & u < a_\mathrm{min}, \\ u & a_\mathrm{min} \le u \le a_\mathrm{max}, \\ a_\mathrm{max} & u > a_\mathrm{max}, \end{cases} \tag{1}$$

where the dot represents the derivative with respect to time t. The acceleration was assigned by the controller:

$$u(t) = \alpha\Big(V(h(t)) - v(t)\Big) + \beta\Big(W(v_\mathrm{L}(t)) - v(t)\Big), \tag{2}$$

$$V(h) = \begin{cases} 0 & h < h_\mathrm{st}, \\ \kappa(h - h_\mathrm{st}) & h_\mathrm{st} \le h \le h_\mathrm{go}, \\ v_\mathrm{max} & h > h_\mathrm{go}, \end{cases} \qquad W(v_\mathrm{L}) = \begin{cases} v_\mathrm{L} & v_\mathrm{L} \le v_\mathrm{max}, \\ v_\mathrm{max} & v_\mathrm{L} > v_\mathrm{max}, \end{cases}$$

where α and β are control gains and the nonlinearites are shown in Fig. 1 (b,c,d). We consider the parameters $\alpha = 0.4\,\mathrm{s}^{-1}$, $\beta = 0.5\,\mathrm{s}^{-1}$, $\kappa = 0.6\,\mathrm{s}^{-1}$, $h_\mathrm{st} = 5\,\mathrm{m}$, $v_\mathrm{max} = 30\,\mathrm{m/s}$,

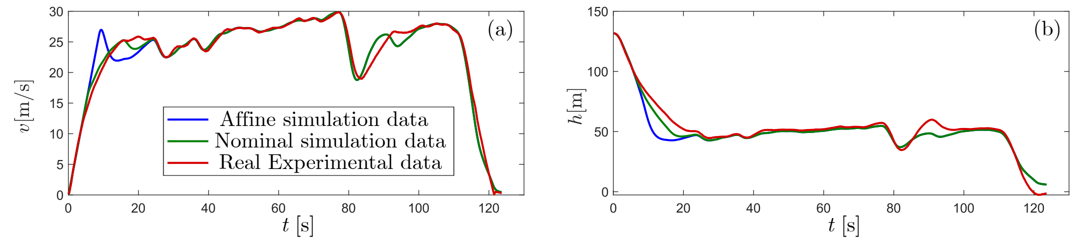

Figure 2: (a) Velocity and (b) headway in three sets of data ($h^* = 45\,\mathrm{m}$).

$h_{\mathrm{go}} = h_{\mathrm{st}} + v_{\max}/\kappa = 55\,\mathrm{m}$, $a_{\min} = -7\,\mathrm{m/s^2}$, $a_{\max} = 3\,\mathrm{m/s^2}$ which were used by the CAV's controller in the experiments, while $\tau \approx 0.6\,\mathrm{s}$ is the approximate time delay associated with communication and actuation through the powertrain.

Note that all nonlinearities in this model can be represented as the shifted and scaled counterpart of the saturation function

$$f(s) = \begin{cases} -1 & s < -1, \\ s & -1 \le s \le 1, \\ 1 & s > 1. \end{cases} \tag{3}$$

When incorporating this as activation function into neural network, we hope to learn the car following behavior of the CAV. In order to understand the role of the nonlinearities in (2) we also investigate an affine model where nonlinearities are neglected:

$$u(t) = \alpha\Big(\kappa(h(t) - h_{\mathrm{st}}) - v(t)\Big) + \beta\Big(v_{\mathrm{L}}(t) - v(t)\Big). \tag{4}$$

We use the velocity profile of the CHV and the initial condition of the CAV from real experimental data to simulate (1, 2) and (1, 4). This way, we create two simulation datasets (each of them containing 10 runs with different h^*). Thus, including the real experimental data itself, we have three datasets prepared for training, validation and testing, which are depicted for the run $h^* = 45\,\mathrm{m}$ in Fig. 2 (a,b). We test the networks and algorithms on the three datasets separately, using eight runs for training, one for validation ($h^* = 25\,\mathrm{m}$) and one for testing ($h^* = 35\,\mathrm{m}$).

3. Delayed Feed-forward Neural Networks (DFNNs)

In this section, we build delayed feed-forward neural networks to capture the car-following behavior of the CAV. We use the inputs h, v_{L} and v for the DFNN to predict the acceleration $\hat{a}$ of the CAV.

3.1. Data processing

We first normalize the inputs and output to scale them approximately between -1 and 1. In particular we define:

$$\tilde{h} = g_{\mathrm{h}}(h - \underline{h}) - 1, \qquad \tilde{v}_{\mathrm{L}} = g_{\mathrm{v}}(v_{\mathrm{L}} - \underline{v}) - 1, \qquad \tilde{v} = g_{\mathrm{v}}(v - \underline{v}) - 1, \qquad \tilde{a} = g_{\mathrm{a}}(a - \underline{a}) - 1, \tag{5}$$

and denote the scaled predicted acceleration as $\hat{\tilde{a}}$. The scaling factors are defined as $g_{\mathrm{h}} = 2/(\overline{h} - \underline{h})$, $g_{\mathrm{v}} = 2/(\overline{v} - \underline{v})$, $g_{\mathrm{a}} = 2/(\overline{a} - \underline{a})$. We use $\overline{h} = 150\,\mathrm{m}$, $\underline{h} = 0\,\mathrm{m}$, $\overline{v} = 30\,\mathrm{m/s}$, $\underline{v} = 0\,\mathrm{m/s}$, $\overline{a} = 3\,\mathrm{m/s^2}$, $\underline{a} = -7\,\mathrm{m/s^2}$ based on the features of the data and the saturation functions defined above.

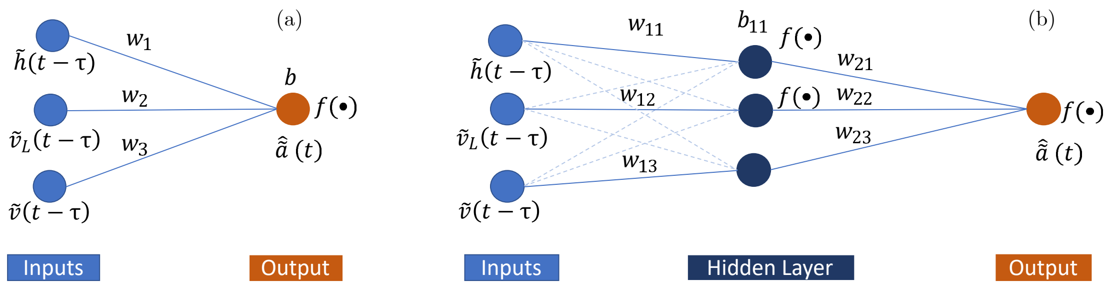

Figure 3: Structures of two DFNNs used in the paper: (a) no-hidden-layer DFNN; (b) one-hidden-layer DFNN.

3.2. DFNN structures

We propose two simple neural networks with delay using the saturation function (3) as the activation function; see Fig. 3. The simplified network with no hidden layer (panel (a)) and the one with one hidden layer (panel (b)) correspond to (1, 4) and (1, 2), respectively.

Since the data is available in discrete time, we define the sampled quantities as $y^j = y(j\Delta t)$ and $y^{j-\sigma} = y\big((j-\sigma)\Delta t\big)$, where the delay is $\tau = \sigma\Delta t$. The no-hidden-layer DFNN is given by

$$s^j = W x^{j-\sigma} + b, \qquad \hat{\tilde{a}}^j = f(s^j), \tag{6}$$

where the weights are collected by $W = [w_1, w_2, w_3]$ and b is a scalar bias. We can rewrite (6) as

$$\begin{aligned}\hat{a}(t) = \frac{1}{g_{\rm a}} f\Bigg(& w_1\Big(g_{\rm h}\Big(h(t-\tau) - \underline{h}\Big) - 1\Big) + w_2\Big(g_v\Big(v_{\rm L}(t-\tau) - \underline{v}\Big) - 1\Big)\\ &+ w_3\Big(g_v\Big(v(t-\tau) - \underline{v}\Big) - 1\Big) + b\Bigg) + \frac{1}{g_a} + \underline{a},\end{aligned} \tag{7}$$

which is equivalent to (1, 4) when $\underline{v} = 0$, $\overline{v} = v_{\max}$, $\underline{a} = a_{\min}$, and $\overline{a} = a_{\max}$. Then we can interpret the weights and bias of the DFNN as

$$\begin{aligned}&\alpha = -(w_3 + w_2)\frac{g_{\rm v}}{g_{\rm a}}, \qquad \beta = w_2\frac{g_{\rm v}}{g_{\rm a}}, \qquad \kappa = -\frac{w_1}{w_3 + w_2}\frac{g_{\rm h}}{g_{\rm v}},\\ &h_{\rm st} = \big(w_1(g_{\rm h}\underline{h} + 1) + w_2(g_{\rm v}\underline{v} + 1) + w_3(g_{\rm v}\underline{v} + 1) - (g_a\underline{a} + 1) - b\big)\frac{1}{g_h w_1}.\end{aligned} \tag{8}$$

Similarly, the formulas of the one-hidden-layer DFNN are written as

$$s_1^j = W_1 x_1^{j-\sigma} + b_1, \qquad x_2^j = F(s_1^j), \qquad s_2^j = W_2 x_2^j + b_2, \qquad \hat{\tilde{a}}^j = f(s_2^j), \tag{9}$$

where $W_1 = {\rm diag}[w_{11}, w_{12}, w_{13}]$, $b_1 = [b_{11}, 0, 0]^T$, $W_2 = [w_{21}, w_{22}, w_{23}]$, $b_2 = 0$, and $F(\cdot)$ means that $f(\cdot)$ is only applied on the first two neurons. Again by rewriting (9) akin to (1, 2), we obtain

$$\begin{aligned}\hat{a}(t) = \frac{1}{g_{\rm a}} f\Bigg(& w_{21} f\Big(w_{11}\Big(g_{\rm h}\big(h(t-\tau) - \underline{h}\big) - 1\Big) + b_{11}\Big) + w_{22} f\Big(w_{12}\Big(g_{\rm v}\big(v_{\rm L}(t-\tau) - \underline{v}\big) - 1\Big)\Big)\\ &+ w_{23}w_{13}\Big(g_{\rm v}\Big(v(t-\tau) - \underline{v}\Big) - 1\Big)\Bigg) + \frac{1}{g_{\rm a}} + \underline{a},\end{aligned} \tag{10}$$

and the parameter equivalency is given by

$$\alpha = -(w_{23}w_{13} + w_{22}w_{12})\frac{g_\mathrm{v}}{g_\mathrm{a}}, \qquad \beta = w_{22}w_{12}\frac{g_\mathrm{v}}{g_\mathrm{a}}, \qquad \kappa = -\frac{w_{21}w_{11}}{w_{23}w_{13} + w_{22}w_{12}}\frac{g_\mathrm{h}}{g_\mathrm{v}}, \tag{11}$$
$$h_\mathrm{st} = \big(w_{21}w_{11}(g_\mathrm{h}\underline{h}+1)+w_{22}w_{12}(g_\mathrm{v}\underline{v}+1)+w_{23}w_{13}(g_\mathrm{v}\underline{v}+1)-(1+g_a\underline{a})-b_{11}w_{21}\big)\frac{1}{g_\mathrm{h}w_{21}w_{11}}.$$

3.3. Cost function and training algorithm for fixed delay

We use the mean square error between scaled training acceleration $\tilde{a}$ and scaled predicted acceleration $\hat{\tilde{a}}$ as cost function, i.e,

$$\widetilde{E} = \frac{1}{N-\sigma}\sum_{j=\sigma+1}^{N}(\tilde{a}^j - \hat{\tilde{a}}^j)^2, \tag{12}$$

where N is the number of data points. To obtain a physically meaningful quantity we define $E = \sqrt{\widetilde{E}}/g_a$ that has the unit $\mathrm{m/s^2}$. The scaled error $\widetilde{E}$ is used for training and the error E is used in the plots below. For a fixed delay, gradient descent can be used to train the weights and biases in the network. Using the learning rate η, the weights and biases in iteration $n+1$ can be expressed as

$$W_i(n+1) = W_i(n) - \eta\frac{\partial\widetilde{E}}{\partial W_i}(n), \qquad b_i(n+1) = b_i(n) - \eta\frac{\partial\widetilde{E}}{\partial b_i}(n). \tag{13}$$

For the no-hidden-layer DFNN we have

$$\frac{\partial\widetilde{E}}{\partial W} = \sum_{j=\sigma+1}^{N}\frac{-2(\tilde{a}^j - \hat{\tilde{a}}^j)}{N-\sigma}\frac{\partial f(s^j)}{\partial s^j}\frac{\partial s^j}{\partial W}, \qquad \frac{\partial\widetilde{E}}{\partial b} = \sum_{j=\sigma+1}^{N}\frac{-2(\tilde{a}^j - \hat{\tilde{a}}^j)}{N-\sigma}\frac{\partial f(s^j)}{\partial s^j}, \tag{14}$$

while for the one-hidden-layer DFNN we obtain

$$\frac{\partial\widetilde{E}}{\partial W_2} = \sum_{j=\sigma+1}^{N}\frac{-2(\tilde{a}^j - \hat{\tilde{a}}^j)}{N-\sigma}\frac{\partial f(s_2^j)}{\partial s_2^j}\frac{\partial s_2^j}{\partial W_2}, \quad \frac{\partial\widetilde{E}}{\partial b_1} = \sum_{j=\sigma+1}^{N}\frac{-2(\tilde{a}^j - \hat{\tilde{a}}^j)}{N-\sigma}\frac{\partial f(s_2^j)}{\partial s_2^j}W_2\frac{\partial F(s_1^j)}{\partial s_1^j}\frac{\partial s_1^j}{\partial b_1},$$
$$\frac{\partial\widetilde{E}}{\partial W_1} = \sum_{j=\sigma+1}^{N}\frac{-2(\tilde{a}^j - \hat{\tilde{a}}^j)}{N-\sigma}\frac{\partial f(s_2^j)}{\partial s_2^j}W_2\frac{\partial F(s_1^j)}{\partial s_1^j}\frac{\partial s_1^j}{\partial W_1}, \tag{15}$$

where we write 0 in the gradient if the corresponding weight or bias is fixed to 0, and we interpret row and column vectors such that compatibility is kept with (13).

We stop the training process when the validation error starts to increase and we simulate the car-following dynamics using the acceleration provided by trained network (using the initial states and the CHV's velocity from the testing data). We use the training error, equivalent parameters and simulation results in acceleration, velocity and headway to evaluate the network performance.

3.4. Training results and error-delay relationship

We train the networks for various fixed delay values. For each delay, we train the network ten times and plot the training error E at the end of the training process.

The corresponding errors are depicted as a function of the delay in Fig. 4 for the three different sets of data and $\eta = 0.1$. Panel (a) shows the results for the no-hidden-layer DFNN. This network is able to reproduce the behavior of the affine simulation data given by (1, 4) for $\tau = 0.6\,\mathrm{s}$ due to

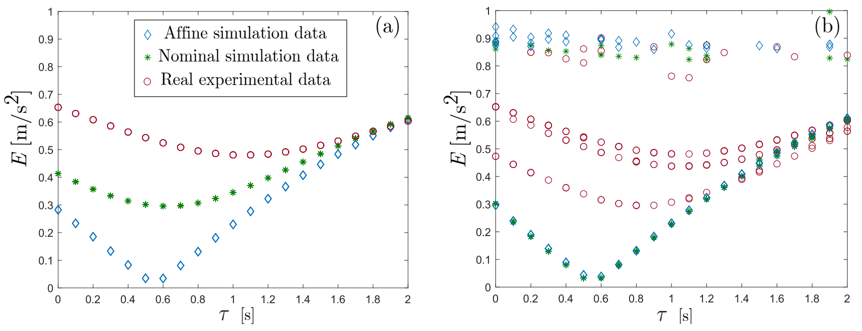

Figure 4: Error-delay relationships for: (a) no-hidden-layer DFNN; (b) one-hidden-layer DFNN.

Datasets	τ	No-hidden-layer					One-hidden-layer				
		E	α	β	κ	h_{st}	E	α	β	κ	h_{st}
Affine Simulation Data	0	0.28	0.44	0.44	0.61	5.29	0.30	0.41	0.42	0.61	5.37
	0.6	0.03	0.39	0.50	0.60	4.98	0.04	0.39	0.50	0.60	4.98
Nominal Simulation Data	0	0.41	0.31	0.46	0.60	5.12	0.27	0.42	0.45	0.61	5.29
	0.6	0.30	0.24	0.51	0.59	4.53	0.03	0.39	0.50	0.60	4.98
Real Experimental Data	0	0.65	0.08	0.26	0.51	2.76	0.47	0.25	0.25	0.59	5.12
	0.8	0.50	0.09	0.34	0.48	0.40	0.29	0.23	0.34	0.57	4.45

Table 1: Training results of two DFNNs with different delays and data

the equivalency explained above; see blue diamonds. Once it is used for the nominal simulation data given by (1, 2), the error increases due to the nonlinearities but the minimum is still at the delay value used for the simulations; see green stars. In case of the real experimental data the trends are still kept. However, the error further increases and the minimum occurs for higher delay; see red circles. Panel (b) depicts the results for the one-hidden-layer DFNN. This network is able to reproduce the affine simulation data as well as the nominal simulation data. However, the algorithm may be trapped in local minima as shown by the symbols at the top of the panel. With the help of embedding suitable nonlinearities, the network can also achieve better performance on the real experimental data but local minima may still show up.

The equivalent parameters are shown in Table 1, comparing the case of the appropriately chosen delay to the case of no delay. The delay does not affect κ much but affects α and β related to the transient dynamics. Meanwhile, the structure affects α and β significantly. The no-hidden-layer DFNN with appropriately chosen delay accurately captures the model parameters in case of the affine simulation data, while the one-hidden-layer DFNN has higher parameter accuracy for the nominal simulation data and real experimental data.

4. Delay-searching Algorithm

The algorithm presented in the previous section is clearly not scalable in the case of multiple delays. This demands an algorithm that allows one to train the delays in the system.

4.1. Gradient descent method for delay-searching

Again we apply gradient decent method for all parameters that now also include the delay σ. We can get the gradient information with respect to σ for the no-hidden-layer DFNN as

$$\frac{\partial \widetilde{E}}{\partial \sigma} = \sum_{k=\sigma+1}^{N} \frac{\partial \widetilde{E}}{\partial \hat{\tilde{a}}^k} \frac{\partial \hat{\tilde{a}}^k}{\partial s^k} \frac{\partial s^k}{\partial x^{k-\sigma}} \frac{\partial x^{k-\sigma}}{\partial \sigma} = \sum_{j=\sigma+1}^{N} \frac{-2(\tilde{a}^j - \hat{\tilde{a}}^j)}{N-\sigma} \frac{\partial f(s^j)}{\partial s^j} W(-\dot{x}^{j-\sigma}), \qquad (16)$$

and for the one-hidden-layer DFNN as

$$\frac{\partial \widetilde{E}}{\partial \sigma} = \sum_{j=\sigma+1}^{N} \frac{-2(\tilde{a}^j - \hat{\tilde{a}}^j)}{N-\sigma} \frac{\partial f(s_2^j)}{\partial s_2^j} W_2 \frac{\partial F(s_1^j)}{\partial s_1^j} W_1(-\dot{x}_1^{j-\sigma}), \qquad (17)$$

where $\dot{x}^{j-\sigma} = x^{j-\sigma+1} - x^{j-\sigma}$. Note that σ is a positive integer, and thus, we use

$$\sigma(n+1) = \max\left(\text{round}\left(\sigma(n) - \eta_\sigma \frac{\partial \widetilde{E}}{\partial \sigma}(n)\right), 0\right) \qquad (18)$$

at each iteration, where the learning rate η_σ is different from the one used for the weights and biases. Note that the length of the data in each run is much longer than the delay, so we neglect the effect of the normalizing term $N-\sigma$ in the gradient. The delay-searching algorithm is summarized as Algorithm 1.

Algorithm 1 Gradient descent with delay searching

```
Data: Training data and validation data
Result: Learn $W_i$, $b_i$ and $\sigma$ from data
normalize data and get derivatives
set maximum iteration number, maxiter
initialize $W_i(1)$, $b_i(1)$ (uniform distribution over $[0,1]$) and $\sigma(1) = 0$
for $n$ = 1,..., maxiter do
    shift data, calculate training error $\widetilde{E}_{\mathrm{tr}}(n)$ and validation error $\widetilde{E}_{\mathrm{va}}(n)$
    if $n > 100$ and $\widetilde{E}_{\mathrm{va}}(n-100) \leq \widetilde{E}_{\mathrm{va}}(n)$ then
        break loop
    else
        get gradient information and update parameters $W_i(n+1)$, $b_i(n+1)$ and $\sigma(n+1)$
    end
end
let $m = \arg\min_{n \in 1,2,\ldots,\mathrm{maxiter}} \widetilde{E}_{\mathrm{va}}(n)$, then $W_i = W_i(m)$, $b_i = b_i(m)$, $\sigma = \sigma(m)$
```

4.2. Implementation and results

To test the delay-searching algorithm, we train the two DFNNs on the three sets of data. The learning rates in the training process are set as $\eta = 0.1$ and $\eta_\sigma = 2000$. We show how the training error decreases and how the delay changes with the number of iterations in Fig. 5 (a) and (b), respectively. Observe that for each jump in the delay, the training error decreases discontinuously. Note that the delay-searching algorithm is able to automatically find the delay value associated with the minimum error in Fig. 4.

Finally, we evaluate the learning performance by simulating the equivalent dynamical system given by the parameters of the trained network. We define the simulation error similarly to (12) as $E_{\mathrm{s}} = \sqrt{\sum_{j=\sigma+1}^{N} (a^j - \hat{a}_{\mathrm{s}}^j)^2/(N-\sigma)}$, where $\hat{a}_{\mathrm{s}}$ is the acceleration in the simulation. The simulation results for the testing data $h^* = 35\,\mathrm{m}$ are shown in Fig. 6. The no-hidden-layer DFNN learns the correct delay for the two sets of simulation data, but it fails to capture the dynamics in the first few seconds where the nonlinearities are activated in the nominal simulation data. The one-hidden-layer DFNN learns the delay with minimum training error, and it also learns the behavior in

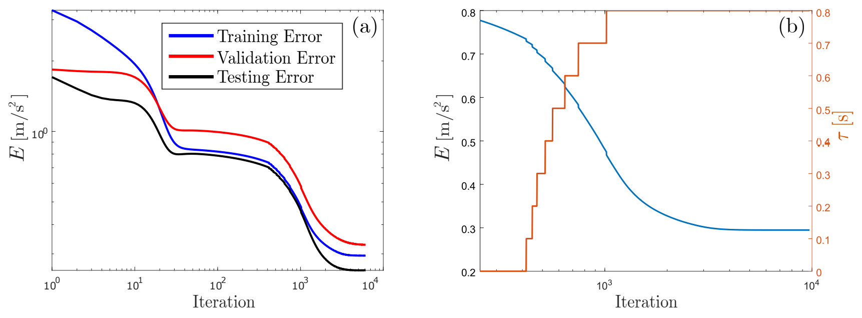

Figure 5: Error in one-hidden-layer DFNN trained on real experimental dataset: (a) three errors as a function of iteration; (b) the training error and the delay as a function of iteration.

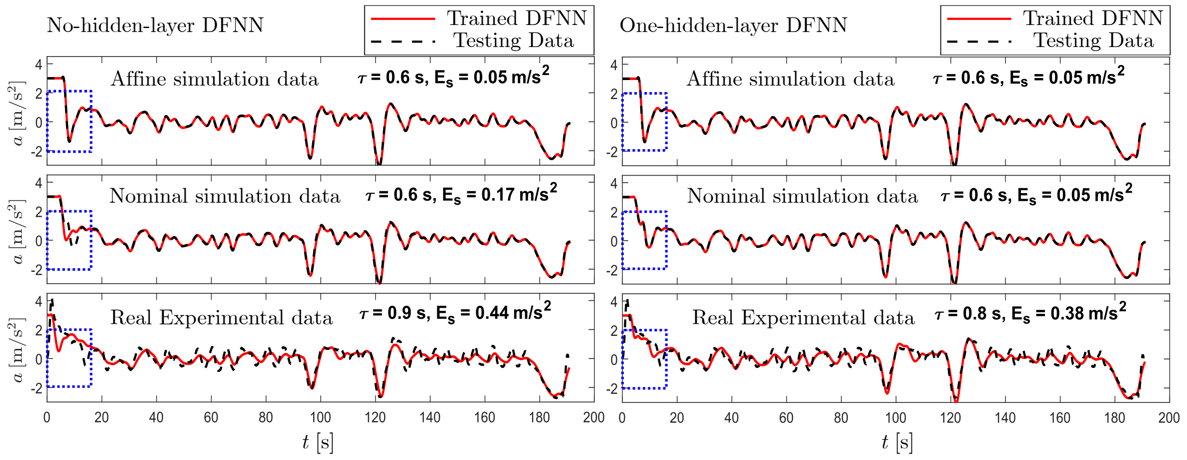

Figure 6: Simulation results for the trained DFNNs

the first few seconds in nominal simulation data due to the saturation functions in the hidden layer. The one-hidden-layer DFNN achieves a much smaller the error when trained on experimental data, though the error is larger compared to when it is trained on simulation data. This is likely related to the unmodeled dynamics of the vehicle in (1).

5. Summary

We have built a connection between feed-forward neural networks and delayed dynamical systems by developing equivalent delayed feed-forward neural networks (DFNNs). We have explained the relationship between training error and delay during learning a car-following model. Based on this relationship, we have proposed a delay-searching algorithm to learn the delay together with the other parameters. We have presented training and simulation results to show successful implementations of this algorithm on two different DFNNs using the car-following data of a connected automated vehicle.

In the future, we will extend the algorithm to cases with multiple delays in larger neural network. Finally we will explore delay searching in systems integrating physics based models with data driven based models Cheng et al. (2019); Zhou et al. (2017); Shi et al. (2019). We will also investigate recurrent neural networks (RNNs), which are suitable for sequential inputs with dynamics.

References

S. S. Avedisov, G. Bansal, A. K. Kiss, and G. Orosz. Experimental verification platform for connected vehicle networks. In *Proceedings of the 21st International Conference on Intelligent Transportation Systems*, pages 818–823, Maui, HI, USA, 2018.

R. Cheng, A. Verma, G. Orosz, S. Chaudhuri, Y. Yue, and J. W. Burdick. Control regularization for reduced variance reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, Long Beach, CA, USA, 2019.

B. de Vries and J. C. Principe. A theory for neural networks with time delays. In *Proceedings of Advances in Neural Information Processing Systems 3*, pages 162–168, Denver, CO, USA, 1990.

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. *Transportation Research Part C*, 91:335–352, 2018.

K. J. Lang, A. H. Waibel, and G. E. Hinton. A time-delay neural network architecture for isolated word recognition. 3(1):23–43, 1990.

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. *Proceedings of the Royal Society A*, 473(2207):20170502, 2017.

G. Orosz, R. E. Wilson, R. Szalai, and G. Stepan. Exciting traffic jams: Nonlinear phenomena behind traffic jam formation on highways. *Physical Review E*, 80(4):046205, 2009.

S. Panwai and H. Dia. Neural agent car-following models. *IEEE Transactions on Intelligent Transportation Systems*, 8(1):60–70, 2007.

G. Shi, X. Shi, M. O'Connell, R. Yu, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S. Chung. Neural lander: Stable drone landing control using learned dynamics. In *Proceedings of the International Conference on Robotics and Automation*, Montreal, QC, Canada, 2019.

D. W. Tank and J. J. Hopfield. Concentrating information in time. 84:1896–1900, 1987.

X. Wang, R. Jiang, L. Li, Y. Lin, X. Zheng, and F.-Y. Wang. Capturing car-following behaviors by deep learning. *IEEE Transactions on Intelligent Transportation Systems*, 19(3):910–920, 2018.

C. Wu, K. Parvate, N. Kheterpal, L. Dickstein, A. Mehta, E. Vinitsky, and A. M. Bayen. Framework for control and deep reinforcement learning in traffic. In *Proceedings of the 20th International Conference on Intelligent Transportation Systems*, Yokohama, Japan, 2017.

F. Wu and D. B. Work. Connections between classical car following models and artificial neural networks. In *Proceedings of the 21st International Conference on Intelligent Transportation Systems*, pages 3191–3198, Maui, HI, USA, 2018.

L. Zhang and G. Orosz. Motif-based design for connected vehicle systems in presence of heterogeneous connectivity structures and time delays. *IEEE Transactions on Intelligent Transportation Systems*, 17(6):1638–1651, 2016.

S. Zhou, M. K. Helwa, and A. P. Schoellig. Design of deep neural networks as add-on blocks for improving impromptu trajectory tracking. In *Proceedings of the 56th Annual Conference on Decision and Control*, Melbourne, VIC, Australia, 2017.