A Novel Neural Filter to Improve Accuracy of Neural Network Models of Dynamic Systems

Parham Oveissi, Turibius Rozario, Ankit Goel

***Abstract*— The application of neural networks in modeling dynamic systems has become prominent due to their ability to estimate complex nonlinear functions. Despite their effectiveness, neural networks face challenges in long-term predictions, where the prediction error diverges over time, thus degrading their accuracy. This paper presents a *neural filter* to enhance the accuracy of long-term state predictions of neural network-based models of dynamic systems. Motivated by the extended Kalman filter, the neural filter combines the neural network state predictions with the measurements from the physical system to improve the estimated state's accuracy. The neural filter's improvements in prediction accuracy are demonstrated through applications to four nonlinear dynamical systems. Numerical experiments show that the neural filter significantly improves prediction accuracy and bounds the state estimate covariance, outperforming the neural network predictions. Furthermore, it is also shown that the accuracy of a poorly trained neural network model can be improved to the same level as that of an adequately trained neural network model, potentially decreasing the training cost and required data to train a neural network.**

keywords: neural networks, dynamical systems, neural network modeling, state estimation.

I. Introduction

The use of neural networks has surged due to advances in computational power and extensive research within the machine learning community. Neural networks, with their inherently nonlinear activation functions, are adept at estimating complex nonlinear functions [1], [2], making them highly suitable for modeling dynamic systems and predicting future states. These capabilities have broad applications, such as modeling nonlinear oscillators [3], controlling nonlinear dynamical systems [4], [5], predicting weather phenomena [6], and enhancing medical diagnostics [7]. Modeling dynamic systems is crucial across various engineering and scientific disciplines. Neural networks offer a powerful alternative to traditional parametric methods by learning complex system behaviors directly from data. For instance, a procedure developed for identifying nonlinear dynamic systems using artificial neural networks demonstrated the capability to effectively predict the response of a damped Duffing oscillator under varying excitations, highlighting the neural network's high-fidelity modeling potential [3]. [4] introduced models for nonlinear dynamical systems identification and control, emphasizing the feasibility of neural networks in handling more complex systems and introducing a combined linear-neural network controller structure. More recently, universal approximation property of neural networks has been leveraged to learn functions from small datasets. A significant reduction in generalization error and high-order error convergence in dynamic systems and PDEs was reported in [8]. Physics-informed neural networks (PINNs) that integrate physical laws into neural network training to ensure that the models adhere to underlying physical principles have also been explored [9].

Despite neural networks' impressive capabilities, a significant challenge arises in the context of long-term predictions, where the error between the true system behavior and the neural network approximation tends to accumulate over time. To address the issue of long-term prediction errors, various approaches have been proposed. A technique based on regularization to improve robustness and reduce error accumulation was investigated in [10]. Recurrent neural networks have been investigated to capture temporal dependencies to reduce error accumulation for both interpolation and extrapolation tasks [11], [12].

The key reason for the prediction error's divergence is that the design and training of the neural network-based model do not stabilize the state error dynamics. Motivated by the Kalman filter [13], where a feedback signal from the physical system stabilizes the error dynamics, this paper presents a *neural filter* to arrest the divergence of the error in the neural network state predictions. The main contribution of this work is thus the extension of the extended Kalman filter [14], [15] to the neural filter and the demonstration of the improvement in the long-term accuracy of the neural filter state predictions.

The paper is organized as follows. Section II formulates the state-estimation problem and presents the neural filter. Section III applies the neural filter to estimate

Parham Oveissi is a graduate student in the Department of Mechanical Engineering, University of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250. parhamo1@umbc.edu

Turibius Rozario is an undergraduate student and a Meyerhoff Scholar in the Department of Mechanical Engineering, University of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250. s175@umbc.edu

Ankit Goel is an Assistant Professor in the Department of Mechanical Engineering, University of Maryland, Baltimore County,1000 Hilltop Circle, Baltimore, MD 21250. ankgoel@umbc.edu

states in four typical dynamical systems. Finally, Section IV summarizes the paper.

II. Problem Formulation

This section formulates notation and terminology associated with the state-estimation problem and presents the *neural filter*. Consider the nonlinear system

$$x_{k+1} = f\big(x_k, u_k\big) + w_k, \tag{1}$$
$$y_k = g\big(x_k\big) + v_k, \tag{2}$$

where, for all $k \geq 0$, x_k is the state, u_k is the input to the system, y_k is the measured output of the system, f, g, are real-valued vector functions, $w_k \sim \mathcal{N}(0, Q_k)$ is the process noise, $v_k \sim \mathcal{N}(0, R_k)$ is the measurement noise, and Q_k and R_k are process and measurement covariance matrices, respectively. The objective is to propagate the system states x_k by approximating the function f with a neural network, and then correct these predictions using the available measurements y_k from the system. In this work, we assume that the function g is known.

A. Neural Network Model

Consider a dynamic system

$$\lambda x(t) = \mathcal{F}(x(t), u(t)), \tag{3}$$

where x and u are the state of the system and the input to the system, respectively, $\mathcal{F}$ is the dynamics of the system, and the operator λ is either a derivative or a forward-shift operator, that is, in continuous-time systems, $\lambda x = \dot{x}$ and, in discrete-time systems, $\lambda x(t) = x(t+1)$. The objective is approximate the state of (3) with a discrete dynamic system

$$\hat{x}_{k+1} = NN(\hat{x}_k, u_k), \tag{4}$$

where $\hat{x}_k$ is the estimated state of the system and NN denotes a neural-network approximation of the system, which is constructed as shown below. Note that k is the iteration number and is related to continuous time t as $t = kT_{\rm s}$, where $T_{\rm s}$ is the discretization timestep.

To generate the training data to construct the neural-network model, we solve (3) from $t = 0$ to $t = T_{\rm s}$. For example, in the case of continuous time system, note that

$$x(T_{\rm s}) = x(0) + \int_0^{T_{\rm s}} \mathcal{F}(x(\tau), u(\tau)){\rm d}\tau. \tag{5}$$

The training data consists of randomly generated samples of $x(0)$ and $u(0)$ and the corresponding $x(T_{\rm s})$, computed using (5). In this work, we use MATLAB's ode45 routine to compute $x(T_{\rm s})$. The trained neural network, denoted by $NN(x)$, can thus be used to propagate the state at a time instant t to $t + T_{\rm s}$ using (4).

In this work, we use the MATLAB routine dlnetwork to define the neural network architecture. To train the neural-network model NN with the training data, which includes the input data $\{x(0), u(0)\}$ and the output data $\{x(T)\}$, we employ the Adam optimizer using the MATLAB routine adamupdate in a mini-batch training setup. A small portion of the training data is reserved as a validation dataset throughout the training process.

B. Neural Filter

Let $NN(x, u)$ be a neural network approximation of the function f. Then, the *neural filter* is

$$\hat{x}_{k+1|k} = NN\big(\hat{x}_{k|k}, u_k\big), \tag{6}$$
$$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + K_{k+1}\big(y_{k+1} - g(\hat{x}_{k+1|k})\big), \tag{7}$$

where $\hat{x}_{k+1|k}$ is the *prior estimate* at step $k+1$, $\hat{x}_{k+1|k+1}$ is the *posterior estimate* at step $k+1$, and the *neural filtering* correction gain K_{k+1} is given by

$$K_{k+1} = P_{k+1|k} C_{k+1}^{\rm T} (C_{k+1} P_{k+1|k} C_{k+1}^{\rm T} + R_{k+1})^{-1}, \tag{8}$$

where the *prior covariance* matrix $P_{k+1|k}$ is given by

$$P_{k+1|k} = A_k P_{k|k} A_k^{\rm T} + Q_k, \tag{9}$$

and the state transition matrix A_k and the measurement matrix C_k are given by

$$A_k = \left.\frac{\partial NN}{\partial x}\right|_{\hat{x}_{k|k}, u_k}, \quad C_k = \left.\frac{\partial g}{\partial x}\right|_{\hat{x}_{k|k}, u_k}. \tag{10}$$

The computation of the Jacobian of a neural network is described in Appendix V-A.

Finally, the *posterior covariance* matrix $P_{k+1|k+1}$ is given by

$$P_{k+1|k+1} = (I - K_{k+1} C_{k+1}) P_{k+1|k} \tag{11}$$

Remark 2.1: Note that, since the *neural filter* is motivated by the extended Kalman filter, the matrices $P_{k+1|k}$ and $P_{k+1|k+1}$ are similar to the *prior* and *posterior covariance* matrices in EKF. Therefore,

$$P_{k+1|k} \approx \mathbb{E}[e_{k+1|k} e_{k+1|k}^{\rm T}], \tag{12}$$
$$P_{k+1|k+1} \approx \mathbb{E}[e_{k+1|k+1} e_{k+1|k+1}^{\rm T}], \tag{13}$$

where the *prior error* $e_{k+1|k}$ and the *posterior error* $e_{k+1|k+1}$ are defined as

$$e_{k+1|k} \overset{\triangle}{=} x_{k+1} - \hat{x}_{k+1|k}, \tag{14}$$
$$e_{k+1|k+1} \overset{\triangle}{=} x_{k+1} - \hat{x}_{k+1|k+1}. \tag{15}$$

Remark 2.2: Since $P_{k|k}$ is the covariance of the posterior state-estimation error, that is, $\mathbb{E}[e_{k|k} e_{k|k}^{\rm T},]$ the trace of $P_{k|k}$ is approximately equal to the square of the 2-norm of the posterior state-estimation error, that is, $\mathbb{E}[e_{k|k}^{\rm T} e_{k|k}.]$

Figure 1 shows the neural filter architecture, which incorporates the available measurements from the physical system to improve the prediction accuracy of the neural network-based model of the physical system.

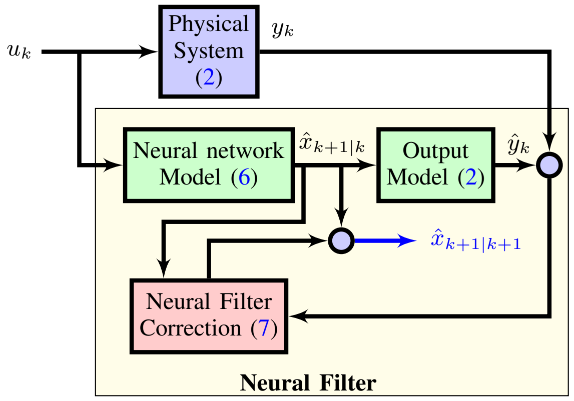

Fig. 1: Neural filter architecture.

III. Numerical Experiments

This section presents several case studies to demonstrate the performance of the neural filter. In particular, we consider the simple pendulum, the Van der Pol oscillator, the Lorenz system, and the double pendulum system to demonstrate that the neural filter maintains prediction accuracy over a long horizon. Note that the Lorenz system and the double pendulum system are chaotic, making long-term predictions especially challenging due to their sensitivity to initial conditions, where small changes can lead to vastly different behaviors [16].

A. Simple Pendulum

Consider the simple pendulum

$$\ell\ddot{\theta} + g\sin\theta = 0. \tag{16}$$

Defining $x \triangleq \begin{bmatrix}\theta & \dot{\theta}\end{bmatrix}^{\mathrm{T}}$, the simple pendulum (16) can be written in the state-space form (3), where

$$f(x) \triangleq \begin{bmatrix} x_2 \\ -\frac{g}{\ell}\sin(x_1) \end{bmatrix}. \tag{17}$$

The output is assumed to be a noisy measurement of the angle θ, that is,

$$y_k = \begin{bmatrix}1 & 0\end{bmatrix} x_k + v_k, \tag{18}$$

where $v_k \sim \mathcal{N}(0, \sigma_v^2 I_2)$ represents zero-mean Gaussian noise. In this example, we set $\sigma_v = 0.1$.

The training data consists of 15,000 samples of $x(0) \in \begin{bmatrix}\mathcal{U}[-\pi/2, \pi/2] \\ \mathcal{U}[-5, 5]\end{bmatrix}$. Using MATLAB's ode45 routine, $x(0.1)$ is computed according to (5). In this work, we use 80 % of the data to train the model, and the remaining 20 % are used for validation during the training process.

The neural network architecture used to approximate the simple pendulum is shown in Figure 2. In particular, the neural network NN_1 consists of an input layer with a dimension of 2, a single hidden layer containing 10 neurons, and an output layer with a dimension of 2. The hidden layer uses the rectified linear unit (ReLU) activation function, while the output layer uses a linear activation function. The Adam optimizer is used for training the neural network. Training is performed with a batch size of 32, and validation is conducted every 30 iterations. To assess the robustness of the neural filter against variations in neural network complexity, we also consider a significantly simpler architecture NN_2. This network consists of an input layer with a dimension of 2, a single hidden layer with 2 neurons, and an output layer with a dimension of 2. The hidden layer employs the ReLU activation function, while the output layer utilizes a linear activation function. Figure 3 shows the smoothed training and validation loss on a logarithmic scale during the training process for both trained networks NN_1 and NN_2.

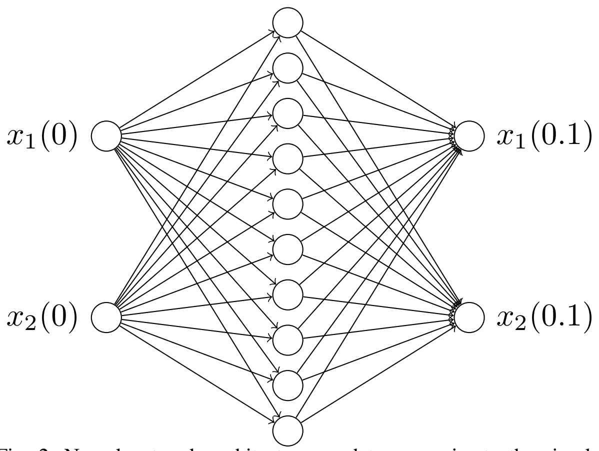

Fig. 2: Neural network architecture used to approximate the simple pendulum system.

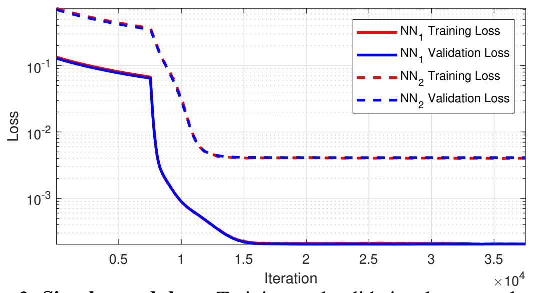

Fig. 3: **Simple pendulum**. Training and validation loss on a logarithmic scale to approximate the simple pendulum system.

Next, the trained neural networks and the corresponding neural filters are used to predict the state of the pendulum system. In particular, we set $x(0) = \begin{bmatrix} \frac{\pi}{3} & 1 \end{bmatrix}^{\rm T}$. In both of the neural filter, we set $P(0) = 10^{-4} \times I_2$, where I_2 is the 2 by 2 identity matrix and $\hat{x}_{0|0} = 0$. Note that $\hat{x}_{0|0} = 0$ reflects the absence of knowledge of the state of the dynamic system in the neural filter. However, since the neural network models NN_1 and NN_2 have no external correction, we initialize them with the same initial conditions as the true system, that is, $\hat{x}_0 = 0$ in the neural network model. Figures 4 and 5 show the predicted states using the neural network (subfigures in the left column) and the corresponding neural filter (subfigures in the right column) using NN_1 and NN_2, respectively.

Note that, in both cases, the state predictions using the neural network degrade over time, as shown by the increasing state error norm $\|e_{k|k}\|$ and the trace of the corresponding state covariance $\operatorname{tr} P_{k|k}$. On the other hand, the state predictions using the neural filter remain accurate over time, as shown by the bounded state error norm $\|e_{k|k}\|$ and the trace of the corresponding state covariance $\operatorname{tr} P_{k|k}$. Furthermore, the NN_1 and NN_2 estimates diverge even with the correct initialization, whereas the neural filter estimates converge despite the complete lack of the system's initial state. *Although* NN_2 *is a much simpler model and shows noticeably poorer performance than* NN_1 *in open-loop state estimation, the neural filter in both instances operates quite similarly.*

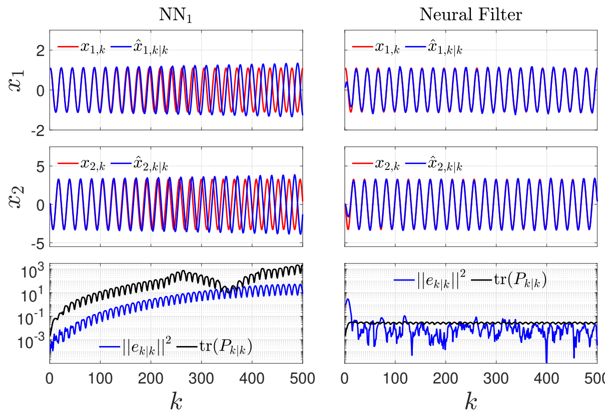

Fig. 4: **Simple pendulum.** State predictions using the neural network and the neural filter using NN_1.

B. Van Der Pol

Consider the Van Der Pol oscillator

$$\ddot{q} - \mu(1 - q^2)\dot{q} + q = 0. \tag{19}$$

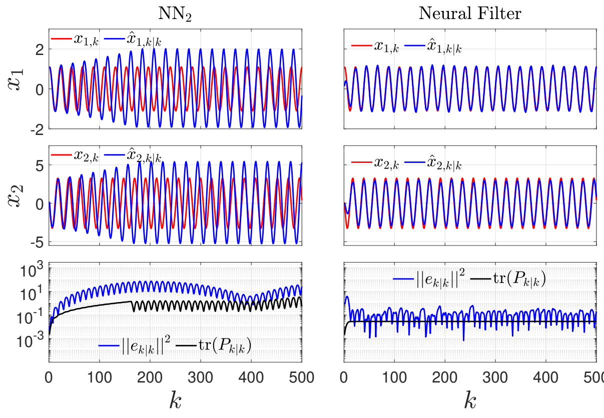

Fig. 5: **Simple pendulum.** State predictions using the neural network and the neural filter using NN_2.

Defining $x \triangleq \begin{bmatrix} q & \dot{q} \end{bmatrix}^{\rm T}$, the Van der Pol oscillator (19) can be written in the state-space form (3), where

$$f(x) \triangleq \begin{bmatrix} x_2 \\ \mu(1 - {x_1}^2)x_2 - x_1 \end{bmatrix}. \tag{20}$$

The measurement model for this example is defined as follows

$$y_k = \begin{bmatrix} 1 & 0 \end{bmatrix} x_k + v_k, \tag{21}$$

where $v_k \sim \mathcal{N}(0, \sigma_v^2 I_2)$ represents zero-mean Gaussian noise. For this example, we specify $\sigma_v = 0.1$.

The training data consists of 30,000 samples of $x(0) \in \begin{bmatrix} \mathcal{U}[-5, 5] \\ \mathcal{U}[-5, 5] \end{bmatrix}$. Using MATLAB's ode45 routine, $x(0.1)$ is computed according to (5). In this work, we use 80 % of the data to train the model, and the remaining 20 % are used for validation during the training process.

The neural network architecture used to approximate the Van der Pol oscillator is shown in Figure 6. In particular, the neural network consists of an input layer with a dimension of 2, two hidden layers containing 10 neurons each, and an output layer with a dimension of 2. Both hidden layers use the rectified linear unit (ReLU) activation function, while the output layer uses a linear activation function. The Adam optimizer is used for training the neural network. Training is performed with a batch size of 32, and validation is conducted every 30 iterations. Figure 7 shows the smoothed training and validation loss on logarithmic scale during the training process.

Next, the trained neural network and the neural filter are used to predict the state of the Van der Pol oscillator. In particular, we set $x(0) = \begin{bmatrix} 2 & 1 \end{bmatrix}^{\rm T}$. In the neural filter, we set $P(0) = 10^{-4} \times I_2$, where I_2 is the 2 by 2 identity matrix. Figure 8 shows the predicted states using the neural network (subfigures in the left column) and the neural filter (subfigures in the right column). Note that the state predictions using the neural network

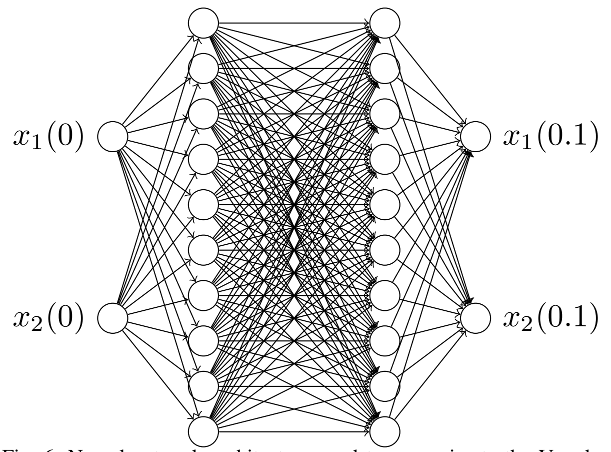

Fig. 6: Neural network architecture used to approximate the Van der Pol oscillator.

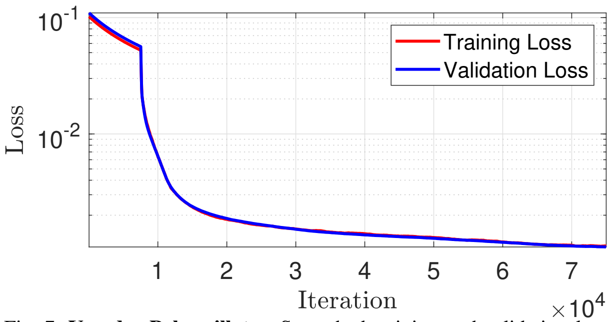

Fig. 7: **Van der Pol oscillator.** Smoothed training and validation loss on logarithmic scale during the training process.

degrade over time, as shown by the increasing state error norm $\|e_{k|k}\|$ and the trace of the corresponding state covariance $\operatorname{tr} P_{k|k}$. On the other hand, the state predictions using the neural filter remain accurate over time, as shown by the bounded state error norm $\|e_{k|k}\|$ and the trace of the corresponding state covariance $\operatorname{tr} P_{k|k}$.

C. Lorenz System

Consider the Lorenz system

$$\begin{aligned}\dot{w} &= \sigma(y - w),\\ \dot{y} &= w(\rho - z) - y,\\ \dot{z} &= wy - \beta z.\end{aligned} \tag{22}$$

Defining $x \triangleq \begin{bmatrix} w & y & z \end{bmatrix}^{\mathrm{T}}$, the Lorenz system (22) can be written in the state-space form (3), where

$$f(x) \triangleq \begin{bmatrix} \sigma(x_2 - x_1) \\ x_1(\rho - x_3) - x_2 \\ x_1 x_2 - \beta x_3 \end{bmatrix}. \tag{23}$$

The measurement model for this example is defined as follows

$$y_k = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} x_k + v_k, \tag{24}$$

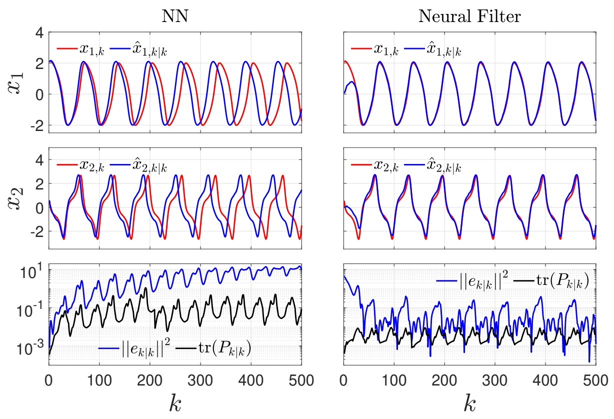

Fig. 8: **Van der Pol oscillator.** State predictions using the neural network and the neural filter.

where $v_k \sim \mathcal{N}(0, \sigma_v^2 I_3)$ represents zero-mean Gaussian noise. For this example, we specify $\sigma_v = 0.1$.

The training data consists of 100,000 samples of $x(0) \in \begin{bmatrix} \mathcal{U}[-15, 15] \\ \mathcal{U}[-15, 15] \\ \mathcal{U}[-15, 15] \end{bmatrix}$. Using MATLAB's ode45 routine, $x(0.01)$ is computed according to (5). In this work, we use 80 % of the data to train the model, and the remaining 20 % are used for validation during the training process.

The neural network architecture used to approximate the Lorenz system is shown in Figure 9. In particular, the neural network consists of an input layer with a dimension of 3, three hidden layers containing 10 neurons each, and an output layer with a dimension of 3. All the hidden layers use the rectified linear unit (ReLU) activation function, while the output layer uses a linear activation function. The Adam optimizer is used for training the neural network. Training is performed with a batch size of 32, and validation is conducted every 30 iterations. Figure 10 shows the smoothed training and validation loss on logarithmic scale during the training process.

Next, the trained neural network and the neural filter are used to predict the state of the Lorenz system. In particular, we set $x(0) = \begin{bmatrix} -6.13 & 1.78 & 1.67 \end{bmatrix}^{\mathrm{T}}$. In the neural filter, we set $P(0) = 10^{-4} \times I_3$, where I_3 is the 3 by 3 identity matrix. Figure 11 shows the predicted states using the neural network (subfigures in the left column) and the neural filter (subfigures in the right column). Note that the state predictions using the neural network degrade over time, as shown by the increasing state error norm $\|e_{k|k}\|$ and the trace of the corresponding state covariance $\operatorname{tr} P_{k|k}$. On the other hand, the state predictions using the neural filter remain accurate over time, as shown by the bounded state error norm $\|e_{k|k}\|$ and the trace of the corresponding state covariance $\operatorname{tr} P_{k|k}$.

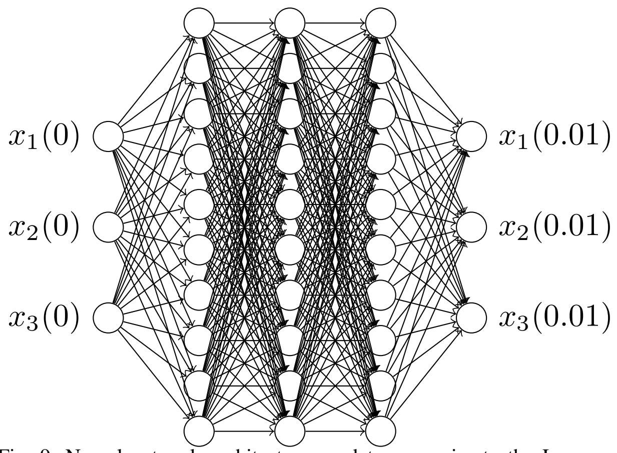

Fig. 9: Neural network architecture used to approximate the Lorenz system.

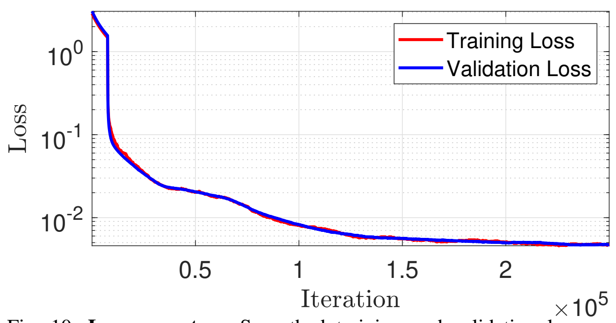

Fig. 10: **Lorenz system.** Smoothed training and validation loss on logarithmic scale during the training process.

D. Double Inverted Pendulum System

As shown in Figure 12, a planar double simple pendulum consists of particles y_1 and y_2 with masses m_1 and m_2, respectively. The particle y_1 is connected to a frictionless pin joint at the point w in the ceiling by means of the massless link $\mathcal{L}_1$ with length ℓ_1, and the particle y_2 is connected by a frictionless pin joint at y_1 to the first link by means of the massless link $\mathcal{L}_2$ with length ℓ_2. The external torque $\tau_{\rm ext}$ is applied to link $\mathcal{L}_1$.

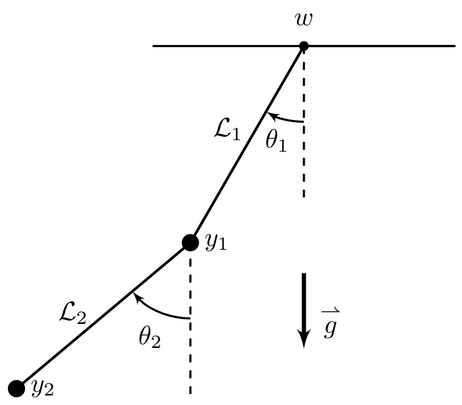

Fig. 12: Planar double simple pendulum.

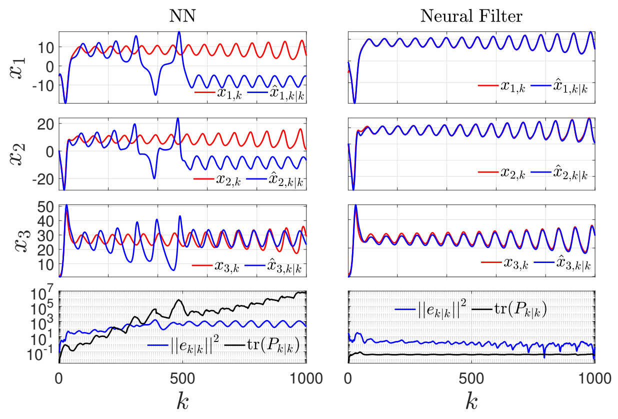

Fig. 11: **Lorenz system.** State predictions using the neural network and the neural filter.

The equations of motion of the double pendulum are

$$(m_2+m_1)\ell_1^2\ddot{\theta}_1 + m_2\ell_1\ell_2(\cos\phi)\ddot{\theta}_2 = m_2\ell_1\ell_2(\sin\phi)\dot{\theta}_2^2 - (m_1+m_2)g\ell_1\sin\theta_1 + \tau_{\rm ext}, \tag{25}$$

$$\ell_1(\cos\phi)\ddot{\theta}_1 + \ell_2\ddot{\theta}_2 = -\ell_1(\sin\phi)\dot{\theta}_1^2 - g\sin\theta_2, \tag{26}$$

where $\phi \triangleq \theta_2 - \theta_1$.

The equations of motion can be written compactly as

$$\mathcal{M}(\theta)\ddot{\theta} + \mathcal{D}(\theta,\dot{\theta}) = T, \tag{27}$$

where $\theta \triangleq \begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix}$ and

$$\mathcal{M}(\theta) \triangleq \begin{bmatrix}(m_2+m_1)\ell_1^2 & m_2\ell_1\ell_2(\cos\phi)\\ \ell_1(\cos\phi) & \ell_2\end{bmatrix}, \tag{28}$$

$$\mathcal{D}(\theta,\dot{\theta}) \triangleq \begin{bmatrix}-m_2\ell_1\ell_2(\sin\phi)\dot{\theta}_2^2 + (m_1+m_2)g\ell_1\sin\theta_1\\ \ell_1(\sin\phi)\dot{\theta}_1^2 + g\sin\theta_2\end{bmatrix}, \tag{29}$$

$$T \triangleq \begin{bmatrix}\tau_{\rm ext}\\ 0\end{bmatrix}. \tag{30}$$

The measurement model for this example is defined as follows

$$y_k = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix} x_k + v_k, \tag{31}$$

where $v_k \sim \mathcal{N}(0, \sigma_v^2 I_4)$ represents zero-mean Gaussian noise. For this example, we specify $\sigma_v = 0.01$.

The training data consists of 200,000 samples of $x(0) \in \begin{bmatrix}\mathcal{U}[-\pi/2,\pi/2]\\ \mathcal{U}[-0.5,0.5]\\ \mathcal{U}[-\pi/2,\pi/2]\\ \mathcal{U}[-0.5,0.5]\end{bmatrix}$. Using MATLAB's ode45 routine, $x(0.01)$ is computed according to (5) assuming $\tau_{\rm ext} = 0$. In this work, we use 80 % of the data to train the model, and the remaining 20 % are used for validation during the training process.

The neural network architecture used to approximate the double pendulum system is shown in Figure 13. In particular, the neural network consists of an input layer with a dimension of 4, four hidden layers containing 10 neurons each, and an output layer with a dimension of 4. All the hidden layers use the rectified linear unit (ReLU) activation function, while the output layer uses a linear activation function. The Adam optimizer is used for training the neural network. Training is performed with a batch size of 32, and validation is conducted every 30 iterations. Figure 14 shows the smoothed training and validation loss on a logarithmic scale during the training process.

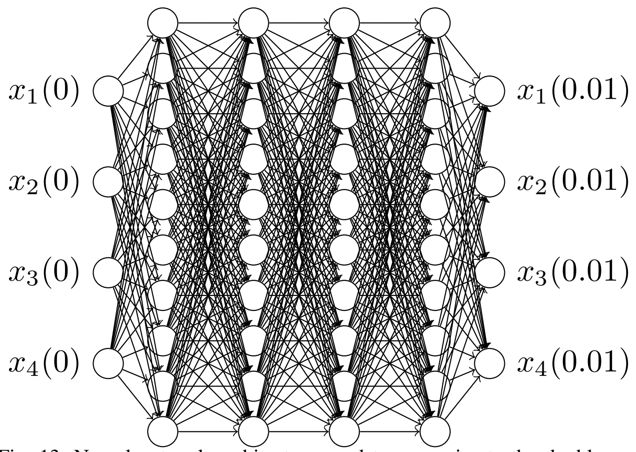

Fig. 13: Neural network architecture used to approximate the double pendulum system.

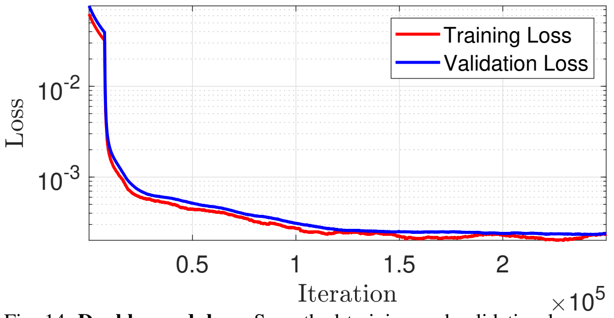

Fig. 14: **Double pendulum.** Smoothed training and validation loss on logarithmic scale during the training process.

Next, the trained neural network and the neural filter are used to predict the state of the double pendulum system. In particular, we set $x(0) = \begin{bmatrix} -0.235 & 0.267 & -0.435 & -0.301 \end{bmatrix}^{\mathrm{T}}$. In the neural filter, we set $P(0) = 10^{-4} \times I_4$, where I_4 is the 4 by 4 identity matrix. Figure 15 shows the predicted states using the neural network (subfigures in the left column) and the neural filter (subfigures in the right column). Note that the state predictions using the neural network degrade over time, as shown by the increasing state error norm $\|e_{k|k}\|$ and the trace of the corresponding state covariance $\operatorname{tr} P_{k|k}$. On the other hand, the state predictions using the neural filter remain accurate over time, as shown by the bounded state error norm $\|e_{k|k}\|$ and the trace of the corresponding state covariance $\operatorname{tr} P_{k|k}$.

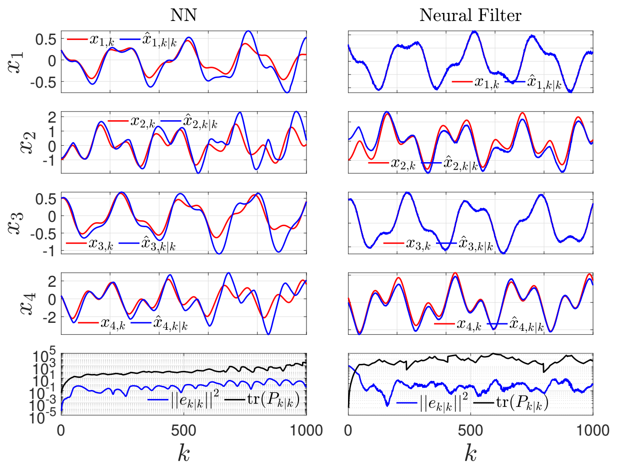

Fig. 15: **Double pendulum.** State predictions using the neural network and the neural filter.

IV. Conclusions

This paper introduced a novel neural filter to improve the accuracy of state predictions using neural network-based models of dynamical systems. Motivated by the extended Kalman filter, the neural filter is constructed using the neural network-based approximation of the dynamical system and augmenting the state prediction by a correction step. The performance of the neural filter is investigated using four nonlinear dynamical systems. In particular, a simple pendulum, a Van der Pol oscillator, Lorenz system, and a double pendulum system are considered to demonstrate the application of the neural filter. In each case, the states predicted by the neural filter remain close to the true states, whereas the state estimates provided by the neural network model alone diverge. Moreover, unlike the neural network model, which requires precise initialization, the neural filter state estimates latch onto true states even with zero initialization. Furthermore, the covariance of the state estimate given by the neural filter remains bounded, whereas the covariance of the state estimate given by the neural network model diverges.

V. Appendix

A. Jacobian of the Neural Network

Following the notation presented in [17], the output of the neural network is given by

$$\begin{aligned} y &= NN(x, \Theta_1, \ldots, \Theta_n, \Theta_{n+1}) \\ &= \Theta_{n+1}^{\mathrm{T}} N_n(N_{n-1}(\ldots(N_1(x, \Theta_1)), \Theta_{n-1}), \Theta_n), \end{aligned} \quad (32)$$

where $N_1, N_2, \ldots, N_n$ are the n neural layers of the neural network and $\{\Theta_i\}_{i=1}^n$ are the neural gains in each layer.

To compute the Jacobian of the neural network function NN with respect to the input x, the output of a neural network function is written using a recursive formula as shown below. By denoting the input and the output of the ith neural layer by x_i and x_{i+1}, it follows that, for $i \in \{1, 2, \ldots, n\}$,

$$x_{i+1} = N_i(x_i, \Theta_i), \tag{33}$$

where $\Theta_i \in \mathbb{R}^{l_{\theta_i} \times \ell_i}$. Note that $x_1 \overset{\triangle}{=} x$. The output of the network is finally given by

$$y = S(x_{n+1}) = \Theta_{n+1}^{\mathrm{T}} x_{n+1}, \tag{34}$$

where $\Theta_{n+1} \in \mathbb{R}^{l_{x_{n+1}} \times l_y}$. Using the chain rule, it follows that

$$\begin{aligned} \frac{\partial NN}{\partial x} = \frac{\partial y}{\partial x} &= \frac{\partial y}{\partial x_{i+1}} \frac{\partial x_{i+1}}{\partial x_i} \frac{\partial x_i}{\partial x_{i-1}} \cdots \frac{\partial x_2}{\partial x_1} \\ &= \Theta_{n+1}^{\mathrm{T}} \frac{\partial N_i}{\partial x_i} \cdots \frac{\partial N_1}{\partial x_1}. \end{aligned} \tag{35}$$

More details about computing the gradients of arbitrarily connected neural networks can be found in [18].

References

[1] L. Song, J. Fan, D.-R. Chen, and D.-X. Zhou, "Approximation of nonlinear functionals using deep relu networks," *Journal of Fourier Analysis and Applications*, vol. 29, no. 4, p. 50, 2023.

[2] I. Higgins, "Generalizing universal function approximators," *Nature Machine Intelligence*, vol. 3, no. 3, pp. 192–193, 2021.

[3] S. Masri, A. Chassiakos, and T. Caughey, "Identification of nonlinear dynamic systems using neural networks," 1993.

[4] K. S. Narendra and K. Parthasarathy, "Neural networks and dynamical systems," *International Journal of Approximate Reasoning*, vol. 6, no. 2, pp. 109–131, 1992.

[5] Y. Y. Chee, P. Oveissi, J. Paredes, D. S. Bernstein, and A. Goel, "Performance comparison of adaptive autopilot architectures for multicopter stabilization and trajectory tracking," in *AIAA SCITECH 2024 Forum*, 2024, p. 1391.

[6] K. Lin, X. Li, Y. Ye, *et al.*, "Spherical neural operator network for global weather prediction," *IEEE Transactions on Circuits and Systems for Video Technology*, 2023.

[7] L. N. Yasnitsky, A. A. Dumler, F. M. Cherepanov, V. L. Yasnitsky, and N. A. Uteva, "Capabilities of neural network technologies for extracting new medical knowledge and enhancing precise decision making for patients," *Expert Review of Precision Medicine and Drug Development*, vol. 7, no. 1, pp. 70–78, 2022.

[8] L. Lu, P. Jin, and G. E. Karniadakis, "Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators," *arXiv preprint arXiv:1910.03193*, 2019.

[9] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational physics*, vol. 378, pp. 686–707, 2019.

[10] S. Pan and K. Duraisamy, "Long-time predictive modeling of nonlinear dynamical systems using neural networks," *Complexity*, vol. 2018, no. 1, p. 4 801 012, 2018.

[11] K. Michałowska, S. Goswami, G. E. Karniadakis, and S. Riemer-Sørensen, "Neural operator learning for long-time integration in dynamical systems with recurrent neural networks," *arXiv preprint arXiv:2303.02243*, 2023.

[12] N. Mohajerin and S. L. Waslander, "Multistep prediction of dynamic systems with recurrent neural networks," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 11, pp. 3370–3383, 2019.

[13] C. K. Chui, G. Chen, *et al.*, *Kalman filtering*. Springer, 2017.

[14] M. I. Ribeiro, "Kalman and extended kalman filters: Concept, derivation and properties," *Institute for Systems and Robotics*, vol. 43, no. 46, pp. 3736–3741, 2004.

[15] M. Mirtaba, M. Jeddi, A. Nikoofard, and Z. Shirmohammadi, "Design and implementation of a low-complexity flight controller for a quadrotor uav," *International Journal of Dynamics and Control*, vol. 11, no. 2, pp. 689–700, 2023.

[16] W. Ditto and T. Munakata, "Principles and applications of chaotic systems," *Communications of the ACM*, vol. 38, no. 11, pp. 96–102, 1995.

[17] T. Rozario, A. Trivedi, and A. Goel, "A tutorial on neural networks and gradient-free training," *arXiv preprint arXiv:2211.17217*, 2022.

[18] B. M. Wilamowski, N. J. Cotton, O. Kaynak, and G. Dundar, "Computing gradient vector and jacobian matrix in arbitrarily connected neural networks," *IEEE Transactions on Industrial Electronics*, vol. 55, no. 10, pp. 3784–3790, 2008.