Scientia Iranica, Vol. 12, No. 2, pp 141–150
© Sharif University of Technology, April 2005

SCIENTIA
IRANICA

Adaptive Nonlinear Observer Design Using Feedforward Neural Networks

M.R. Dehghan Nayeri[1] and A. Alasty*

This paper concerns the design of a neural state observer for nonlinear dynamic systems with noisy measurement channels and in the presence of small model errors. The proposed observer consists of three feedforward neural parts, two of which are MLP universal approximators, which are being trained off-line and the last one being a Linearly Parameterized Neural Network (LPNN), which is being updated on-line. The off-line trained parts are able to generate state estimations instantly and almost accurately, if there are not catastrophic errors in the mathematical model used. The contribution of the on-line adapting part is to compensate the remainder estimation error due to uncertain parameters and/or unmodeled dynamics. A time delay term is also added to compensate the arising differential effects in the observer. The proposed observer can learn the noise cancellation property by using noise corrupted data sets in the MLP's off-line training. Simulation results in two case studies show the high effectiveness of the proposed state observing method.

INTRODUCTION

The state observation problem is one of the most essential problems in modern control theory. In linear systems, the solution is well known and can be expressed by the Kalman filter (for stochastic noise) and Luenberger's observer (for noises of a deterministic nature) [1]. Since early 1980, many published papers have been devoted to the theory and practice of nonlinear observers [2-4]. Based on linearization techniques, the extended Luenberger observers were proposed for nonlinear systems [5]. Nonlinear observer analysis and synthesis, using the Lie-algebra approach and Lyapunov based methods, can be found in [2,6]. The sliding mode observers for linear systems were considered and studied in [7].

On the other hand, the growing need of industry for tackling complex systems and the capability of Neural Networks (NNs) for approximating functions and dynamical systems [8,9], have motivated NN-based identification and control approaches [10]. The main reason for this is the fact that NN-based approaches allow the modeling and control of highly uncertain dynamical systems with unknown nonlinearities, unmodeled dynamics and disturbances.

In [11], Zhu et al. focused on the application of Dynamic Recurrent Neural Networks (DRNN), as observers for nonlinear systems. They considered a class of Single-Input-Single-Output (SISO) nonlinear time-varying systems, where they proved the boundedness of the observer error and the DRNN weights during adaptation using the Lyapunov stability theory and the well-known universal approximation theorem for neural networks [9,11]. With an alternative approach, Wang and Wu [12] exploited the multilayer recurrent neural networks as matrix equation solvers and utilized this scheme to synthesize linear state observers in real time by solving Sylvester's equation for pole placement. There are, also, examples of static feedforward neural network applications in observer and controller designs. For example, Ahmed and Riyaz [13] considered an off-line training scheme for a Multilayer Perceptron (MLP) based observer design for nonlinear systems. They noted that although the NN observer requires more computation in the training phase, it is more computation-efficient compared to the Extended Kalman Filter (EKF) in the implementation phase.

An interesting approach was presented in [14] by Vargas and Hemerly, where they employed Linearly Parameterized Neural Networks (LPNN) for the design

1. *Department of Aerospace Engineering, Sharif University of Technology, Tehran, I.R. Iran.*
*. *Corresponding Author, Center of Excellence in Design, Robotics and Automation (CEDRA), Department of Mechanical Engineering, Sharif University of Technology, Tehran, P.O. Box 11365-9567, I.R. Iran.*

of an adaptive observer for general nonlinear systems. LPNN include a wide class of networks, including Radial-Basis-Function (RBF) networks, adaptive fuzzy systems and wavelet networks. They used the Lyapunov stability theory to prove the stability of the observer and the NN weights. In [15], the Luenberger observer was suggested for extension in two ways; first, the unknown nonlinear dynamics were estimated by a dynamic NN; second, the time delay term was added to compensate the arising differential effects in the Luenberger observer.

Besides all the theoretical approaches to the subject matter, there are also a number of application-oriented studies on the use of neural networks. Most commonly investigated applications are observer and controller designs for robot manipulators [16], induction motors [17], synchronous generators [18] and the air-fuel ratio in gasoline engines [19]. Most of the referred studies followed a conservative approach to the NN based observer design, wherein they mostly extended a classical approach, such as EKF or Luenberger, by using the NNs. In this study, a design approach, which uses only NNs, is suggested for observing the states of nonlinear dynamic systems with noisy measurement channels.

The structure of the proposed observer consists of three parts, which are being trained separately. The first two parts are off-line trained MLP networks and the last part is a Linearly Parameterized Neural Network (LPNN) that is being trained on-line.

Two different off-line schemes are proposed for training of the first two MLP networks. The first one is trained, based on an error Backpropagation (BP) algorithm and, then, the other is trained, based on a Backpropagation Through Time (BTT) algorithm [20,21]. Finally, a recursive steepest descent on-line algorithm is used to train the third part (the LPNN). It is pointed out that the off-line trained parts (without an on-line training part) would be able to generate state estimates instantly and almost accurately, if there were not catastrophic errors in the mathematical model used. The contribution of the on-line part is to compensate the remainder of the estimation error, due to uncertain parameters and/or unmodeled dynamics, from instantaneous output error feedback (measured output minus estimated output).

DEVELOPMENT OF THE NEURAL OBSERVER DESIGN

Consider the class of nonlinear dynamic systems given by:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t), t), \tag{1}$$

$$\mathbf{y}(t) = C\mathbf{x}(t) + \boldsymbol{\xi}_y(t), \tag{2}$$

where $\mathbf{x}(t) \in \Re^n$ is the state vector of the system (the initial state, $\mathbf{x}_0$, is unknown); $\mathbf{u}(t) \in \Re^q$ is a given control action; $\mathbf{y}(t) \in \Re^m$ is the output vector ($u(t)$ and $y(t)$ are assumed to be measurable at each time t); $\mathbf{C} \in \Re^{m\times n}$ is a known output matrix; $\mathbf{f} : \Re^{\mathbf{n+q+1}} \to \Re^{\mathbf{n}}$ is a nonlinear function describing the system dynamics and $\boldsymbol{\xi}_{\mathbf{y}}(\mathbf{t}) \in \Re^{\mathbf{m}}$ is an unknown random vector representing additive measurement noises.

One can utilize the following scheme for an observer mathematical model:

$$\dot{\hat{\mathbf{x}}}(t) = F(\hat{\mathbf{x}}(t), \mathbf{u}(t), t) + H([\mathbf{y}(t) - \hat{\mathbf{y}}(t)]), \tag{3}$$

where $\hat{\mathbf{x}}(t)$ is the observed state vector at time $t, \hat{y}(t) = C\hat{x}(t), \mathbf{F}$ can be regarded as an approximation of the described state space mapping and H is an unknown nonlinear function that can be approximated. The second term on the right-hand side of Equation 3 intends to correct the estimated trajectory, based on the current residual values $[\mathbf{y}(t) - \hat{\mathbf{y}}(t)]$.

The application of such observers to a class of mechanical systems when only the position measurements are available, turns out to be not so good. To describe this, consider an original dynamic mechanical system that is given as a second-order ODE:

$$\ddot{\mathbf{Z}}(t) = G\left(\mathbf{Z}(t), \dot{\mathbf{Z}}(t), \mathbf{u}(t), t\right), \tag{4}$$

$$\mathbf{y}(t) = \mathbf{Z}(t), \tag{5}$$

or, in an equivalent standard Cauchy form of:

$$\dot{\mathbf{x}}_1(t) = \mathbf{x}_2(t),$$

$$\dot{\mathbf{x}}_2(t) = \tilde{G}(\mathbf{x}(t), \mathbf{u}(t), t), \tag{6}$$

$$\mathbf{y}(t) = \mathbf{x}_1(t). \tag{7}$$

So, the corresponding nonlinear observer (Equation 3) has the form:

$$\begin{pmatrix} \dot{\hat{\mathbf{x}}}_1(t) \\ \dot{\hat{\mathbf{x}}}_2(t) \end{pmatrix} = \begin{pmatrix} \hat{\mathbf{x}}_2(t) \\ \overset{\sim}{\mathbf{G}}(\hat{\mathbf{x}}(t), \mathbf{u}(t), t) \end{pmatrix} + \begin{pmatrix} H_1([\mathbf{y}(t) - \hat{\mathbf{x}}_1(t)]) \\ H_2([\mathbf{y}(t) - \hat{\mathbf{x}}_1(t)]) \end{pmatrix}. \tag{8}$$

In such systems, if only the position estimation error $[\mathbf{y}(t) - \hat{\mathbf{x}}_1(t)]$ is fedback, any current information, containing the output $(y = x_1(t))$, has no influence on the velocity estimates, $\hat{x}_2(t)$, that leads to their bad estimates. To improve the rate estimation error $[\mathbf{x}_2(t) - \hat{\mathbf{x}}_2(t)]$, it has been suggested in [15] to add a time delay term to the observer model (Equation 3) as follows:

$$\begin{aligned} \dot{\hat{\mathbf{x}}}(t) =& F(\hat{\mathbf{x}}(t), \mathbf{u}(t), t) \\ &+ H([\mathbf{y}(t) - \hat{\mathbf{y}}(t)], [\mathbf{y}(t-h) - \hat{\mathbf{y}}(t-h)]), \end{aligned} \tag{9}$$

where h is a positive constant corresponding to a selected time delay. If h is small enough, the following interpretation can be utilized:

$$\dot{\mathbf{y}}(t) \approx h^{-1}\left(\hat{\mathbf{y}}(t) - \hat{\mathbf{y}}(t-h)\right). \tag{10}$$

In this section, it is assumed that the mathematical model described by Equation 1 is known to be accurate enough and almost certain. The arising problems, due to uncertain parameters and unmodeled dynamics, are addressed in the following section. Considering the above mentioned assumption, the proposed neural observer consists of two off-line trained networks with the following discrete form:

$$\begin{aligned}\hat{\mathbf{x}}_{k+1} =&\mathrm{NI}\left(\hat{\mathbf{x}}_k, \mathbf{u}_k, \mathbf{W}_{\mathrm{NI}}\right)\\ &+ \mathrm{NFT}\left([\mathbf{y}_k - \hat{\mathbf{y}}_k], [\mathbf{y}_{k-1} - \hat{\mathbf{y}}_{k-1}], \mathbf{W}_{\mathrm{NFT}}\right),\end{aligned} \tag{11}$$

where NI (Neural-Identifier) and NFT (Neural-Feedback-Tuning) are separately trained MLP networks, which are used to approximate the nonlinear mapping functions, F and H (in Equation 9), respectively and $\mathbf{W}_{\mathrm{NI}}$ and $\mathbf{W}_{\mathrm{NFT}}$ are their corresponding weight and bias parameters. The architecture for the proposed observer is shown in Figure 1.

The training procedures of the NI and NFT, which are done off-line, are quite different. First, the NI is trained by the well-known backpropagation (BP) algorithm and, then, using the trained NI, the NFT is trained, based on the Backpropagation Through Time (BTT) algorithm. Detailed descriptions of training procedures are addressed in the next section.

OFF-LINE TRAINING FOR THE NEURAL OBSERVER

Both the NI and NFT are two-layer tansig/pureline MLP networks, as shown in Figure 2. Such MLP networks are universal approximators, because they can learn any nonlinear complex mapping using generated input/target sets, given sufficient neurons in the hidden layer [9]. Training procedures of the mentioned networks are, as follows, in the next subsections.

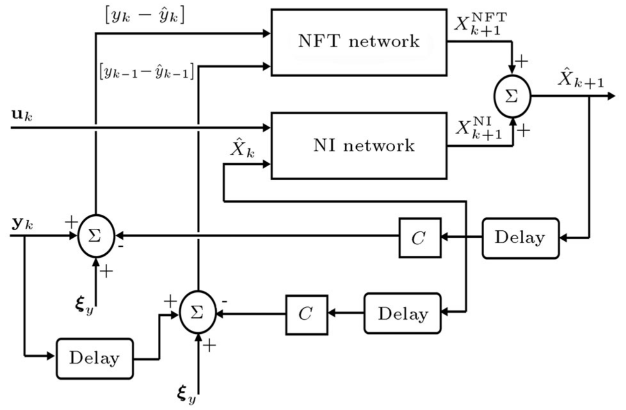

Figure 1. Neural observer architecture.

BP Training for the NI

The function of the NI is to identify the plant model (Equation 1) and to be used as a part of the observation process. It is also used to backpropagate the equivalent error to the NFT by calculating the plant Jacobians.

By using the state space model of the system, the problem of NI training can be regarded as an approximation process of state space nonlinear mapping, so that the NI can approximate the discrete-time model of the system as:

$$\mathbf{x}^{\mathrm{NI}}_{k+1} = \mathrm{NI}\left(\mathbf{x}^M_k, \mathbf{u}_k, \mathbf{W}_{\mathrm{NI}}\right), \tag{12}$$

where $\mathbf{x}^M_k$ is the state vector generated by the differential equations of the system (Equation 1) and $\mathbf{x}^{\mathrm{NI}}_{k+1}$ is the NI output vector.

As previously mentioned (in Figure 1), $\mathbf{x}^{\mathrm{NI}}_{k+1}$ is the contribution of the NI network in the observed state vector ($\hat{\mathbf{x}}_{k+1}$). Here, the training of the NI is to adjust its weight parameters so that it emulates the differential Equation 1. The objective of training is to reduce average error defined by:

$$J = \frac{1}{2N_d}\sum_{i=1}^{N_d}\sum_{k=0}^{N_{h-1}}\left(\mathbf{x}^M_{k+1}(i) - \mathbf{x}^{\mathrm{NI}}_{k+1}(i)\right)^T\left(\mathbf{x}^M_{k+1}(i) - \mathbf{x}^{\mathrm{NI}}_{k+1}(i)\right), \tag{13}$$

where N_d is the number of training sets, i represents the data set, which is the ith training sample and N_h is the number of time horizons. Input-state training patterns are obtained from the operation history of the plant under various conditions.

One of the most important problems in observer operation is the measurement noise. Even if a typical observer can work in the presence of measurement noise, these noises are transferred to the estimated states and, sometimes, the performance of the observer deteriorates significantly. One of the solutions to this problem, which is proposed in this paper, is to use neural network learning capabilities, in the same way that one can learn the neural observer, so that it cancels incoming noises from the outgoing states. For this reason, in the off-line training phase, the incoming states to the NI, i.e. $\mathbf{x}^M_k$, are corrupted with a Gaussian white noise, while the noise-free states are used for error backpropagating. Using the proposed method, the neural network learns to cancel the noises, which are coming from the measurement channels. The same procedure would be employed in NFT training, as will be discussed. The block diagram for NI training is given in Figure 3.

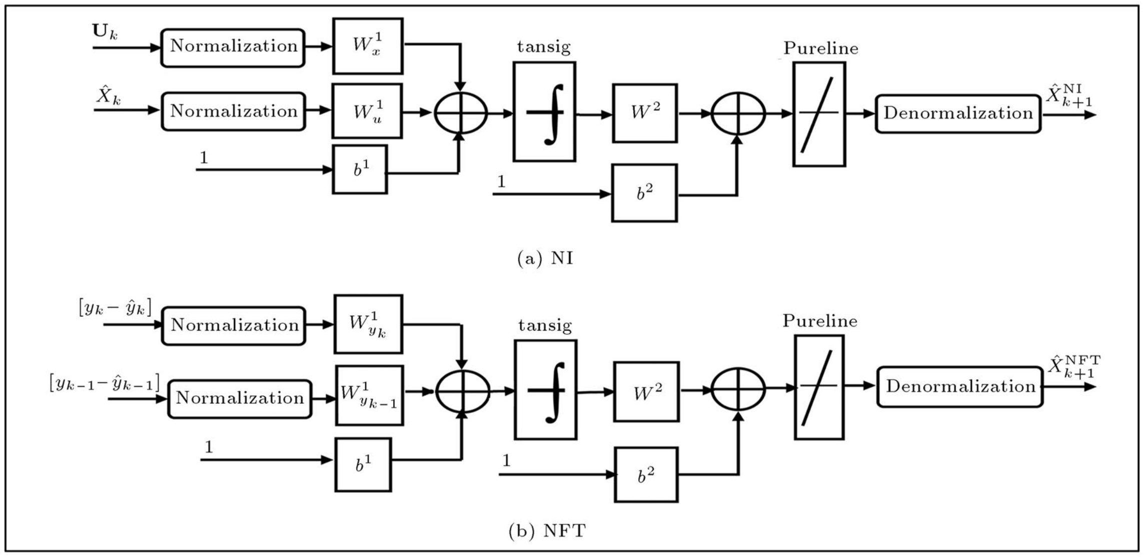

Figure 2. MLP network structure for (a) NI and (b) NFT.

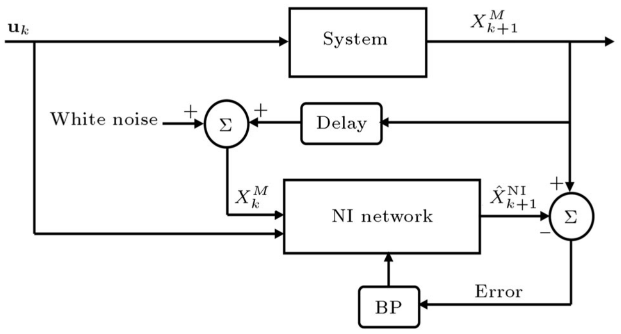

Figure 3. Block diagram for training the NI network.

Using the backpropagation algorithm, the weight parameters of the NI are updated in the following manner:

$$\Delta \mathbf{W}^k = \gamma \Delta \mathbf{W}^{k-1} + \alpha(\gamma - 1)\frac{\partial J}{\partial \mathbf{W}^k}, \tag{14}$$

where α and γ are learning rate and momentum coefficient, respectively and γ should satisfy the following condition:

$$0 \leq \gamma \leq 1. \tag{15}$$

Through the learning process, α and γ should be identified adaptively and normalization of training sets should be done. For more details of the BP method, readers are referred to [22].

The training would be terminated when the average error between the plant states and NI outputs (J in Equation 13) converges to a small value.

The problem arising here is that, in a real time process, $\mathbf{x}_k^M$ does not exist to be fedback to the NI network, therefore, after the training is done, the following relation is used as an approximation to Equation 12 in real time:

$$\hat{\mathbf{x}}_{k+1} = \mathrm{NI}(\hat{\mathbf{x}}_k, \mathbf{u}_k, \mathbf{W}_{\mathrm{NI}}). \tag{16}$$

BTT Training for the NFT

As the observer structure (Figure 1 and Equation 11) indicates, the contribution of the NFT (Neural-Feedback-Tuning) is to close the estimation loop, because the NI network is an open loop estimator and is not stable by itself. Here, the NFT makes the neural observer be a stable closed-loop estimator. As previously mentioned (Figure 2), the NFT network is a two-layer MLP, which has a universal approximation property. For NFT off-line training, a method, based on the Backpropagation-Through-Time (BTT) algorithm, is developed. The objective of this method is to minimize the following receding horizon cost function:

$$J = \frac{1}{2}\sum_{k=1}^{N+1} \left[\mathbf{x}_k^M - \hat{\mathbf{x}}_k\right]^T Q \left[\mathbf{x}_k^M - \hat{\mathbf{x}}_k\right], \tag{17}$$

where N is an appropriate time horizon and Q is a positive definite weighting matrix. The BTT algorithm can be regarded as a trial and error learning procedure, which consists of two main parts. First, from randomly selected initial states and an arbitrary given control effort (which can be randomly selected or generated by a designed control system), the plant model and observer are derived for N steps. Second, the weight and bias parameters in the NFT are updated, using the equivalent error generated. To perform these steps, the state sensitivity of the cost function is defined by:

$$\boldsymbol{\delta}_x^k \triangleq -\frac{\partial \mathbf{J}}{\partial \hat{\mathbf{x}}_k}, \quad k = 1, 2, \cdots, N+1. \tag{18}$$

Using the chain rule, the above gradient can be developed as:

$$\begin{aligned}\boldsymbol{\delta}_x^k =& Q\left(\mathbf{x}_k^M - \hat{\mathbf{x}}_k\right) + \left(\frac{\partial \hat{\mathbf{x}}_{k+1}}{\partial \hat{\mathbf{x}}_k}\right)^T \boldsymbol{\delta}_x^{k+1} \\ &+ \left(\frac{\partial \hat{\mathbf{x}}_{k+2}}{\partial \hat{\mathbf{x}}_k}\right)^T \boldsymbol{\delta}_x^{k+2} \quad k = 1, 2, \cdots, N-1, \quad (19)\end{aligned}$$

$$\boldsymbol{\delta}_x^N = Q\left(\mathbf{x}_N^M - \hat{\mathbf{x}}_N\right) + \left(\frac{\partial \hat{\mathbf{x}}_{N+1}}{\partial \hat{\mathbf{x}}_N}\right)^T \boldsymbol{\delta}_x^{N+1}, \quad (20)$$

$$\boldsymbol{\delta}_x^{N+1} = Q\left(\mathbf{x}_{N+1}^M - \hat{\mathbf{x}}_{N+1}\right). \quad (21)$$

Using the weight parameters of the NI and NFT, the existing Jacobians can be expressed as:

$$\begin{aligned}\frac{\partial \hat{\mathbf{x}}_{k+1}}{\partial \hat{\mathbf{x}}_k} =& \left[\frac{\partial \mathrm{NI}\left(\hat{\mathbf{x}}_k, \mathbf{u}_k, \mathbf{W}_{\mathrm{NI}}\right)}{\partial \hat{\mathbf{x}}_k}\right] \\ &- \left[\frac{\partial \mathrm{NFT}\left(\hat{\mathbf{e}}_{y_k}, \hat{\mathbf{e}}_{y_{k-1}}, \mathbf{W}_{\mathrm{NFT}}\right)}{\partial \hat{\mathbf{e}}_{y_k}}\right] C, \quad (22)\end{aligned}$$

$$\frac{\partial \hat{\mathbf{x}}_{k+1}}{\partial \hat{\mathbf{x}}_{k-1}} = -\left[\frac{\partial \mathrm{NFT}\left(\hat{\mathbf{e}}_{y_k}, \hat{\mathbf{e}}_{y_{k-1}}, \mathbf{W}_{\mathrm{NFT}}\right)}{\partial \hat{\mathbf{e}}_{y_{k-1}}}\right] C, \quad (23)$$

$$\hat{\mathbf{e}}_{y_k} = \mathbf{y}_k - C\hat{\mathbf{x}}_k. \quad (24)$$

The formula for derivation of MLP Jacobians $\frac{\partial \mathrm{NI}}{\partial \hat{\mathbf{x}}_k}$, $\frac{\partial \mathrm{NFT}}{\partial \hat{\mathbf{e}}_{y_k}}$ and $\frac{\partial \mathrm{NFT}}{\partial \hat{\mathbf{e}}_{y_{k-1}}}$ is described in the Appendix, Equations A1 and A2.

Now, the off-line BTT training procedure of the NFT can be summarized as follows:

i) Generate small random weights and biases for the NFT network;

ii) Set the plant initial states with random numbers in the operation region of the plant and observer initial states with arbitrary numbers (for example, zero);

iii) Forward pass: Run the plant model, neural observer and controller for N steps forward from $k = 1$ to $N + 1$. If there is not a designed control algorithm, set the control inputs as a sinusoidal function with random amplitude and frequency in the operation region of the plant. The forward pass generates sequences of plant states, $\mathbf{x}_1^M, \mathbf{x}_2^M, \cdots, \mathbf{x}_{N+1}^M$ and observed states, $\hat{x}_1, \hat{x}_2, \cdots, \hat{x}_{N+1}$;

iv) Backward pass: Using the operation results in Step (iii), run the state sensitivity equations backward from $k = N + 1$ to 1, to evaluate the equivalent error, $\boldsymbol{\delta}_x^k, k = N+1, N, \cdots, 1$;

v) Update the weights and biases of the NFT network using:

$$\begin{aligned}\Delta\mathbf{W}_{\mathrm{NFT}}(j+1) =& \gamma_j \Delta\mathbf{W}_{\mathrm{NFT}}(j) \\ &+ \eta_j\left(\gamma_j - 1\right)\sum_{k=1}^{N}\left(\frac{\partial \hat{\mathbf{x}}_{k+1}}{\partial \mathbf{W}_{\mathrm{NFT}}(j)}\right)^T \boldsymbol{\delta}_X^{k+1}, \quad (25)\end{aligned}$$

where:

$$\begin{aligned}\frac{\partial \hat{\mathbf{x}}_{k+1}}{\partial \mathbf{W}_{\mathrm{NFT}}(j)} =& \frac{\partial \mathrm{NFT}\left(\hat{\mathbf{e}}_{y_k}, \hat{\mathbf{e}}_{y_{k-1}}, \mathbf{W}_{\mathrm{NFT}}(j)\right)}{\partial \mathbf{W}_{\mathrm{NFT}}(j)} \\ &+ \left(\frac{\partial \hat{\mathbf{x}}_{k+1}}{\partial \hat{\mathbf{x}}_k}\right)\left(\frac{\partial \hat{\mathbf{x}}_k}{\partial \mathbf{W}_{\mathrm{NFT}}(j)}\right) \\ &+ \left(\frac{\partial \hat{\mathbf{x}}_{k+1}}{\partial \hat{\mathbf{x}}_{k-1}}\right)\left(\frac{\partial \hat{\mathbf{x}}_{k-1}}{\partial \mathbf{W}_{\mathrm{NFT}}(j)}\right). \quad (26)\end{aligned}$$

γ_j and η_j are momentum coefficient and variable learning rate, respectively, and should be selected adaptively [22]. The terms $\frac{\partial \hat{\mathbf{x}}_{k+1}}{\partial \hat{\mathbf{x}}_k}$ and $\frac{\partial \hat{\mathbf{x}}_{k+1}}{\partial \hat{\mathbf{x}}_{k-1}}$ can be calculated using Equations 22 and 23;

vi) Go to Step iii until convergence.

The training would be terminated when, by updating the NFT weights, no appreciable change in the receding horizon cost function (Equation 17) is observed.

In Equation 26, the formulas for derivation of $\frac{\partial \mathrm{NFT}\left(\hat{\mathbf{e}}_{y_k}, \hat{\mathbf{e}}_{y_{k-1}}, \mathbf{W}_{\mathrm{NFT}}(j)\right)}{\partial \mathbf{W}_{\mathrm{NFT}}(j)}$, which is the gradient of the MLP output with respect to its weights and biases vector, are explained in the Appendix, Equations A3 through A7.

To achieve the generalization property, the training algorithm should be repeated for other sets of initial conditions and set-points iteratively. An important matter is the off-line training of NFT. In the forward pass phase, the output signals are corrupted with a Gaussian white noise; but, in performance measure and state sensitivity equations, noise free output signals are backpropagated. Using this procedure, the NFT learns the noise cancellation property (the same as the NI, as described in the last section).

ADDITIVE ON-LINE ADAPTING PART

In the previous section, a comprehensive neural observer was designed, based on the assumption that the mathematical model of the system is known and there are not catastrophic errors in the model. But, there are many nonlinear systems, in which the mathematical models are not completely known and/or some of their parameters are not certain. Although the neural observers, generally, have robustness properties, in the presence of large model errors, the observer results may be unsatisfactory. One of the possible solutions to this problem is to add an adaptive online term, whose contribution is to compensate the remainder estimation error, due to unmodeled dynamics and/or parameter variations, from instantaneous output error feedback (measured output minus observed output).

In this paper, Linearly Parameterized Neural Networks (LPNNs) are used as the adaptive part. These networks are mathematically very simple and computationally efficient for on-line training. The neural observer structure (Equation 11) can be modified by adding the on-line updating LPNN, as:

$$\begin{aligned}\hat{\mathbf{x}}_{k+1} =& \mathrm{NI}\left(\hat{\mathbf{x}}_k, \mathbf{u}_k, \mathbf{W}_{\mathrm{NI}}\right)\\ &+\mathrm{NFT}\left(\left[\mathbf{y}_k-\hat{\mathbf{y}}_k\right],\left[\mathbf{y}_{k-1}-\hat{\mathbf{y}}_{k-1}\right], \mathbf{W}_{\mathrm{NFT}}\right)\\ &+\left[\mathbf{W}_B\left(\mathbf{y}_k-\hat{\mathbf{y}}_k\right)+\mathbf{b}_B\right]. \end{aligned} \tag{27}$$

During real time implementation, the off-line trained parts (i.e., NI and NFT) are only simulated and are not updated, but, W_B and b_B are updated, at each time step, using the following recursive steepest descent algorithm:

$$\Delta \mathbf{W}_b^k=\gamma \Delta \mathbf{W}_b^{k-1}+\eta(\gamma-1) \frac{\partial J^k}{\partial \mathbf{W}_b^k}, \tag{28}$$

where α and γ are learning rate and momentum coefficient, respectively, and $\mathbf{W}_b$ is a vector made by arranging the elements of $\mathbf{W}_B$ and $\mathbf{b}_B$.

The instantaneous performance measure (J^k) is defined by:

$$J^k=\frac{1}{2}\left(\mathbf{y}_k-\hat{\mathbf{y}}_k\right)^T\left(\mathbf{y}_k-\hat{\mathbf{y}}_k\right). \tag{29}$$

Finally, $\frac{\partial J^k}{\partial \mathbf{W}_b^k}$ can be developed as:

$$\frac{\partial J^k}{\partial \mathbf{W}_b^k}=\left(\hat{\mathbf{y}}_k-\mathbf{y}_k\right)^T C \frac{\partial \hat{\mathbf{x}}_k}{\partial \mathbf{W}_b^k}, \tag{30}$$

where $\frac{\partial \hat{\mathbf{x}}_k}{\partial \mathbf{W}_b^k}$ is calculated through the following recursive relation:

$$\begin{aligned}\frac{\partial \hat{\mathbf{x}}_k}{\partial \mathbf{W}_b^k}=&\Bigg(\Big[[D_1][D_2]\cdots[D_n][I]\Big]_{n\times(n\times m+n)}\\ &+\Bigg(\frac{\partial \mathrm{NI}\left(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1}, \mathbf{W}_{\mathrm{NI}}\right)}{\partial \hat{\mathbf{x}}_{k-1}}\\ &-\cdots\Bigg(\frac{\partial \mathrm{NFT}\left(\hat{\mathbf{e}}_{y_{k-1}}, \hat{\mathbf{e}}_{y_{k-2}}, \mathbf{W}_{\mathrm{NFT}}\right)}{\partial \hat{\mathbf{e}}_{y_{k-1}}}\\ &+\mathbf{W}_B\Bigg) C\Bigg) \frac{\partial \hat{\mathbf{x}}_{k-1}}{\partial \mathbf{W}_b^k}\Bigg)\Bigg). \end{aligned} \tag{31}$$

n and m have been defined in the previous section and:

$$[D_j]=\begin{bmatrix} 0_{1\times m}\\ \vdots\\ \mathbf{e}_{y_k}^T \rightarrow j\text{th row}\\ \vdots\\ 0_{1\times m}\end{bmatrix}_{n\times m}, \tag{32}$$

$$[\mathbf{I}]=\begin{bmatrix}1 & 0 & \cdots & 0\\ 0 & 1 & \cdots & 0\\ \vdots & \vdots & & \vdots\\ 0 & 0 & \cdots & 1\end{bmatrix}_{n\times n}. \tag{33}$$

The training would be terminated when the instantaneous performance measure (Equation 29) becomes less than a desired value or processor computation time becomes more than a time step. The architecture of the overall neural observer is shown in Figure 4.

It is also noted that the stability of the on-line training part is not a challenging matter here, because the NI + NFT can stably observe the states, even though the on-line part is removed in the estimation process. The main role of the on-line part is not to stabilize the process, but to reduce the remainder error due to unmodeled dynamics. Most of the referred papers have used the on-line part as the process stabilizer [14-16], in which the guaranty of stability is crucial. Therefore, if the on-line updating (LPNN) tends to diverge in a time step, the algorithm will remove it in that step and, then, in the next step, on-line updating begins with zero (or random) initial weights and biases.

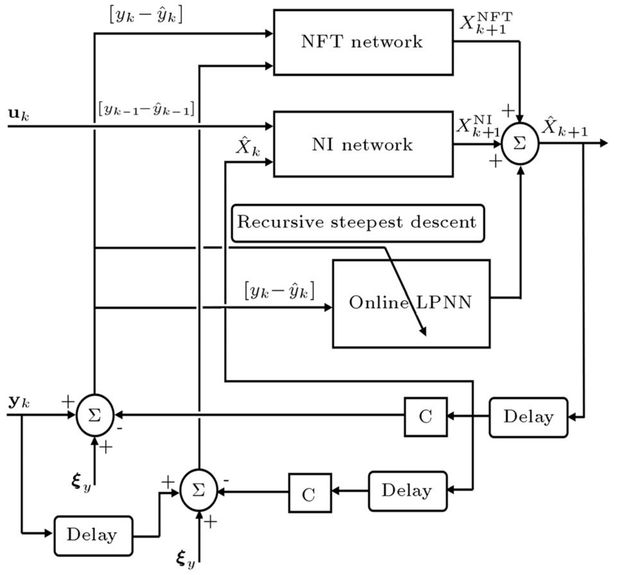

Figure 4. Architecture of the overall neural observer.

SIMULATION RESULTS

In this section, simulation results of two examples, illustrating the applicability of the proposed neural observer, are presented.

Case Study 1

Consider the Van der Pol oscillator [15] with the output subjected to the measurement noises:

$$\dot{x}_1 = x_2,$$

$$\dot{x}_2 = \mu(1 - x_1^2)x_2 - x_1,$$

$$y = x_1 + \xi_y,$$

$$\text{IC}s : x_1(0) = 2, \quad x_2(0) = 1.$$

First, it is assumed that the model is perfect and μ is known and equal to 1.50. Initial conditions for the observer and time step are chosen as [15]:

$$\hat{x}_1(0) = \hat{x}_2(0) = 0,$$

$$h = 0.3[\text{sec}].$$

Measurement noise, (ξ_y), is assumed to be white noise, with a variance equal to 0.1. The number of hidden neurons in the NI and NFT networks is selected to be 20 and 16, respectively.

First, NI and then NFT were trained off-line using BP and BTT, respectively. For off-line training, a hundred ICs were picked up randomly from the operating range of the plant. The training iteration was stopped when no appreciable change in the criterion function was observed. In this case, the observer results for an IC, which has not been used in the trainings, are reported in Figure 5. It is noted that, in this case, the on-line part is not presented. The results show that the proposed scheme yielded satisfactory smooth state estimates.

Second, it is assumed that the μ's value are not known and, also, an unmodeled dynamics $L(t)$ is added, such that:

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = \mu(1 - x_1^2)x_2 - x_1 + L(t).$$

For off-line training, the value of μ is chosen equal to 1.5, but, its real value (for the plant) is taken as 2.5. Initial conditions, measurement noise and time step are the same as the first part. $L(t)$ is selected to be a Gaussian white noise with the variance equal to 0.4. In this case, the behavior of the overall neural observer, in the presence of the on-line adapting part, is shown in Figure 6. By studying the results, it is obvious that the observation is quite satisfactory, even in the presence of a 40% parameter error (μ's error) and almost large, unmodeled dynamics. In comparison to [15], where the observed states are noisy, due to measurement noise, the results of the proposed simulation show that the measurement noise is not transferred to the observed states.

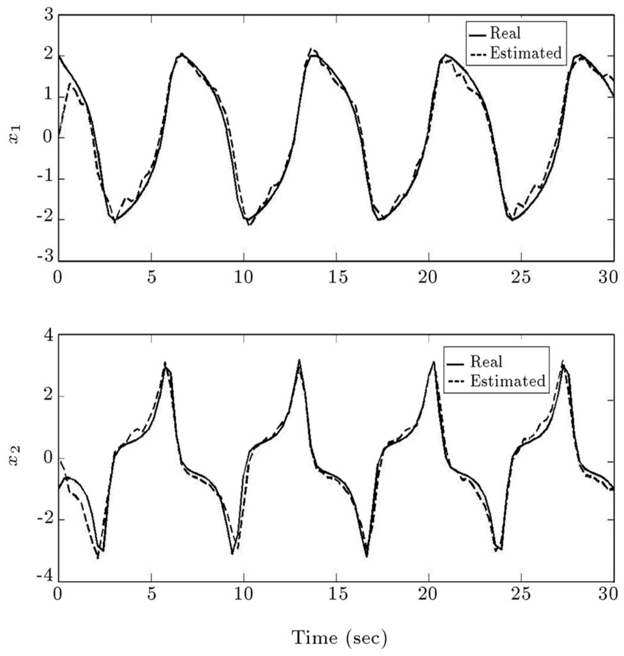

Figure 5. Real and estimated states of the Van der Pol oscillator with noise.

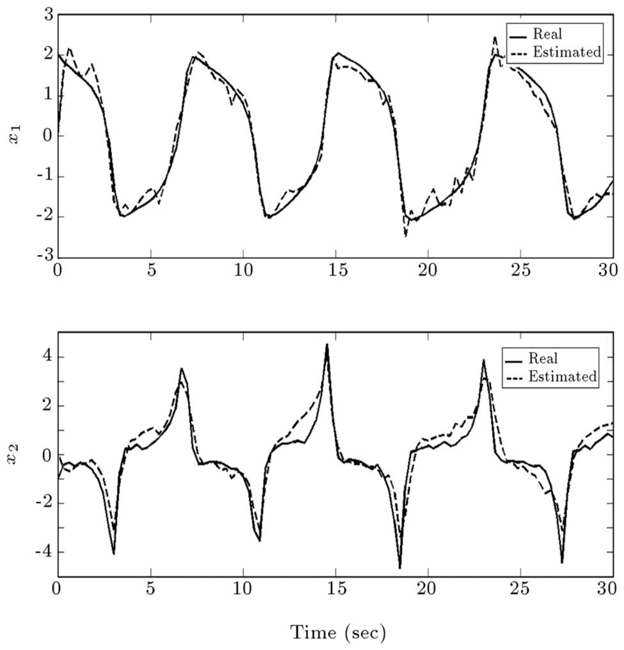

Figure 6. Real and estimated states of the Van der Pol oscillator with modeling error and noise.

Case Study 2

Consider a single-link robot manipulator, rotating in a vertical plane [14], described as:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix} x_2 \\ -\sin(x_1) + u(t)\end{bmatrix} + \Delta\mathbf{f},$$

$$y = x_1 + \xi_y,$$

where the unmodeled dynamics are given as follows:

$$\Delta\mathbf{f} = 0.01 \begin{bmatrix} x_1\cos(x_1) \\ x_2\sin(x_2)\end{bmatrix},$$

and $u(t)$ is selected to be zero. In addition to what has been considered in [14], this paper considers unmodeled dynamics, $\Delta\mathbf{f}$, and measurement noise, ξ_y, which introduces a more general case. After off-line training, the overall observer was tested with plant initial conditions as follows:

$$x_1(0) = 2, \quad x_2(0) = 1.$$

While the observer initial conditions, time step and measurement noise are chosen as:

$$\hat{x}_1(0) = \hat{x}_2(0) = 0,$$

$$h = 0.4[\text{sec}],$$

$$\xi_y \rightarrow \quad \text{white noise, with the variance equal to 0.15,}$$

the number of hidden neurons in the NI and NFT networks is selected as 23 and 18, respectively. The behavior of the overall neural observer is shown in Figure 7. The results are completely satisfactory, even in the presence of measurement noise and unmodeled dynamics. The results verify, again, that the observed states are not noisy, due to measurement noise.

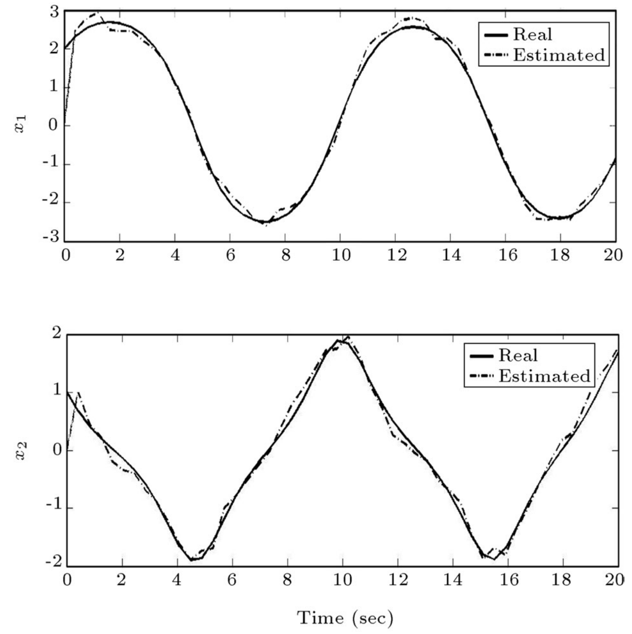

Figure 7. Real and estimated states of the robot manipulator with modeling error and noise.

CONCLUSION

In this paper, a new neural observer is designed and it is shown that it can provide a good enough estimation process for nonlinear dynamic systems in the presence of internal uncertainties and external perturbations. The proposed scheme consists of three neural parts, two of which are off-line trained MLP networks and the other being an on-line updating LPNN. If the mathematical model is perfect, the off-line parts are sufficient for the observation process, but, in the presence of model error, the on-line part adapts and compensates the estimation error, due to model error. By adding a time delay term, the arising differential effects are compensated. MLP's off-line training, using noise corrupted data sets, helps the observer to cancel most of the measurement noises from the observed states. The numerical experiments demonstrate the high effectiveness of the proposed technique.

REFERENCES

1. Luenberger, D.G. "Observing the state of linear systems", *IEEE Transactions on Military Electron*, **8**, pp 74-90 (1964).
2. Gauthier, J.P., Hammouri H. and Othman S. "A simple observer for nonlinear systems: Applications to bioreactors", *IEEE Transactions on Automatic Control*, **37**, pp 875-880 (1992).
3. Marino, R. and Tomei, P. "Adaptive observer with arbitrary exponential rate of convergence for nonlinear systems", *IEEE Transactions on Automatic Control*, **40**, pp 1300-1304 (1995).
4. Ciccarella, G., Dalla Mora, M. and Germani, A. "A Luenberger-like observer for nonlinear systems", *International Journal of Control*, **45**, pp 537-556 (1993).
5. Walcott, B.L., Corless, M.J. and Zak, S.H. "Comparative study of nonlinear state observation technique", *International Journal of Control*, **45**, pp 2109-2132 (1987).
6. Tsinias, J. "Further results on observer design problem", *Systems and Control Letters*, **14**, pp 411-418 (1990).
7. Sira-Ramirez, H. and Spurgeon, S.k. "On the robust design of sliding observers for linear systems", *Systems and Control Letters*, **23**, pp 9-14 (1994).
8. Funahashi, K. and Nakamura, Y. "Approximation of dynamic systems by continuous time recurrent neural networks", *Neural Networks*, **6**, pp 801-806 (1993).

9. Hornik, K., Stinchcombe, M. and White, H. "MLP's are universal approximators", *Neural Networks*, **2**, pp 359-366 (1989).

10. Narendra, K.S. and Parthasarathy, K. "Identification and control of dynamic systems using neural networks", *IEEE Transactions on Neural Networks*, **1**, pp 4-27 (1990).

11. Zhu, R., Chai, T. and Shao, C. "Robust nonlinear adaptive observer design using dynamic recurrent neural networks", *Proc. Amer. Cont. Conf.*, pp 1096-1100, New Mexico (June 1997).

12. Wang, J. and Wu, G. "Real time synthesis of linear state observers using a multilayer recurrent neural networks", *Proc. IEEE Int. Conf. Industrial Tech.*, pp 287-282 (1994).

13. Ahmed, M.S. and Riyaz, S.H. "Design of dynamic neural observers", *IEEE Proc. Cont. Theory Appl.*, **147**(3), pp 257-266 (May 2000).

14. Vargas, J.R. and Hemerly, E.M. "Adaptive observers for unknown general nonlinear systems", *IEEE Transactions on Systems, Man. and Cybernetics*, **31**(5), pp 683-690 (2001).

15. Poznyak, A. and Yu, W. "Robust asymptotic neuro-observer with time delay term", *International Journal of Robust and Nonlinear Control*, **10**, pp 535-559 (2000).

16. Sun, F., Sun, Z. and Woo, P. "Neural network-based adaptive controller design of robotic manipulators with an observer", *IEEE Transactions on Neural Networks*, **12**(1), pp 54-67 (2001).

17. Morino, P., Milano, M. and Vasca, F. "Linear quadratic state feedback and robust neural network estimator for field-oriented-controlled induction motors", *IEEE Transactions on Industrial Electronics*, **46**(1), pp 150-161 (1999).

18. Pilluta, S. and Keyhani, A. "Development and implementation of neural network observers to estimate the state vector of a synchronous generator from online operating data", *IEEE Transactions on Energy Conversion*, **14**(4), pp 1081-1087 (1999).

19. Powell, J.D., Fekete, N.P. and Chang, C.F. "Observer-based air-fuel ratio control", *IEEE Control Systems Magazine*, **18**(5), pp 72-83 (1998).

20. Webros, P.J. "Backpropagation through time: What it does and how to do it", *Proc. IEEE*, **78**, pp 1550-1560 (Oct. 1990).

21. Park, Y.M., Choi, M.S. and Lee, K.Y. "An optimal tracking neuro-controller for nonlinear dynamic systems", *IEEE Transactions on Neural Networks*, **7**(5), pp 1099-1110 (1996).

22. Hagan, M.T., Demuth, H.B. and Baal, M., *Neural Network Design*, PWS Publications (1996).

APPENDIX

Derivation of MLP Jacobian

Suppose there exists a two-layer tansig/pureline MLP network where its input and output are $p_{n\times 1}$ and $a_{m\times 1}$, respectively. The MLP Jacobian is given by:

$$\begin{aligned}\frac{d\mathbf{a}}{d\mathbf{p}} =& [\mathrm{diag}(\boldsymbol{\Delta}\mathbf{a}(j))]_{m\times m}\mathbf{W}^2 \\ & \left[\mathrm{diag}\left(1-[\tan\mathrm{sig}(\mathbf{n}_1(j))]^2\right)\right]_{s^1\times s^1}\mathbf{W}^1 \\ & \left[\mathrm{diag}\left(\frac{1}{\boldsymbol{\Delta}\mathbf{p}(j)}\right)\right]_{n\times n},\end{aligned} \tag{A1}$$

where W^1, W^2, b^1 and b^2 are the weights and biases for the first and second layers, respectively, and $\boldsymbol{\Delta}\mathbf{a}$ and $\boldsymbol{\Delta}\mathbf{p}$ are expressed as:

$$\boldsymbol{\Delta}\mathbf{p} = \left\{\begin{matrix}\max(\mathbf{p}(1)) - \min(\mathbf{p}(1)) \\ \max(\mathbf{p}(2)) - \min(\mathbf{p}(2)) \\ \vdots \\ \max(\mathbf{p}(n)) - \min(\mathbf{p}(n))\end{matrix}\right\}_{n\times 1},$$

$$\boldsymbol{\Delta}\mathbf{a} = \left\{\begin{matrix}\max(\mathbf{a}(1)) - \min(\mathbf{a}(1)) \\ \max(\mathbf{a}(2)) - \min(\mathbf{a}(2)) \\ \vdots \\ \max(\mathbf{a}(m)) - \min(\mathbf{a}(m))\end{matrix}\right\}_{m\times 1}, \tag{A2}$$

Gradient of MLP Output with Respect to its Weights and Biases

This gradient can be expressed as:

$$\begin{aligned}\frac{\partial\mathbf{a}}{\partial\mathbf{W}} =& \frac{1}{2}[\mathrm{diag}(\boldsymbol{\Delta}\mathbf{a})]_{s^2\times s^2}\left[\left[\frac{\partial\mathbf{n}_2}{\partial\mathbf{W}^1}\right]\left[\frac{\partial\mathbf{n}_2}{\partial\mathbf{b}^1}\right]\right. \\ & \left.\left[\frac{\partial\mathbf{n}_2}{\partial\mathbf{W}^2}\right]\left[\frac{\partial\mathbf{n}_2}{\partial\mathbf{b}^2}\right]\right]_{s^2\times(n\times s^1+s^1+s^2\times s^1+s^2)},\end{aligned} \tag{A3}$$

where $\mathbf{W}$ is a vector made by arranging the elements of W_1, W_2, b_1 and b_2 and one has:

$$s^2 = m,$$

$$\left[\frac{\partial\mathbf{n}_2}{\partial\mathbf{b}^2}\right] = \begin{bmatrix}1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1\end{bmatrix}_{S^2\times S^2},$$

$$\left[\frac{\partial \mathbf{n}_2}{\partial \mathbf{W}^2}\right]=\left[\left[\frac{\partial \mathbf{n}_2}{\partial \mathbf{W}^2(1)}\right]_{S^2\times S^1}\left[\frac{\partial \mathbf{n}_2}{\partial \mathbf{W}^2(2)}\right]_{S^2\times S^1}\cdots\left[\frac{\partial \mathbf{n}_2}{\partial \mathbf{W}^2(S^2)}\right]_{S^2\times S^1}\right]_{S^2\times(S^2\times S^1)}, \quad \text{(A4)}$$

$$\left[\frac{\partial \mathbf{n}_2}{\partial \mathbf{W}^2(j)}\right]=\begin{bmatrix}\mathbf{0}_{1\times S^1}\\ \vdots\\ \mathbf{a}_1^T\rightarrow j^{\text{th}}\text{ row}\\ \vdots\\ \mathbf{0}_{1\times S^1}\end{bmatrix}_{S^2\times S^1}, \quad \text{(A5)}$$

$$\left[\frac{\partial \mathbf{n}_2}{\partial \mathbf{b}^1}\right]=\mathbf{W}^2\left[\text{diag}\left(1-[\text{tansig}\,(\mathbf{n}_1(j))]^2\right)\right]_{S^1\times S^1}, \quad \text{(A6)}$$

$$\left[\frac{\partial \mathbf{n}_2}{\partial \mathbf{W}^1}\right]_{S^2\times(S^1\times n)}=\mathbf{W}^2\left[\text{diag}\left(1-[\text{tansig}\,(\mathbf{n}_1(j))]^2\right)\right]_{S^1\times S^1}\left[\frac{\partial \mathbf{n}_1}{\partial \mathbf{W}^1}\right],$$

$$\left[\frac{\partial \mathbf{n}_1}{\partial \mathbf{W}^1}\right]=\left[\left[\frac{\partial \mathbf{n}_1}{\partial \mathbf{W}^1(1)}\right]_{S^1\times n}\left[\frac{\partial \mathbf{n}_1}{\partial \mathbf{W}^1(2)}\right]_{S^1\times n}\cdots\left[\frac{\partial \mathbf{n}_1}{\partial \mathbf{W}^1(S^1)}\right]_{S^1\times n}\right]_{S^1\times(S^1\times n)},$$

$$\left[\frac{\partial \mathbf{n}_1}{\partial \mathbf{W}^1(j)}\right]=\begin{bmatrix}\mathbf{0}_{1\times n}\\ \vdots\\ \mathbf{p}^T\rightarrow j^{\text{th}}\text{ row}\\ \vdots\\ \mathbf{0}_{1\times n}\end{bmatrix}_{S^1\times n}. \quad \text{(A7)}$$