Design of a General-Purpose MIMO Predictor with Neural Networks*

XIANZHONG CUI AND KANG G. SHIN**
Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122

ABSTRACT: A new multi-step predictor for multiple-input, multiple-output (MIMO) systems is proposed. The output prediction of such a system is represented as a mapping from its historical data and future inputs to future outputs. A neural network is designed to learn the mapping without requiring *a priori* knowledge of the parameters and structure of the system. The major problem in developing such a predictor is how to train the neural network. In case of the back propagation algorithm, the network is trained by using the network's output error which is not known due to the unknown predicted future system outputs. To overcome this problem, the concept of updating, instead of training, a neural network is introduced and verified with simulations. The predictor then uses only the system's historical data to update the configuration of the neural network and always works in a closed loop.

If each node can only handle scalar operations, emulation of an MIMO mapping requires the neural network to be excessively large, and it is difficult to specify some known coupling effects of the predicted system. So, we propose a vector-structured, multilayer perceptron for the predictor design. MIMO linear, nonlinear, time-invariant, and time-varying systems are tested via simulation, and all showed very promising performances.

INTRODUCTION

THERE are numerous industrial applications that require on-line prediction of the system output. For example, load forecasting in an electric power system is essential for an economical dispatch of electricity being generated. Automatic tracking of a flying object is the first step for fire control. In a power plant, prediction of the temperature and pressure at the outlet of a boiler is very useful to operators, especially during the period of startup and shutdown. Moreover, output prediction plays a vital role in predictive control.

It is not difficult to design a predictor for linear single-input single-output (SISO) systems; for example, a self-tuning predictor is designed to predict the Si-content of pig iron for a blast furnace (Keyser and Cauvenberghe, 1981). However, it is a difficult task to design a multi-step predictor for a multiple-input multiple-output (MIMO) system. This is especially true for those systems with nonlinear, time-varying dynamics and/or long system time delays. The dynamic equation of such a system is in general very difficult to derive, since the dynamic parameters are usually unknown, and/or even the internal structure of the system dynamics is sometimes unknown. No general method is known to exist for the design of a predictor for such systems.

Fortunately, neural networks (NNs) seem to shed light on solving this problem. Weigend et al. presented a good example of using NNs for time series prediction in which the NN was trained to predict the time series of the sunspot and a computational ecosystem (Weigend, Huberman and Rumelhart, 1990). A major advantage of using an NN is that it can represent any specified mapping with a learned configuration. A multilayer perceptron with sufficiently many nodes is known to be able to approximate *any* continuous mapping (Lippmann, 1987; Barron, 1989). As a result, many NN applications have been focusing on optimization and retrieval/classification problems (Kung and Hwang, 1989). The output prediction of a system can be viewed as the mapping from the system's historical data and future inputs to future outputs, though it cannot be represented in an analytical form. The main intent of this paper is to design a general-purpose MIMO predictor for such systems using NNs.

One of the well-developed NNs is a multilayer perceptron with the back propagation (BP) training algorithm (Rumelhart and McCelland, 1986; Werbos, 1988). However, most NN applications are in the mode of train-first-and-then-operate; that is, the NN is trained with a set of training data before putting it in operation. After the NN becomes "well-

*The work reported in this paper was supported in part by the National Science Foundation under Grant No. DMC-8721492. Any opinions, findings, and recommendations expressed in this publication are those of the authors and do not necessarily reflect the view of the NSF.
**Author to whom correspondence should be addressed.

1045-389X/94/02 0198-13 $6.00/0

trained", the weights of the NN will no longer be changed. This working mode is called the *"training-operation"* mode. For example, a multilayer perceptron was used in Bhat and McAvoy (1989) as an SISO predictor to predict the pH value of a stirred tank reactor and this was then used as a basis for the design of a predictive controller. The moving-window concept was adopted and the weights of the NN would no longer be adjusted after the training period. The predictor proposed in Weigend, Huberman and Rumelhart (1990) also worked in the training-operation mode. However, this mode may not be suitable for the control or prediction of some time-varying industrial processes. If adjusting NN's weights is viewed as a feedback from the NN's output error, then this feedback loop would be broken when the NN becomes "well-trained". (Obviously, this mode cannot be applied to time-varying systems.) To remedy this problem, an NN should be *updated,* rather than trained, that is, the weights of the NN should be adjusted on-line in order to keep track of the variation of a system. We call such a working mode as the *"updating"* mode. Updating an NN is essential to the design of an MIMO predictor for time-varying systems.

In the standard BP algorithm, the weights of the NN are adjusted by minimizing the network's output error. However, when an NN is used as a predictor, its output error is unknown since the future outputs of the predicted system are not known. We have designed a predictor which is updated only by using the historical data of the predicted system and does not require the knowledge of the dynamic parameters nor the structure of the predicted system. Weights of the NN are dynamically adjusted to deal with the effects of nonlinear, time-varying properties, and/or long system time delays. Because the NN-based predictor will always work in a closed loop, component failures in the NN will be learned and the NN will subsequently be re-configured, improving system reliability. Furthermore, the parallel processing structure of the NN makes it suitable for high-dimensional systems.

Each node of an NN is usually designed to perform only scalar operations. However, for an MIMO system, if each node can only handle scalar operations, the size of NN may become too large to manage. We therefore propose to equip each node of the NN with the ability of vector operations in order to easily specify some known coupling relations within the predicted system and to get an easier (thus more intuitive) form of the training algorithm. This vector structured multilayer perceptron will form the backbone of the proposed MIMO predictor.

This paper is organized as follows. In the second section, the basic structure of the NN-based predictor is described and the input-output mapping of an MIMO system is analyzed. The third section focuses on the problem of tracking time-varying systems by updating the NN. The scaling problem and error analysis of the NN-based predictor are also addressed in this section. A multi-dimensional back propagation algorithm is developed in the fourth section; it is an extension of the scalar version presented in Rumelhart and

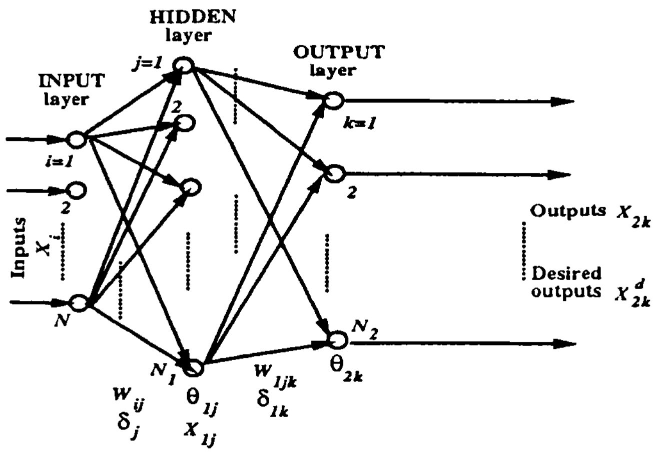

Figure 1. Basic structure of a three-layer perceptron.

McCelland (1986). Simulation results are presented in the fifth section for various systems: MIMO linear, nonlinear, time-invariant, and time-varying. Finally, the sixth section concludes the paper.

BASIC STRUCTURE OF AN MIMO NN-BASED PREDICTOR

The standard structure of a three-layer perceptron (with one hidden layer) is shown in Figure 1. With the ability of learning from examples of a mathematical mapping, an NN can be trained to attain the dynamical property of the mapping. Typically, a set of input-output pairs $\{(u_i,y_i)\,|\,y_i = G(u_i),\ i = 1,2,\ldots\}$ are used as *training data,* where G is the input-output mapping. These training data are used to adjust the weights of the NN that represents the input-output mapping. One of the popular training algorithms is BP, which attempts to approximate a mapping in the sense of least mean squares (Lippmann, 1987). The computation of an NN with the BP algorithm consists of two steps: computing the output of the NN forward from the INPUT layer to the OUTPUT layer, and adjusting the weights backward from the OUTPUT layer to INPUT layer. The computation at each node is shown in Figure 2 with $X_j = f(\Sigma_{i=1}^{L} W_{ij}X_i + \theta_j)$, where X_j is the output of node j, X_i the input from node i, W_{ij} the weight on the arc from node i to node j, θ_j the

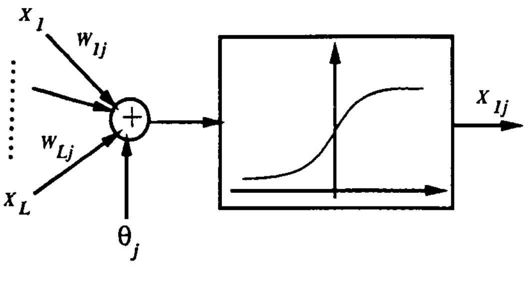

Figure 2. Basic structure of a neuron.

threshold at node j, and $f(.)$ is a sigmoid function. Once a new output at the OUTPUT layer is ready and denoted by X_{2k}, the network-output error is computed as $E_k = X_{2k}^d - X_{2k}$, where X_{2k}^d is the desired output of the NN. The weights of each layer are then modified according to the network-output error E_k; in other words, this training process is driven by the NN's output error. It is proven in Cybenko (1989) that a multilayer perceptron with two hidden layers and sufficient many hidden nodes can approximate *any* input-output mapping.

If the predicted system is causal, then the dynamic relationship between input and output can be conceptually represented as

$$Y(k) = \mathbf{h}(Y(k-i), U(k-j), k) \tag{1}$$

where $Y \in \mathbf{R}^p$ is the output vector, $U \in \mathbf{R}^n$ the input vector, $k \in \mathbf{Z}$ the discrete time index, and $\mathbf{h}:\mathbf{R}^p \times \mathbf{R}^n \times \mathbf{Z} \rightarrow \mathbf{R}^p$. $Y(k-i)$, $i = 1,2.,\ldots$ is the historical data of the system output. $U(k-j)$, $j = 0,1,2,\ldots$ is the system inputs at and before time k. If the mapping $\mathbf{h}$ in Equation (1) were known, the future output of the system could be computed step-by-step by predicting the future inputs $U(k+d)$, $d = 1,2,\ldots$. However, it is usually very difficult, if not impossible, to derive a closed-form expression for Equation (1) (even if such a closed-form expression exists). The d-step ahead prediction of $Y(k)$ at time k can be represented as

$$Y(k + d/k) = \mathbf{h}_p(\bar{\mathbf{Y}},\bar{\mathbf{U}},k) \tag{2}$$

where

$$\begin{aligned}\bar{\mathbf{Y}} &= [Y(k), Y(k-1),\ldots, Y(k-i)]\\ \bar{\mathbf{U}} &= [U(k+i_1),\ldots, U(k), U(k-1),\ldots, U(k-i_2)]\\ &\mathbf{h}_p:\mathbf{R}^p \times \mathbf{R}^n \times \mathbf{Z} \rightarrow \mathbf{R}^p\end{aligned}$$

where i, i_1 and i_2 are positive integers. The parameters and/or structure of this mapping are not known either. It is therefore practically impossible to design a general-purpose predictor with mathematical synthesis alone, though a closed-form expression for such an MIMO mapping may sometimes exist. So, we propose to design an NN which will learn the mapping Equation (2). A three-layer perceptron is used for this purpose, where the inputs are $\bar{\mathbf{Y}}$ and $\bar{\mathbf{U}}$, and the output is $\hat{Y}(k + d/k)$, $d = 1,2,\ldots$. There are two major problems in implementing this scheme.

The first problem is related to the training process. In the standard BP algorithm, the NN should be trained by minimizing the NN's output errors. For our case, however, the desired values of the NN's output are the system outputs, $Y(k+d)$, $d = 1,2,\ldots$. The network-output errors are then the system prediction errors which are computed as

$$e_p(k+d) = Y(k+d) - \hat{Y}(k + d/k)$$

This implies that the network should be trained by using the system's *unknown* future outputs $Y(k+d)$. A set of training data can be acquired beforehand, and used to train the NN. After the NN is "well trained", the NN will no longer modify its weights and produce the output, while the inputs are present at the INPUT nodes. For time-varying mappings, however, it is meaningless to say that an NN is "well trained". Moreover, as pointed out in the previous section, this training-operation mode implies that the NN work in an open loop after the training, which is not acceptable in a real-time control or prediction system. Therefore, our MIMO predictor needs an NN that is updated on-line in order to keep track of a time-varying mapping and to work always in a closed loop. More on this will be discussed in the next section.

The second problem is how to efficiently and clearly represent the training algorithm for an MIMO system. Note that Equation (2) is an MIMO mapping. If each node of the NN can only handle scalar operations, then we need a fully-connected multilayer mesh. Each node at the INPUT or OUTPUT layer of this mesh corresponds to an element of the input or output vector of the mapping. By using the standard BP algorithm, it is difficult to express some known coupling relations within the mapping and obtain a set of succinct formulas for the training algorithm. Therefore, we need an NN which can handle vector operations in order to reduce the total number of the nodes required. Such an NN will be discussed in the fourth section.

TRACKING A TIME-VARYING SYSTEM AND ERROR ANALYSIS

Training Algorithm for the Updating Mode

Suppose the predicted system is an SISO system with output $y(t)$ and its d-step ahead prediction $\hat{y}(t + d/t)$ at time t (t is the continuous-time index). Let $X_{2k}(t)$ and $X_{2k}^d(t)$ be respectively the actual output of the NN-based predictor and its desired value at time t. Then the network-output error is computed by

$$E_k(t) = X_{2k}^d(t) - X_{2k}(t)$$

In the standard BP algorithm, we must use this network-output error to train the NN. However, when the NN is used as a predictor, the network's output is the prediction of the system output,† and the network-output error is the prediction error:

$$E_k(t) = X_{2k}^d(t) - X_{2k}(t) = y(t+d) - \hat{y}(t + d/t) \tag{3}$$

†In fact, the network's output X_{2k} is a scaled value of the system's output prediction $\hat{y}$. At this stage, it is assumed that $\hat{y}$ is within the range of (0,1). The scaling problem will be discussed later.

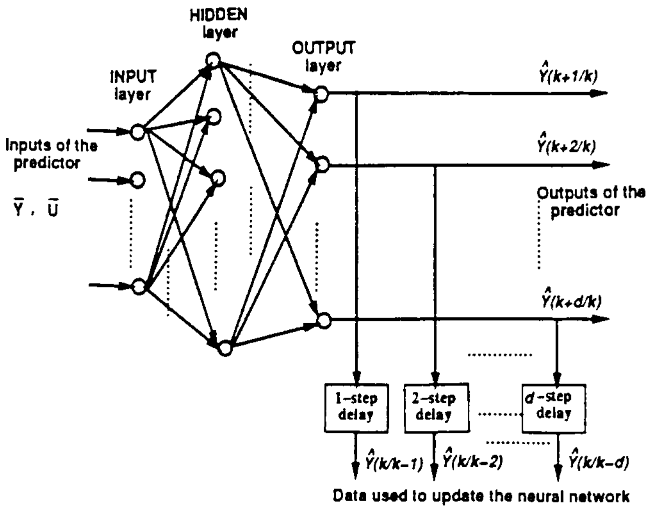

Figure 3. Basic structures of the NN-based multi-step predictor.

$E_k(t)$ is unavailable since the system's future output $y(t + d)$ is not available at time t. Hence, we must use the system's historical data to update the NN-based predictor on-line in order to maintain the closed-loop operation by keeping track of a time-varying system. To update the NN-based predictor, instead of using Equation (3), we propose to use a posterior prediction error:

$$E_k(t - d) = X^d_{2k}(t - d) - X_{2k}(t - d)$$
$$= y(t) - \hat{y}(t/t - d)$$

This arrangement is equivalent to cascading the NN with delay elements, as shown in Figure 3. In what follows, a modified BP algorithm is derived to handle these delay elements. The formulas are given for SISO systems, which will be extended to the MIMO case in the next section.

1. Compute the Output of the HIDDEN Layer, X_{1j}

The HIDDEN layer's outputs are

$$X_{1j}(t) = \frac{1}{1 + \exp(-O_{1j} - \theta_{1j})}$$

where $O_{1j} = \sum_{i=1}^{N} W_{ij}X_i(t), \quad j = 1,2,\ldots,N_1$

X_i is the input at INPUT node i, W_{ij} is the weight from INPUT node i to HIDDEN node j, and θ_{1j} is the threshold at HIDDEN node j.

2. Compute the Output of the OUTPUT Layer, X_{2k}

The outputs of the OUTPUT layer are

$$X_{2k}(t) = \frac{1}{1 + \exp(-O_{2k} - \theta_{2k})} \tag{4}$$

where $O_{2k} = \sum_{j=1}^{N_1} W_{1jk}X_{1j}(t), \quad k = 1,2,\ldots,N_2$

W_{1jk} is the weight from HIDDEN node j to OUTPUT node k, and θ_{2k} is the threshold at OUTPUT node k.

3. Update the Weights from the HIDDEN to OUTPUT Layer, W_{1jk}

We want to use the delayed data $X_{2k}(t - d)$ to update the network. The cost function of the network is defined by

$$E(t) = \frac{1}{2}\sum_{k=1}^{N_2}(E_k(t - d))^2$$
$$= \frac{1}{2}\sum_{k=1}^{N_2}(X^d_{2k}(t - d) - X_{2k}(t - d))^2 \tag{5}$$

Let the updated weights be $W_{1jk}(t + \Delta t) = W_{1jk}(t) + \Delta W_{1jk}$. Using the gradient algorithm,

$$\Delta W_{1jk} \propto -\frac{\partial E(t)}{\partial W_{1jk}} = -\frac{\partial E(t)}{\partial O_{2k}}\frac{\partial O_{2k}}{\partial W_{1jk}} \tag{6}$$

From Equation (4), we have

$$\frac{\partial E(t)}{\partial O_{2k}} = \frac{\partial E(t)}{\partial X_{2k}(t)}\frac{\partial X_{2k}(t)}{\partial O_{2k}} = \frac{\partial E(t)}{\partial X_{2k}(t)}X_{2k}(t)(1 - X_{2k}(t))$$

and

$$\frac{\partial O_{2k}}{\partial W_{1jk}} = \frac{\partial}{\partial W_{1jk}}\left(\sum_{l=1}^{N_1} W_{1lk}X_{1l}(t)\right) = X_{1j}(t)$$

Then, we get

$$\frac{\partial E(t)}{\partial W_{1jk}} = \frac{\partial E(t)}{\partial X_{2k}(t)}X_{2k}(t)(1 - X_{2k}(t))X_{1j}(t) \tag{7}$$

Because

$$\frac{\partial E(t)}{\partial X_{2k}(t)} = \frac{\partial E(t)}{\partial X_{2k}(t - d)}\frac{\partial X_{2k}(t - d)}{\partial X_{2k}(t)}$$
$$= \frac{1}{2}\frac{\partial}{\partial X_{2k}(t - d)}\left(\sum_{l=1}^{N_2}(X^d_{2l}(t - d) - X_{2l}(t - d))^2\right)\frac{\partial X_{2k}(t - d)}{\partial X_{2k}(t)}$$
$$= -(X^d_{2k}(t - d) - X_{2k}(t - d))\frac{\partial X_{2k}(t - k)}{\partial X_{2k}(t)}$$

Let

$$\delta_{1k} = (X^d_{2k}(t-d) - X_{2k}(t-d))X_{2k}(t)(1 - X_{2k}(t)) \tag{8}$$

then Equation (7) becomes

$$\frac{\partial E(t)}{\partial W_{1jk}} = -\delta_{1k}X_{1j}(t)\frac{\partial X_{2k}(t-d)}{\partial X_{2k}(t)} \tag{9}$$

Now, the problem is how to compute $[\partial X_{2k}(t-d)]/\partial X_{2k}(t)$. Because $X_{2k}(t) = \hat{y}(t + d/t)$ is the d-step ahead prediction and $X_{2k}(t-d) = \hat{y}(t/t-d)$ is the same value of $\hat{y}(t + d/t)$ delayed by d, we conclude

$$\frac{\partial X_{2k}(t-d)}{\partial X_{2k}(t)} = 1, \quad \forall\, t \text{ and } d \tag{10}$$

In Equation (6), we set $\Delta W_{1jk} = -\eta_1[\partial E(t)/\partial W_{1jk}]$, $\eta_1 > 0$ is a gain factor. Therefore, using the results of Equations (9) and (10), we get

$$W_{1jk}(t + \Delta t) = W_{1jk}(t) + \Delta W_{1jk}$$

$$\Delta W_{1jk} = \eta_1\delta_{1k}X_{1j}(t)$$

where δ_{1k} is given by Equation (8). This has the same form as the standard BP algorithm, but the definition of the cost function in Equation (5) is different. Similarly to the above process, other formulas are listed below without any detailed account.

4. Update the Weights from the INPUT to HIDDEN Layer, W_{ij}

$$W_{ij}(t + \Delta t) = W_{ij}(t) + \Delta W_{ij}$$

where $\Delta W_{ij} = \eta\delta_j X_i(t)$

$$\delta_j = \left[\sum_{k=1}^{N_2} \delta_{1k}W_{1jk}\right]X_{1j}(t)(1 - X_{1j}(t)) \tag{11}$$

where $\eta > 0$ is a gain factor.

5. Update the Thresholds θ_{2k} and θ_{1j}

$$\theta_{2k}(t + \Delta t) = \theta_{2k}(t) + \eta_{1\theta}\delta_{1k}$$

$$\theta_{1j}(t + \Delta t) = \theta_{1j}(t) + \eta_\theta\delta_j$$

where $\eta_{1\theta} > 0$ and $\eta_\theta > 0$ are gain factors, and δ_{1k}, δ_j are given by Equations (8) and (11), respectively.

Scaling Problem and Error Analysis

In the NN, the output of each node passes through a sigmoid function (Figure 2) which forces the output of each node to be within the range of (0,1). Suppose the output of a predicted system is $y(k) \in (y_{min}, y_{max})$ and a linear scaling formula is used, then the scaled value of $y(k)$ is given as

$$y_s(k) = \frac{y(k) - y_{min}}{y_{max} - y_{min}} \tag{12}$$

The output of the NN-based predictor is the scaled value of d-step ahead prediction $X_{2k}(k) = \hat{y}_s(k + d/k)$. Then the unscaled values are

$$y(k) = y_s(k)(y_{max} - y_{min}) + y_{min} \tag{13}$$

$$\hat{y}(k + d/k) = \hat{y}_s(k + d/k)(y_{max} - y_{min}) + y_{min} \tag{14}$$

From Equations (13) and (14), we get the absolute prediction error

$$e_p(k) = y(k) - \hat{y}(k/k-d) = e_{ps}(k)(y_{max} - y_{min}) \tag{15}$$

where $e_{ps}(k) = y_s(k) - \hat{y}_s(k/k-d) = X^d_{2k}(k-d) - X_{2k}(k-d)$ is the absolute output error of the NN-based predictor.

The relative prediction error is defined as

$$\Delta e_p(k) \equiv \left|\frac{y(k) - \hat{y}(k/k-d)}{y_{max} - y_{min}}\right| = \left|\frac{e_p(k)}{y_{max} - y_{min}}\right|$$

Similarly, the relative output error of the NN is defined as

$$\Delta e_{ps}(k) \equiv \left|\frac{y_s(k) - \hat{y}_s(k/k-d)}{1.0}\right| \tag{16}$$

Substituting the scaling formula Equation (12) into Equation (16), we get

$$\Delta e_{ps}(k) = \left|\frac{y(k) - \hat{y}(k/k-d)}{y_{max} - y_{min}}\right| = \Delta e_p(k) \tag{17}$$

From Equations (15) and (17), we can observe that the unscaled absolute prediction error may be much larger than the scaled one, but the unscaled relative prediction error is the same as the scaled one. This indicates that the accuracy of the NN-based predictor depends only on the accuracy of the NN's approximation to the actual mapping. Finally, y_{max} and y_{min} should be determined as

$$y_{max} = \sup\{y(k) \mid \text{for all } k \geq 0\}$$

$$y_{min} = \inf\{y(k) \mid \text{for all } k \geq 0\}$$

such that the range of the scaled value approaches (0,1) in order to excite the network.

MULTI-DIMENSIONAL BACK PROPAGATION ALGORITHM

Each node within a conventional NN is designed only to handle scalar operations, and the number of the nodes or layers is increased to deal with a more complex I/O mapping. However, the NN's size required for a complex MIMO mapping, may become too large to manage. Alternatively, one can equip each node of an NN with the ability of vector operations, because this is easier to express some known coupling relations and will result in a set of succinct formulas. So, all inputs and outputs of this NN are vectors. Referring to Figure 1, let $X_i \in \mathbf{R}^n$, $X_{1j} \in \mathbf{R}^m$, and $X_{2k} \in \mathbf{R}^p$ be the output of the INPUT, HIDDEN and OUTPUT layer, respectively, for $1 \le i \le N$, $1 \le j \le N_1$, and $1 \le k \le N_2$. Extension of the BP algorithm to a vector form is presented below.

1. Compute the Output of the HIDDEN Layer, X_{1j}

The output of the HIDDEN layer is computed by

$$X_{1j} \equiv [x_{1j1},\ldots,x_{1jm}]^T$$
$$= f_j(O_{1j}) = \left[\frac{1}{1+\exp(-o_{1j1}-\theta_{1j1})},\ldots,\frac{1}{1+\exp(-o_{1jm}-\theta_{1jm})}\right]^T \quad (18)$$

$$O_{1j} = \sum_{i=1}^{N} \mathbf{W}_{ij} X_i, \quad j = 1,2,\ldots,N_1 \quad (19)$$

where $\mathbf{W}_{ij} \in \mathbf{R}^{m\times n}$ is the weight matrix from node i of the INPUT layer to node j of the HIDDEN layer, $f_j:\mathbf{R}^m \to \mathbf{R}^m$ is defined as a sigmoid function of each component of a vector, and $\Theta_{1j} \equiv [\theta_{1j1},\ldots,\theta_{1jm}]^T$ is the threshold vector at node j of the HIDDEN layer.

2. Compute the Output of the OUTPUT Layer, X_{2k}

The output of the OUTPUT layer is

$$X_{2k} \equiv [x_{2k1},\ldots,x_{2kp}]^T$$
$$= f_k(O_{2k}) = \left[\frac{1}{1+\exp(-o_{2k1}-\theta_{2k1})},\ldots,\frac{1}{1+\exp(-o_{2kp}-\theta_{2kp})}\right]^T \quad (20)$$

$$O_{2k} = \sum_{j=1}^{N_1} \mathbf{W}_{1jk} X_{1j}, \quad k = 1,2,\ldots,N_2 \quad (21)$$

where $\mathbf{W}_{1jk} \in \mathbf{R}^{p\times m}$ is the weight matrix from node j of the HIDDEN layer to node k of the OUTPUT layer, $f_k:\mathbf{R}^p \to \mathbf{R}^p$, is defined as a sigmoid function of each component of a vector, and $\Theta_{2k} \equiv [\theta_{2k1},\ldots,\theta_{2kp}]^T$ is the threshold vector at node k of the OUTPUT layer. Note that if $m = p = n$, $\mathbf{W}_{ij} = diag[w_{11},\ldots,w_{nn}]_{ij}$ and $\mathbf{W}_{1jk} = diag[w_{111},\ldots,w_{1nn}]_{jk}$, then the system is uncoupled.

3. Update the Weights from the HIDDEN to OUTPUT Layer, $\mathbf{W}_{1jk}$

Let the desired output of the network be $X^d_{2k} = [x^d_{2k1},\ldots,x^d_{2kp}]^T$, and let the output error of node k be defined as $E_k = X^d_{2k} - X_{2k}$ and the cost function be defined as

$$E = \frac{1}{2}\sum_{k=1}^{N_2} E_k^T E_k = \frac{1}{2}\sum_{k=1}^{N_2} (X^d_{2k} - X_{2k})^T (X^d_{2k} - X_{2k}) \quad (22)$$

We want to update $\mathbf{W}_{1jk}$ and $\mathbf{W}_{ij}$ by minimizing E and taking the form of

$$\mathbf{W}_{1jk}(t+\Delta t) = \mathbf{W}_{1jk}(t) + \Delta\mathbf{W}_{1jk} \quad (23)$$

$$\mathbf{W}_{ij}(t+\Delta t) = \mathbf{W}_{ij}(t) + \Delta\mathbf{W}_{ij} \quad (24)$$

where t is the continuous-time index.

By using the gradient algorithm, we should set

$$\Delta\mathbf{W}_{1jk} \propto -\left[\frac{\partial E}{\partial \mathbf{W}_{1jk}}\right]^T = -\left[\frac{\partial E}{\partial O_{2k}}\mathrm{T}_1\right]^T \quad (25)$$

where $(\partial E/\partial O_{2k}) \in \mathbf{R}^{1\times p}$, and $\mathrm{T}_1 \equiv (\partial O_{2k}/\partial \mathbf{W}_{1jk}) \in \mathbf{R}^{p\times m\times p}$ is a three-dimensional tensor since $O_{2k} \in \mathbf{R}^{p\times 1}$ and $\mathbf{W}_{1jk} \in \mathbf{R}^{p\times m}$.

To compute $\partial E/\partial O_{2k}$, referring to Equation (22), we get

$$\frac{\partial E}{\partial O_{2k}} = \frac{\partial E}{\partial X_{2k}}\frac{\partial X_{2k}}{\partial O_{2k}} = -(X^d_{2k} - X_{2k})^T\frac{\partial X_{2k}}{\partial O_{2k}} \quad (26)$$

From Equation (20), we obtain

$$\frac{\partial X_{2k}}{\partial O_{2k}} = \frac{\partial f_k(O_{2k})}{\partial O_{2k}} = \begin{bmatrix} \frac{\partial x_{2k1}}{\partial o_{2k1}} & \cdots & \frac{\partial x_{2k1}}{\partial o_{2kp}} \\ \cdots & \cdots & \cdots \\ \frac{\partial x_{2kp}}{\partial o_{2k1}} & \cdots & \frac{\partial x_{2kp}}{\partial o_{2kp}} \end{bmatrix} \quad (27)$$

Because $x_{2kl} = 1/[1+\exp(-o_{2kl}-\theta_{2kl})]$ and $\partial x_{2kl}/\partial o_{2kl} = x_{2kl}(1-x_{2kl})$, Equation (27) can be written in the form

$$\frac{\partial X_{2k}}{\partial O_{2k}} = diag[x_{2k1}(1-x_{2k1}),\ldots,x_{2kp}(1-x_{2kp})] \quad (28)$$

Therefore, substituting Equation (28) into Equation (26) and using the notation $\boldsymbol{\delta}_{1k}$, we get

$$\begin{aligned}\boldsymbol{\delta}_{1k} &\equiv -\frac{\partial E}{\partial \boldsymbol{O}_{2k}} = (\boldsymbol{X}_{2k}^d - \boldsymbol{X}_{2k})^T \frac{\partial \boldsymbol{X}_{2k}}{\partial \boldsymbol{O}_{2k}} \\ &= (\boldsymbol{X}_{2k}^d - \boldsymbol{X}_{2k})^T \, diag[x_{2k1}(1 - x_{2k1}), \ldots, \\ &\quad x_{2kp}(1 - x_{2kp})] \in \mathbf{R}^{1\times p} \end{aligned} \tag{29}$$

To compute T_1, from Equation (21), we have the lth component of $\boldsymbol{O}_{2k}$ as

$$o_{2kl} = \sum_{j=1}^{N_1} (\boldsymbol{W}_l)_{1jk} \boldsymbol{X}_{1j}, \quad l = 1,2,\ldots,p$$

where $(\boldsymbol{W}_l)_{1jk}$ is the lth row of $\mathbf{W}_{1jk}$. Then, the lth matrix of T_1, $\mathbf{T}_{1l} \equiv \partial o_{2kl}/\partial \mathbf{W}_{1jk} \in \mathbf{R}^{m\times p}$, $l = 1,2,\ldots,p$, has the form of

$$\mathbf{T}_{1l} \equiv \frac{\partial o_{2kl}}{\partial \mathbf{W}_{1jk}} = \begin{bmatrix} \mathbf{0} \\ \vdots \\ \dfrac{\partial}{\partial (\boldsymbol{W}_l)_{1jk}^T}\left(\sum_{q=1}^{N_1} (\boldsymbol{W}_l)_{1qk}\boldsymbol{X}_{1q}\right) \\ \mathbf{0} \\ \vdots \end{bmatrix}^T$$

$$= \begin{bmatrix} \mathbf{0} \\ \vdots \\ \dfrac{\partial}{\partial (\boldsymbol{W}_l)_{1jk}^T}((\boldsymbol{W}_l)_{1jk}\boldsymbol{X}_{1j}) \\ \mathbf{0} \\ \vdots \end{bmatrix}^T = \begin{bmatrix} \mathbf{0} \\ \vdots \\ (\boldsymbol{X}_{1j})^T \\ \mathbf{0} \\ \vdots \end{bmatrix}^T \begin{matrix} \\ \\ \leftarrow \text{at the } l\text{th row} \\ \\ \\ \end{matrix} \tag{30}$$

In Equation (25), we set $\Delta \boldsymbol{W}_{1jk} = -\eta_1[\partial E/\partial \mathbf{W}_{1jk}]^T$ where $\eta_1 > 0$ is the gain factor of the OUTPUT layer. Therefore, from Equations (29) and (30), we get

$$\Delta \mathbf{W}_{1jk} = -\eta_1 \left[\frac{\partial E}{\partial \boldsymbol{O}_{2k}} \mathrm{T}_1\right]^T = \eta_1[\boldsymbol{\delta}_{1k}\mathrm{T}_1]^T \tag{31}$$

4. Update the Weights from the INPUT to HIDDEN Layer, $\mathbf{W}_{ij}$

According to the gradient algorithm, we should set

$$\Delta \mathbf{W}_{ij} \propto -\left[\frac{\partial E}{\partial \mathbf{W}_{ij}}\right]^T = -\left[\frac{\partial E}{\partial \boldsymbol{O}_{1j}}\mathrm{T}\right]^T \tag{32}$$

where $\partial E/\partial \boldsymbol{O}_{1j} \in \mathbf{R}^{1\times m}$, and $\mathrm{T} \equiv \partial \boldsymbol{O}_{1j}/\partial \mathbf{W}_{ij} \in \mathbf{R}^{m\times n\times m}$ is a three-dimensional tensor, since $\boldsymbol{O}_{ij} \in \mathbf{R}^{m\times 1}$ and $\mathbf{W}_{ij} \in \mathbf{R}^{m\times n}$.

To compute $\partial E/\partial \boldsymbol{O}_{1j}$, we have

$$\frac{\partial E}{\partial \boldsymbol{O}_{1j}} = \frac{\partial E}{\partial \boldsymbol{X}_{1j}} \frac{\partial \boldsymbol{X}_{1j}}{\partial \boldsymbol{O}_{1j}} \tag{33}$$

using Equations (19) and (18) leads to

$$\begin{aligned}\frac{\partial \boldsymbol{X}_{1j}}{\partial \boldsymbol{O}_{1j}} &= \frac{\partial f_j(\boldsymbol{O}_{1j})}{\partial \boldsymbol{O}_{1j}} \\ &= diag[x_{1j1}(1 - x_{1j1}), \ldots, x_{1jm}(1 - x_{1jm})] \end{aligned} \tag{34}$$

Because

$$\frac{\partial E}{\partial \boldsymbol{X}_{1j}} = \sum_{k=1}^{N_2} \frac{\partial E}{\partial \boldsymbol{O}_{2k}} \frac{\partial \boldsymbol{O}_{2k}}{\partial \boldsymbol{X}_{1j}} \tag{35}$$

substituting Equations (21) and (29) into Equation (35) leads to:

$$\begin{aligned}\frac{\partial E}{\partial \boldsymbol{X}_{1j}} &= \sum_{k=1}^{N_2} (-\boldsymbol{\delta}_{1k}) \frac{\partial}{\partial \boldsymbol{X}_{1j}} \left(\sum_{l=1}^{N_1} \mathbf{W}_{1lk}\boldsymbol{X}_{1l}\right) \\ &= \sum_{k=1}^{N_2} (-\boldsymbol{\delta}_{1k})\mathbf{W}_{1jk} \end{aligned} \tag{36}$$

Therefore, substituting Equations (34) and (36) into Equation (33), and using the notation $\boldsymbol{\delta}_j$, we get

$$\begin{aligned}\boldsymbol{\delta}_j &\equiv -\frac{\partial E}{\partial \boldsymbol{O}_{1j}} = -\frac{\partial E}{\partial \boldsymbol{X}_{1j}} \frac{\partial \boldsymbol{X}_{1j}}{\partial \boldsymbol{O}_{1j}} \\ &= \left(\sum_{k=1}^{N_2} \boldsymbol{\delta}_{1k}\mathbf{W}_{1jk}\right) diag[x_{1j1}(1 - x_{1j1}), \ldots, \\ &\quad x_{1jm}(1 - x_{1jm})] \in \mathbf{R}^{1\times m} \end{aligned} \tag{37}$$

To compute T, from Equation (19), we have the lth component of $\boldsymbol{O}_{1j}$ as

$$o_{1jl} = \sum_{i=1}^{N} (\mathbf{W}_l)_{ij}\boldsymbol{X}_i, \quad l = 1,2,\ldots,m$$

where $(\mathbf{W}_l)_{ij}$ is the lth row of $\mathbf{W}_{ij}$. Then, the lth matrix of T, $\mathbf{T}_l \equiv \partial o_{1jl}/\partial \mathbf{W}_{ij} \in \mathbf{R}^{n\times m}$, $l = 1,2,\ldots,m$, has the form of

$$\mathrm{T}_l \equiv \frac{\partial o_{1jl}}{\partial \mathrm{W}_{ij}} = \begin{bmatrix} 0 \\ \vdots \\ \dfrac{\partial}{\partial (W_l)_{ij}^T}\left(\sum_{q=1}^{N} (W_l)_{qj} X_q\right) \\ 0 \\ \vdots \end{bmatrix}^T$$

$$= \begin{bmatrix} 0 \\ \vdots \\ \dfrac{\partial}{\partial (W_l)_{ij}^T}((W_l)_{ij} X_i) \\ 0 \\ \vdots \end{bmatrix}^T = \begin{bmatrix} 0 \\ \vdots \\ (X_i)^T \\ 0 \\ \vdots \end{bmatrix}^T \leftarrow \begin{matrix}\text{at the} \\ l\text{th row}\end{matrix} \tag{38}$$

In Equation (32), we set $\Delta \mathrm{W}_{ij} = -\eta[\partial E/\partial \mathrm{W}_{ij}]^T$, where $\eta > 0$ is the gain factor of the HIDDEN layer. Therefore, from Equations (37) and (38), we conclude

$$\Delta \mathrm{W}_{ij} = -\eta\left[\frac{\partial E}{\partial O_{1j}} \mathrm{T}\right]^T = \eta[\delta_j \mathrm{T}]^T \tag{39}$$

5. *Update the Thresholds of the OUTPUT Layer,* Θ_{2k}

We want to update the thresholds Θ_{2k} and Θ_{1j} by:

$$\Theta_{2k}(t + \Delta t) = \Theta_{2k}(t) + \Delta\Theta_{2k} \tag{40}$$

$$\Theta_{1j}(t + \Delta t) = \Theta_{1j}(t) + \Delta\Theta_{1j} \tag{41}$$

By the gradient algorithm, we should set

$$\Delta\Theta_{2k} \propto -\left[\frac{\partial E}{\partial \Theta_{2k}}\right]^T \tag{42}$$

Using Equations (22), (21) and (20), we get

$$\frac{\partial E}{\partial \Theta_{2k}} = \frac{\partial E}{\partial X_{2k}} \frac{\partial X_{2k}}{\partial \Theta_{2k}} = -(X_{2k}^d - X_{2k})^T \frac{\partial}{\partial \Theta_{2k}}(f_k(O_{2k}))$$

$$= -(X_{2k}^d - X_{2k})^T \, diag[x_{2k1}(1 - x_{2k1}), \ldots, x_{2kp}(1 - x_{2kp})] \tag{43}$$

In Equation (42), we set $\Delta\Theta_{2k} = -\eta_{1\theta}[\partial E/\partial \Theta_{2k}]^T$, where $\eta_{1\theta} > 0$ is the gain factor of the thresholds at the OUTPUT layer. Therefore, from Equations (43) and (29), we get

$$\Delta\Theta_{2k} = \eta_{1\theta}[\delta_{1k}]^T \tag{44}$$

6. *Update the Thresholds of the HIDDEN Layer,* Θ_{1j}

Using the gradient algorithm, we should set

$$\Delta\Theta_{1j} \propto -\left[\frac{\partial E}{\partial \Theta_{1j}}\right]^T \tag{45}$$

Because $\partial E/\partial \Theta_{1j} = (\partial E/\partial X_{1j})(\partial X_{1j}/\partial \Theta_{1j})$, using Equations (36), (19) and (18) we get

$$\frac{\partial E}{\partial \Theta_{1j}} = -\left(\sum_{k=1}^{N_2} \delta_{1k} \mathrm{W}_{1jk}\right) \frac{\partial}{\partial \Theta_{1j}}(f_j(O_{1j}))$$

$$= -\left(\sum_{k=1}^{N_2} \delta_{1k} \mathrm{W}_{1jk}\right) diag[x_{1j1}(1 - x_{1j1}), \ldots, x_{1jm}(1 - x_{1jm})] \tag{46}$$

In Equation (45), we set $\Delta\Theta_{1j} = -\eta_\theta[\partial E/\partial \Theta_{1j}]^T$, where $\eta_\theta > 0$ is the gain factor of the thresholds at the HIDDEN layer. Therefore, from Equations (46) and (37), we get

$$\Delta\Theta_{1j} = \eta_\theta[\delta_j]^T \tag{47}$$

To summarize what we have developed so far, the computation of the multi-dimensional BP algorithm is done as listed below.

1. Compute the output of the HIDDEN layer X_{1j} by Equations (19) and (18).
2. Compute the output of the OUTPUT layer X_{2k} by Equations (21) and (20).
3. Update the weights from the HIDDEN to OUTPUT layer W_{1jk} by Equations (23), (31), (29) and (30).
4. Update the weights from the INPUT to HIDDEN layer W_{ij} by Equations (24), (39), (37) and (38).
5. Update the thresholds of the OUTPUT layer Θ_{2k} by Equations (40), (44) and (29).
6. Update the thresholds of the HIDDEN layer Θ_{1j} by Equations (41), (47) and (37).

Extending the BP algorithm to a vector form shifted the complexity from the network level to the node level. Though the overall computation requirement is not reduced, it results in a set of succinct formulas and is easier to specify the I/O nodes of the NN for an MIMO mapping and to express some known coupling relations. Moreover, if the NN is implemented in software and instructions of vector operations are provided, then the programming is more efficient with this vector form of BP algorithm.

SIMULATION RESULTS

To test the capability of the proposed predictor, a series of simulation experiments are conducted and the main results

Table 1. The relative RMS prediction errors of the linear system.

Sample Intervals for Statistics	Relative RMS Prediction Errors of y_1 (%)				
	Prediction Steps				
	1	2	3	4	5
0–500	20.61574	18.59294	15.80285	15.43578	17.34273
1000–1500	3.50542	2.99628	3.24761	4.43171	5.31734
2500–3000	3.08927	2.26003	2.52720	3.14259	3.43695
3500–4000	2.87110	1.75731	2.20443	2.60701	2.60416
4500–5000	2.68230	1.56914	2.17914	2.38665	2.06842
	Relative RMS Prediction Errors of y_2 (%)				
	Prediction Steps				
	1	2	3	4	5
0–500	17.47703	14.82588	14.74718	16.53000	18.53330
1000–1500	2.41279	2.68619	3.95526	5.09985	4.82806
2500–3000	1.86467	1.95556	2.69478	3.20278	2.90345
3500–4000	1.53732	1.60446	2.21500	2.35390	2.12424
4500–5000	1.35910	1.60634	2.09154	1.88261	1.69240

are summarized below. First, a two-input, two-output, linear, time-invariant system

$$\begin{aligned} \dot{x}_1(t) &= x_2(t) + u_1(t) \\ \dot{x}_2(t) &= -5x_1(t) - 3x_2(t) + u_2(t) \qquad (48) \\ y_i(t) &= 22x_i(t) + 10, \quad i = 1,2 \end{aligned}$$

is simulated with the sampling interval $T_s = 0.01$ sec. The inputs $u_1(t)$ and $u_2(t)$ are set to be sinusoidal waves of frequency 10 Hz and magnitude 10.0. Let $Y(k) \equiv [y_1(k), y_2(k)]^T$, $U(k) \equiv [u_1(k), u_2(k)]^T$, and $\hat{Y}(k + d/k) \equiv [\hat{y}_1(k + d/k), \hat{y}_2(k + d/k)]^T$, an NN-based predictor is designed with ten input nodes, five hidden nodes, and five output nodes. The inputs of the network are

$$U(k), \quad U(k-1),\ U(k-2),\ U(k-3),\ U(k-4)$$

$$Y(k-1),\ Y(k-2),\ Y(k-3),\ Y(k-4),\ Y(k-5)$$

The outputs are the d-step ahead predictions $\hat{Y}(k + d/k)$, $d = 1, \ldots, 5$. The network is *trained* with the actual system outputs $Y(k + d)$. The training period is 4000 sample intervals, and the relative RMS (root-mean-square) prediction errors are tabulated in Table 1. The network is shown to be "well trained" after 4000 sample intervals.

Using the same structure of the NN-based predictor as above, the following nonlinear, time-invariant system is tested

$$\begin{aligned} \dot{x}_1(t) &= x_2(t) + u_1(t) \\ \dot{x}_2(t) &= -5(x_1^2(t) + x_1(t)) - 3x_2(t) + u_2(t) \qquad (49) \\ y_i(t) &= 22x_i(t) + 10, \quad i = 1,2 \end{aligned}$$

The inputs $u_1(t)$ and $u_2(t)$ are set to be sinusoidal waves of frequency 10 Hz and magnitude 10.0. The sampling interval is $T_s = 0.01$ sec. The network is *trained* with the actual system outputs $Y(k + d)$. The training period is 4000 sample intervals. The actual values of $y_1(k)$, $y_2(k)$ and their 2-step ahead prediction errors are plotted in Figures 4 and 5, which have shown that the NN-based predictor works well for the nonlinear, time-invariant system.

Both Equations (48) and (49) are time-invariant for which the "training-operation" mode works well. However, this is not the case for time-varying systems. To test such a system, we simulated a two-link robotic manipulator, whose dynamic equation is given as

$$\mathbf{H}(q)\ddot{q} + \mathbf{C}(q,\dot{q})\dot{q} + \mathbf{G}(q) = \tau$$

where $q \equiv [q_1, q_2]^T$ is the vector of joint positions, $\tau \equiv [\tau_1, \tau_2]^T$ joint torques, $\mathbf{H}(q)$ is the inertial matrix, $\mathbf{C}(q,\dot{q})\dot{q}$ represents the centrifugal and Coriolis forces, and $\mathbf{G}(q)$ is the gravitational force (Asada and Slotine, 1986). It is a nonlinear, two-input two-output, time-varying system. The joint torques τ are set to be sinusoidal waves of 10 Hz and magnitude 2.0 for τ_1 and 0.1 for τ_2. The sampling interval is chosen to be $T_s = 0.01$ sec. With an initial configuration as shown in Figure 6, and dynamic and kinematic parameters are presented in Table 2.

The joint positions vary due to gravitational force and joint torques. The NN-based predictor has ten input nodes, five hidden nodes, and five output nodes. The inputs of the network are

$$\tau(k), \quad \tau(k-1),\ \tau(k-2),\ \tau(k-3),\ \tau(k-4)$$

$$q(k-1),\ q(k-2),\ q(k-3),\ q(k-4),\ q(k-5)$$

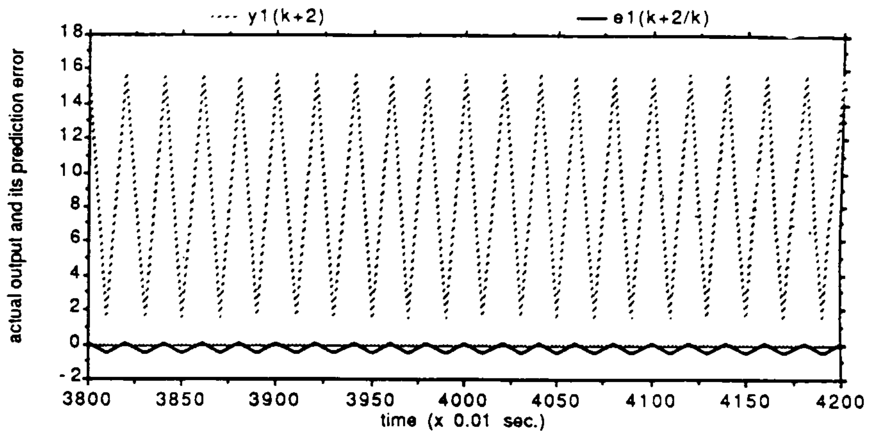

Figure 4. *$y_1(k)$ of the nonlinear, time-invariant system and its prediction error.*

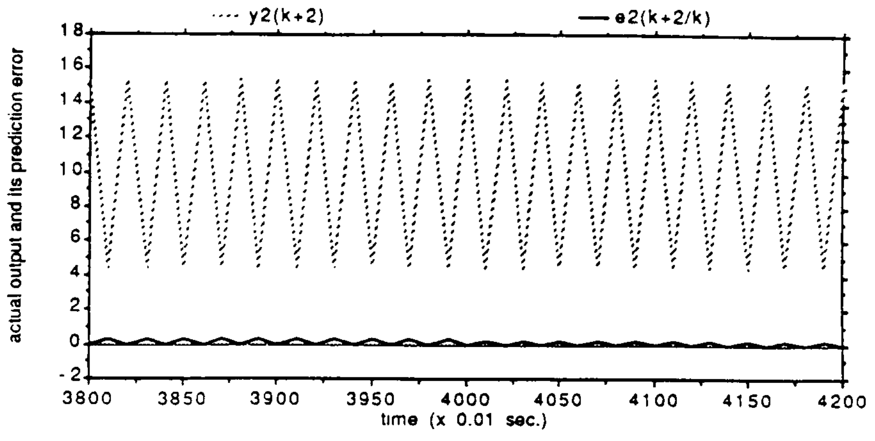

Figure 5. *$y_2(k)$ of the nonlinear, time-invariant system and its prediction error.*

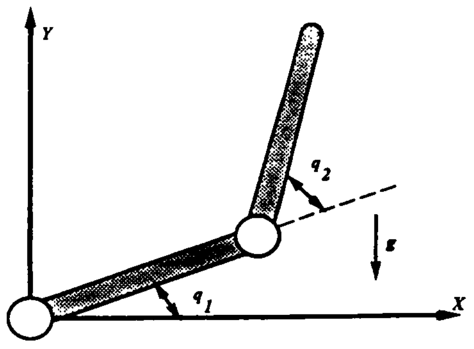

Figure 6. *A 2-link robot manipulator.*

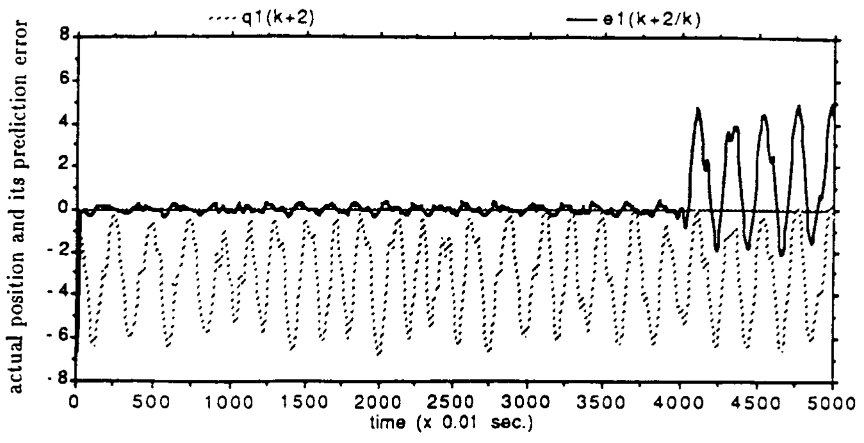

Figure 7. $q_1(k)$ *and its prediction errors with "training-operation" mode.*

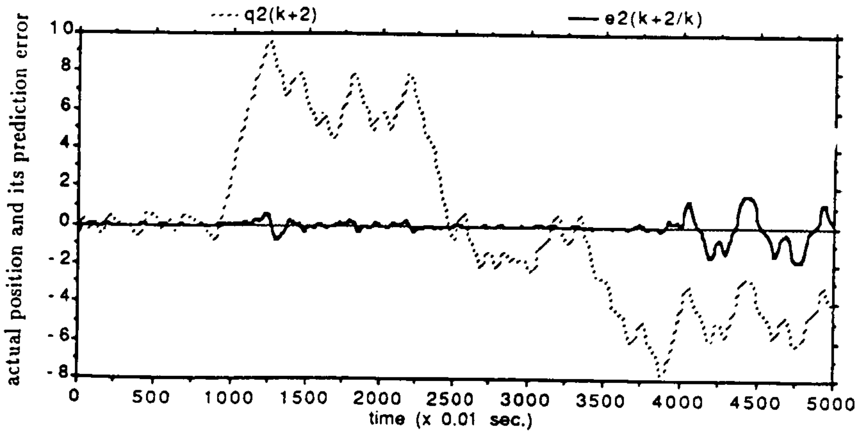

Figure 8. $q_2(k)$ *and its prediction errors with "training-operation" mode.*

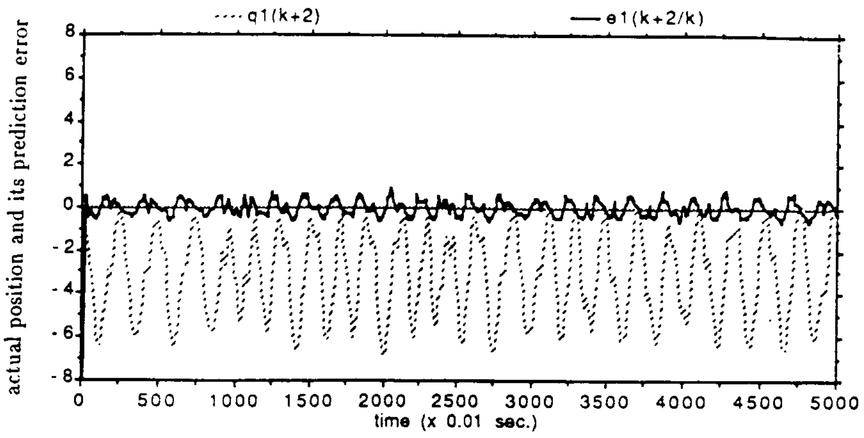

Figure 9. $q_1(k)$ *and its prediction errors with "updating" mode.*

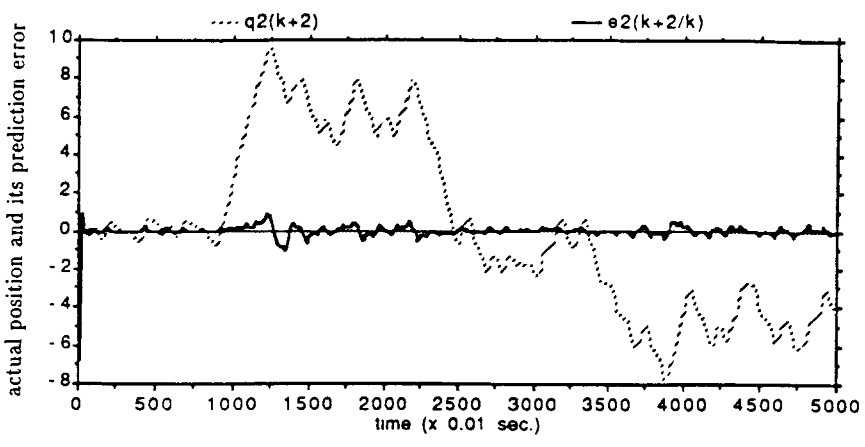

Figure 10. *q₂(k) and its prediction errors with "updating" mode.*

The outputs are the d-step ahead prediction of the joint position $\hat{q}(k + d/k)$, $d = 1, \ldots, 5$. The simulation results are plotted in Figures 7 and 8 for $q_1(k)$, $q_2(k)$, and their 2-step ahead prediction errors, respectively. The training period is 4000 sampling intervals. As soon as the training is stopped after this period, the prediction error increased dramatically, indicating that the "training-operation" mode does not work for time-varying systems.

So, we keep track of the variation of this time-varying system by using the updating mode described in the third section. The results are plotted in Figures 9 and 10. When compared with Figures 7 and 8, only the predictor with the updating mode is shown to work well. For this NN-based predictor, we evaluated the convergent process of the system-output prediction and the convergent process of the NN's weights. When the NN's weights are no longer changed, the NN becomes "well-trained". The simulation results have indicated that the system-output prediction converges to its true value within 200 sampling intervals. However, the NN becomes "well-trained" only after 3000–4000 sampling intervals. That is, the system-output prediction converges much faster than the well-training of an NN. This again supports the idea that the NN-based predictor should be updated, but not trained. Certainly, for a time-varying system, it is meaningless to say that an NN is "well-trained".

CONCLUSION

The output prediction of a system can be represented as a mapping from its historical data and future inputs to future outputs. Even if the parameters and/or structure of the system dynamics were unknown, an NN can be designed to approximate this mapping. Using these facts, an MIMO NN-based predictor is proposed and tested for various systems. The basic structure of the predictor is determined, and the following two major problems are solved. First, in order to track a time-varying mapping, the concept that an NN should be updated, rather than trained, is introduced and verified. By this concept and its corresponding algorithm, the proposed predictor uses only the system's historical data to adjust the weights of the NN. This also makes the network always work in a closed loop so that the reliability of the NN-based predictor is improved. Second, the BP algorithm is extended to a vector form so that an NN can be used to represent an MIMO mapping more efficiently and express some known coupling relations within the mapping more easily. This requires the nodes of the network to be capable of vector operations. Furthermore, the prediction error is analyzed and is shown to depend only on the network's error in approximating the actual mapping.

The proposed NN-based predictor has been tested for MIMO linear, nonlinear, time-invariant, and time-varying systems. All of them have shown promising results, indicating the potential use of the proposed predictor for many industrial applications.

Table 2. Kinematic and dynamic parameters of the simulated robot.

	Link 1	Link 2
Length	1 m	1 m
Mass center	0.5 m	0.5 m
Mass	20 kg	10 kg
Moment of inertia	0.8 kg m s^2	0.2 kg m s^2

REFERENCES

Asada, H. and J.-J. E. Slotine. 1986. *Robot Analysis and Control.* John Wiley and Sons, Inc.

Barron, A. R. 1989. "Statistical Properties of Artificial Neural Networks", *Proc. of the 28th Conference on Decision and Control, Vol. 1,* pp. 280–285.

Bhat, N. and T. J. McAvoy. 1989. "Use of Neural Nets for Dynamic Modeling and Control of Chemical Process Systems", *Proc. of 1989 American Control Conference,* pp. 1342–1347.

Cybenko, G. 1989. "Approximation by Superpositions of a Sigmoidal Function", *Mathematics of Control, Signals and Systems,* 2(4):303–314.

De Keyser, R. M. C. and A. R. Van Cauvenberghe. 1981. "A Self-Tuning Multi-Step Predictor Application", *Automatica,* 17(1):167–174.

]

Kung, S. Y. and J. N. Hwang. 1989. "Neural Network Architectures for Robotic Applications", *IEEE Trans. on Robotics and Automation*, 5(5):641-657.

Lippmann, R. P. 1987. "An Introduction to Computing with Neural Networks", *IEEE ASSP Magazine*, pp. 4-22.

Rumelhart, D. E. and J. L. McCelland. 1986. "Learning Internal Representations by Error Propagation", *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. MIT Press.

Weigend, A. S., B. A. Huberman and D. E. Rumelhart. 1990. "Predicting the Future: A Connectionist Approach", *Int'l. Journal of Neural Systems*, 1(3):193-209.

Werbos, P. J. 1988. "Backpropagation: Past and Future", *Proc. of Int. Conf. on Neural Networks*, 1:I343-I353.