A New Approach to Real-time Training of Dynamic Neural Networks

Fahmida N Chowdhury

University of Louisiana at Lafayette, PO Box 43890, Lafayette, LA 70504-3890, USA

Email: fnchowdh@louisiana.edu, Tel. 337-482-5366, Fax. 337-482-6687

Acknowledgement: This work was supported by National Science Foundation grant ECS-9753084 and LaSpace Consortium grant R199533.

Summary

A fast, efficient, and new way of real-time training of dynamic neural networks is presented in this paper. The proposed method would be suitable for training neural networks that need to be used for real-time system identification, fault detection, and control of uncertain dynamic systems. The training method proposed in this paper is unique in the sense that only the outer layer weights are updated; the hidden layer weights are chosen randomly at the beginning of the process, and left unchanged. The outer layer weights are estimated with an ordinary Kalman filter. This provides a learning rate which is an optimally time-varying vector. Results of computer simulation experiments, including a comparison with conventional backpropagation, are presented.

1 Introduction

Over the last decade, it has been well-established that neural networks can successfully model nonlinear dynamic systems that are usually represented by nonlinear difference equations. In practical applications, some of the most successful system identification and modeling techniques are based on discrete-time recursive input-output (i/o) models. These are usually NARMA[1]-type models given by:

$$y_k = \Psi(y_{k-1}, \ldots, y_{k-n}, u_{k-1}, \ldots, u_{k-n}). \tag{1}$$

In this paper, we consider nonlinear systems described by Eq. 1, where $u \in \mathcal{U} \subset I\!R$ is the scalar input variable, $y \in \mathcal{Y} \subset I\!R$ is the scalar output variable and Ψ is a real analytic function defined on $\mathcal{Y}^n \times \mathcal{U}^n$. Due to their good function approximation abilities, feedforward neural networks with $2n$ inputs and one output are used to model function $\Psi(\cdot)$. These neural networks have input vectors that are composed of past inputs and outputs of the dynamic system that needs to be identified. In this paper, we categorize these neural networks as "dynamic neural networks". In general, these networks have one hidden layer with sigmoidal nonlinearities, while the output neuron is linear. This paper[2] presents a new technique for real-time (on-line) training of those dynamic neural networks, where the input-output data are used one-by-one as they become available.

If all the training data are available at once, then regular backpropagation or any other off-line training method can be used. Many researchers, notably Narendra, Werbos, McAvoy [1, 2] have shown results and applications of such off-line training: for a good overview see the book [3] and the references cited there. The point to note, however, is that all these researchers assume the availability of the complete training data set in the beginning of the training process, and the training requires multiple epochs of presenting the same data to the NN, until the desired accuracy of modeling is achieved. In contrast, this paper is focused on the situation when:

[1] Nonlinear AutoRegressive Moving Average

[2] A preliminary version of this work was presented at the *IEEE Conference on Control Applications*, 2001.

1. The training data (input-output pair) are available one sample at a time, and the model must be built in real time with current and past data only.

2. Each training data vector can be presented exactly once to the NN (no repeats).

3. As new data become available, effects of old data can be forgotten (adaptation).

The motivation for selecting the above scenario is that in many engineering applications, on-line real-time modeling is required in order to improve the quality of performance and reliability of closed-loop systems. Although backpropagation (BP) has long been the most practical and widespread method of neural network training, on-line versions of BP have not been perfected yet; some research groups are currently using various ad-hoc methods [4] with varying degrees of success, and there is a lot of room for improvement. A few researchers have implemented the Extended Kalman Filter (EKF) for real-time training of neural networks [5, 6, 7], but the implementation has been for static systems such as the XOR problem and the parity problem etc. [5]. In [6], the entire derivation is for scalar (algebraic) maps of scalar variables. In contrast, our work is devoted to the case when the system to be modeled is a causal dynamic system, described by nonlinear difference equations w.r.t. time.

The technique of EKF assumes that [5, 7] the full set of NN weights must be estimated. In [7], after all the weights are estimated by the EKF, the effects of pruning some weights are investigated. Using EKF for updating the weights of a muli-layered NN can result in faster convergence in the sense that fewer training iterations are needed; however, there is a substantial increase in computational complexity at each iteration [8]. In this paper, we achieve a significant computational simplification by choosing the hidden-layer weights at random and keeping them unchanged; this gives a high probability that the hidden-layer weights will form a basis for the approximation, which can be tested by a simple rank condition. In such a case, only the outer-layer weights need to be computed. Since the outer layer is linear, the Kalman filter used here is a simple (not Extended) one, resulting in a much simpler and faster algorithm than other algorithms currently available in the literature. This simplification, and the shift from EKF to ordinary KF, is the main contribution of this

paper.

2 Main Results

Estimation of the system given by Eq. 1 can be performed by the following neural network:

$$\hat{y}_k = \sum_{i=1}^{n} c_i \phi_i(w_{i1} y_{k-1} + \ldots + w_{in} y_{k-n} + w_{i,n+1} u_{k-1} + \ ldots + w_{i,2n} u_{k-n}) = C^T \phi[W X_{k-1}] \quad (2)$$

where X_{k-1} is the NN input vector composed of past outputs $y_{k-1}, \ldots, y_{k-n}$ and past inputs $u_{k-1}, \ldots, u_{k-n}$ of the dynamic system. That is, $X_{k-1} = [y_{k-1}, \ldots, y_{k-n}, u_{k-1}, \ldots, u_{k-n}]^T$, $\phi_i(\cdot)$ is a saturation-type smooth nonlinear function, and$C$ and W are synaptic weight matrices of appropriate dimensions. Now we can write:

$$y_k = C^T \phi[W X_{k-1}] + \epsilon_k, \quad (3)$$

where ϵ_k is the function reconstruction error: $\epsilon_k = \Psi(X_{k-1}) - \hat{y}_k$. Training the NN implies estimating the matrices W and C, knowing the input-output data of the dyanmic system and assuming suitable functions $\phi_i(\cdot)$.

The basic justification for developing the algorithm presented in this paper is that the solution to the neural network training problem is not unique. Consider the case of noise-free measurement data, and assume that we wish to find the weights W and C so that the neural network is an exact i/o model of the unknown plant, that is, $y_k = C^T \phi[W X_{k-1}]$. Suppose the order of the plant equation is n, so the vector X has length $2n$; let the number of hidden units be m, and the available number of i/o pairs $\{X_i, y_i\}$ be p. For a fixed hidden-layer weight matrix $W_{(m,2n)}$, let the vectors

$$q_k = \phi[W X_{k-1}] \quad (4)$$

be collected in a matrix $Q_{(m,p)}$ (called the "Hidden Output Matrix"), and the outputs y_k be collected in a vector $Y_{(p,1)}$, while all the input vectors are collected in one matrix $X_{(2n,p)}$. Now we can write:

$$Q^T C = Y, \quad (5)$$

where $C_{(m,1)}$ is the weight vector of the output neuron, which is a linear combiner. The condition for existence of a unique and exact solution to this problem is that the rank of Q be p. The necessary condition for that is $m = p$, that is, we need as many hidden units as there are data points. Under these conditions, any weight matrx W that generates a hidden output matrix Q such that $\text{Rank}(Q) = p$, has a unique "companion weight" vector C that can be computed as

$$C = [Q^T]^{-1}Y. \tag{6}$$

Now let us consider the more realistic case that the plant is uncertain, the order is not known exactly, and the i/o measurements may be corrupted by noise, and therefore the number of data points (i/o pairs) obtained should be much greater than the number of hidden units and the order of the system. Then, the least-squares solution for the vector C, assuming that $\text{Rank}(Q) = m$, is given by:

$$C = [QQ^T]^{-1}QY. \tag{7}$$

In order to satisfy the rank condition of Q, we need $m \leq p$. It can be easily shown that with enough hidden units, specifically, $m \geq p$, it is always possible to generate weights W and C such that even noisy data points are a perfect fit. However, since it is not desirable to *exactly* fit noisy data points with our neural network, we do not elaborate further on that issue. Instead, assuming that we have an i/o data record $(X_{2n,p}, Y_{p,1})$ where $X = [X_{k-1}, \ldots, X_{k-p}]$ with $2n < p$ and an unknown plant $\Psi(X) = Y$, we state the following.

Theorem 1: For every matrix $W_{m,2n}$, every i/o data record $(X_{2n,p}, Y_{p,1})$ with $p > m$ from an unknown plant $\Psi(X) = Y$, and all permissible nonlinearities $\phi(\cdot)$ that satisfy the rank condition that $\text{Rank}[Q_{m,p}] = \text{Rank}[\phi(WX)] = m$, there exists a companion vector C such that the error $\epsilon = \Psi(X) - C^T\phi(WX)$ is minimized in a least-squares sense.

Proof: By direct substitution.

Remark 1: It is possible to model the unknown plant $\Psi(X) = Y$ with error $\epsilon = 0$ provided

(1) the number of hidden neurons is incrased to equal the length of the data record, that is, $m = p$, and (2) the rank condition on the hidden output matrix is satisfied, that is, $\text{Rank}[\phi(WX)] = m = p$.

This follows from the fact that the neural network training problem, when the output neuron is linear, can be written as

$$\phi(WX)^T C = Q^T C = Y,$$

and an exact solution exists when Q is invertible. Then, the outer-layer weight is given by $C = [Q^T]^{-1} Y$.

Here, the error ϵ refers to the error in reconstructing the function at the training data points *only*. This, however, does not guarantee zero error at other data points. This drawback is not peculiar to our method; it exists in *all* curve-fitting problems when the underlying function generating the data is unknown.

Corollary 1: The squared norm of the error ϵ is then given by:

$$\epsilon^T \epsilon = Y^T (I - Q^T [QQ^T]^{-1} Q) Y. \tag{8}$$

Remark 2: It cannot be guaranteed that the error ϵ would be the minimum over all possible choices of (W, C) for a given fixed neural network architecture (m = constant); however, for many practical purposes, this error may be well within the acceptable range of model-mismatch for the specific application at hand.

Remark 3: The search for a global minimum of the function reconstruction error ϵ is not a reasonable goal in neural network training; this is illustrated by the following theorem.

Theorem 2: Let neural network $(\tilde{C}_{m,1}, \tilde{W}_{m,2n})$ optimally model dynamic system $Y = \Psi(X)$ for a given data record (X, Y) in the sense that the function reconstruction error $\tilde{\epsilon}$ is the minimum for the chosen network architecture; that is, for fixed values of n and m, no lower error

is possible. Then, there exist many other pairs of combinations $(C_{m,1}, W_{m,2n})$ that generate exactly the same error $\tilde{\epsilon}$. These other pairs (C, W) can be obtained by linear transformations T such that

$$\phi(WX) = T\phi(\check{W}X), \tag{9}$$

and

$$C = [T^{-1}]^T \tilde{C}. \tag{10}$$

Proof: Let $\tilde{Q} = \phi(\check{W}X)$ and $Q = \phi(WX)$. Then $\tilde{Q}^T\tilde{C} = Y - \epsilon$. Choose any invertible matrix $T_{m,m}$ and compute C according to the theorem. For all appropriate matrices W satisfying Eq. 9, we then obtain

$$Q^T C = [T\phi(\check{W}X)]^T[T^{-1}]^T\tilde{C} = \tilde{Q}^T\tilde{C} = Y - \tilde{\epsilon}, \tag{11}$$

that is, the function reconstruction error remains unchanged.

In Theorem 2, conditions for an exact match of the optimal error vector are stated: however, for all practical purposes, it is sufficient that neural networks (C, W) generate error ϵ such that

$$\epsilon^T\epsilon = \tilde{\epsilon}^T\tilde{\epsilon},$$

that is, the two errors are similar in the sense of the norm but they are not required to match exactly.

Corollary 2: All matrices W (such that $Q = \phi(WX)$) that satisfy, for the given data record (X, Y), the condition

$$Y^T[I - Q^T(QQ^T)^{-1}Q]Y = \tilde{\epsilon}^T\tilde{\epsilon}$$

and the corresponding $C = [QQ^T]^{-1}QY$, constitute neural networks that are also optimal models of the unknown plant $Y = \Psi(X)$.

The main result is that since the weight pair (W, C) is not unique, any algorithm that

finds any feasible solution with an acceptable level of the function reconstruction error ϵ, should be sufficient for all practical purposes. Based on this, we develop the fast on-line algorithm described below.

3 Proposed Real-time Training

We have established that the solution to the NN training problem is not unique, and we are not searching for a solution with globally minimum error. In real-time training, our goal is to find any feasible solution (that is, set of NN weights) that provides the minimum possible error between the plant output and the NN output at the current time-step, using data only up to the current time-step. The main concept of the proposed training method consists of the following steps:

- Generate the hidden-layer weight matrix W at random.
- For this specific W matrix, find the best possible output-layer weight vector C, given all the i/o data available up to the current time step, and updating the estimate with each newly available data point.
- Compute the function reconstruction error ϵ, and if $\epsilon > \delta$ (acceptable or desired threshold), choose one of the following:
    1. Update W by backpropagation and repeat the steps to re-estimate C, or
    2. Increase the number of hidden units m, adjust W by adding extra (randomly generated) rows to it, and re-estimate C.

The above process can be implemented by a combination of a Kalman-filter-like recursive optimization technique, and an on-line version of backpropagation. In all the simulation experiments we carried out, the Kalman-like optimization was sufficient to achieve a very low error, so that backpropagation was never required.

3.1 Estimating the Outer Layer Weights in Real Time

In this paper, the problem of fast, real-time optimization of the outer layer weights is solved by using recursive least-squares – casting the problem into a Kalman-filter-like format. First, define the vector

$$H_k = \phi(WX_{k-1}),$$

where

$$X_{k-1} = [y_{k-1}, \ldots, y_{k-n}, u_{k-1}, \ldots, u_{k-n}]^T,$$

W is the randomly generated weight matrix for the hidden layer, and ϕ is the sigmoidal activation function. Using the equation

$$y_k = H_k^T C_k + \epsilon$$

as the observation equation of a *fictitious* dynamic system, and assuming a *fictitious* state equation:

$$C_{k+1} = C_k + w_k, \tag{12}$$

C_k is estimated recursively; denote the estimates by $\hat{c}_k$. The estimation process is started with assumed values of a few terms, as in the typical Kalman filter. The covariance of the overall modeling error (ϵ) is defined as:

$$S_k = E[\epsilon_k \epsilon_k^T],$$

and is a design parameter – it should be kept small. The covariance of the outer-layer weight estimation error is defined as:

$$P_k = E[C_k - \hat{C}_k][C_k - \hat{C}_k]^T,$$

and is assumed to be a large matrix in the beginning of the process, to indicate poor apriori knowledge. This covariance P_k is updated at each time step according to the equation

$$P_k = [I - K_k H_k^T] P_{k-1} + Q_k.$$

The initial value of the state vector estimate $\hat{C}_k$ is generated randomly at $k = 1$, and the predicted plant output is computed recursively at each subsequent time-step:

$$\hat{y}_{k/k-1} = H_k^T \hat{C}_{k-1}.$$

Knowing the measured actual output y_k, the residual of the estimation process is computed:

$$R_k = y_k - \hat{y}_{k/k-1}.$$

The Kalman gain is computed as:

$$K_k = P_{k-1} H_k [S_k + H_k^T P_{k-1} H_k]^{-1}. \tag{13}$$

Remark 4: The Kalman gain (Eq. 13) plays the role of the "learning rate" in this proposed method of neural network training. Note that K_k is time varying, and has as many components as the number of hidden neurons.

The updated estimate of the state vector (in this implementation of the Kalman filter, "state vector" = "outer-layer weights") at step k is:

$$\hat{C}_k = \hat{C}_{k-1} + K_k R_k. \tag{14}$$

The algorithm presented in this section is similar to the standard Kalman filter, but the interpretation and construction of most major terms are different:

1. The observation matrix H_k is time-varying, and contains the outputs of nonlinear functions $\phi(X)$, where X contains past inputs and outputs of the dynamic system that is being identified.

2. The "state vector" C contains the weights of the output neuron, which is a linear combiner.

3. The "system matrix" used in the algorithm (see Eq. 12) is the Identity matrix.

4. The state vector is assumed to be an unknown constant, with a small random walk component to allow for adaptive estimation. artificial process noise.

5. The covariance Q_k of the "process noise" of this *fictitious linear system* is chosen to be αI, where α is a small positive number. In fact, if the plant is not expected to change at all during the time of interest, the process noise covariance can be dropped to zero.

6. The Kalman filter was originally designed for linear time-invariant systems; but here, we use it for parameter estimation of a nonlinear system which is a neural network. Note that this is not an Extended Kalman filter: there is no joint estimation of states and parameters - here, the parameter vector (that is, unknown weights of the outer layer) is directly defined as the state vector in the Kalman filter.

Remark 5: The method described above is very powerful because the learning rate (that is, the Kalman gain) is time-varying and optimal in a probilistic sense.

After a few time steps of the process described above, the state (weight) vector converges to a near-constant (provided the original plant remains time-invariant), implying that there exists a set of constant weights C such that the function reconstruction error ϵ can be minimized. At this time, we check the desired tolerance, and if $\epsilon_k > \delta$, we apply backpropagation to update the hidden-layer weights W, or increase the number of hidden units and repeat the same (Kalman filter) procedure. The backpropagation equations are not described here, since they are well known in the literature.

4 Computer Simulation Experiments

In this section, we present results of implementing the proposed training method to two nonlinear plants: one taken from the textbook [3] in order to compare with conventional BP training, and the other, an industrial process, taken from a previously published paper [9].

4.1 Comparison with Conventional BP Training

Consider a scalar nonlinear plant given by the following discrete-time state equation [3]:

$$x_{k+1} = \frac{x_k x_{k-1} x_{k-2} u_{k-1} (x_{k-2} - 1) + u_k}{1 + x_{k-2}^2 + x_{k-1}^2}. \tag{15}$$

A single-layer NN with 20 bipolar sigmoidal hidden activation units was used for this plant (see [3] for further details). It was trained using incremental BP, with random inputs uniformly distributed in the interval [-1, +1]. With 500 training samples, it took $10,000$ iterations to achieve what is seen in Figure 1, which shows the result of testing with the input u_k: for $k = 1, 2, ..., 500$, $u_k = sin(2\pi k/250)$, and for $k > 500, u_k = 0.8sin(2\pi k/250) + 0.2sin(2\pi k/25)$. Clearly, the test result in Figure 1 shows that the NN is incapable of tracking the plant output at the negative peaks. For comparison, see Figure 2 which shows the performance of the training method proposed in this paper. We used only 6 hidden neurons (as opposed to 20 in [3]), with random (Gaussian zero mean) weights selected once and kept fixed; the outer layer weights are also chosen randomly, and updated with the Kalman filter algorithm at each time-step, with a 3-rd order NARMA model. The sigmoidal nonlinearity in the hidden layer is the "logsig" function, $\phi(\alpha) = [1 + exp(-\alpha)^{-1}]$, and the outer layer consists of one linear neuron. In our method, the weight adaptation is kept up at all time-steps – at each step only past and current data are used, with no repeats. Notice that in Figure 2 the plant and NN outputs practically overlap, except in the very beginning of the process, when the estimation is just starting with all random guesses. In Figure 3 the evolution of the weight vector of the output neuron is shown. Figure 4 shows the error between the plant and the NN outputs, and Figure 5 shows the evolution of the learning rate vector. This time-varying, optimal learning rate (the Kalman gain, which is obtained by solving the matrix Riccati equation) is the main instrument for the impressive performance of the training method. In conventional BP-type NN training, the learning rate typically is a small constant, chosen arbitrarily.

4.2 Experiment with Industrial Plant: Adaptive Training

The industrial plant used here is described by:

$$y(l) = -0.00113 - 0.0628u(l-2) - 0.0675u(l-3) - 0.0215u(l-4) + 0.84y(l-1) - 0.0526u(l-3)y(l-2)$$

$$-0.053u(l-2)y(l-1) + 0.0613y(l-1)^2 - 0.0071u(l-2)u(l-3) - 0.0234(u(l-2)^2)y(l-1)$$

$$-0.044(u(l-3)^2)y(l-1) + 0.0573u(l-2)y(l-1)^2 - 0.02y(l-3)^2. \qquad (16)$$

This plant, taken from [9], represents the change in the frequency (y) of a generator as a function of the current (u), and was identified in an industrial process. We used a neural network with $m = 9$ hidden units, and the order of the plant was assumed $n = 3$, although in reality the plant is 4-th order - we assumed that this information was unknown. Initial guess for the outer-layer weights was random, and the plant input u was also random.

We ran experiments where the plant Eq. 16 was subjected to changes during the simulation time-period, and the outer-layer weight adaptation was kept running at all times. Moreover, we added random noise (Gaussian, zero mean, standard deviation of 2% of the output level) to the output of the plant. This simulation was run for a length of 1500 time-steps. In this specific experiment, at time step 200, the plant equation was changed from Eq. 16 to the following:

$$y(l) = -0.3 - 0.0628u(l-2) - 0.0675u(l-3) - 0.0215u(l-4) + 0.84y(l-1) - 0.0526u(l-3)y(l-2)$$

$$-0.053u(l-2)y(l-1) + 0.0613y(l-1)^2 - 0.0071u(l-2)u(l-3) - 0.0234(u(l-2)^2)y(l-1). \quad (17)$$

In addition to this change in the plant equation, we changed the input from a random sequence to a sinusoid at time step 500. The plots presented in this paper clearly show how the changes in the plant are reflected in the actual output and the NN-estimated output. The neural network is able to adapt its outer-layer weights to model this changing plant. Figure 6 shows the plant output and NN-output; notice that the two are very close most of the time. Figure 7 shows the modeling error $y_k - \hat{y}_k$. Figure 8 shows the evolution of the outer-layer weights; at steps 200 and 500, the plant changes – and the outer-layer weights clearly show these changes. When the change occurs in the plant, the error (ϵ) becomes higher than a pre-set threshold, and weight adaptation is resumed automatically. In our experiments, there was no need to increase the number of hidden neurons, or resort to backpropagation.

5 Conclusion

A novel method of fast real-time training of dynamic neural networks is presented in this paper. The method consists of randomly choosing the hidden-layer weights and using a simple Kalman filter to update the outer-layer weights, with the time-varying Kalman gain as the learning rate of the neural network. Simulation experiments demonstrate the implementation of the concept. From a dynamic systems point of view, such on-line training can be useful for all neural network applications requiring real-time updating of the connection weights.

References

[1] Narendra KS, Parthasarathy K. Identification and control of dynamic systems using neural networks. *IEEE Trans. Neural Networks* 1990; **1**(1):4-27.

[2] Werbos PJ, McAvoy T, Su T. Neural Networks, System Identification, and Control in the Chemical Process Industries. In *Handbook of Intelligent Control*, Ed. White and Sofge, Van Nostrand Reinhold: New York, 1992; 283-356.

[3] Hassoun MH. *Fundamentals of Artificial Neural Networks*, MIT Press 1995; 259-261.

[4] Hovakimyan N, Lee H, Calise AJ. On approximate NN realization of an unknown dynamic system from its input-output history. *Proc. American Control Conference* 2000; Paper N WM17-1, Chicago, US.

[5] Iiguni Y, Sakai H, Tokumaru H. A real-time learning algorithm for a multilayered neural network based on the extended Kalman filter. *IEEE Trans Signal Processing* 1992; **40**(4):959-966.

[6] Nicolao GD, Ferrari-Trecate G. Regularization networks: fast weight calculation via Kalman filtering. *IEEE Trans Neural Networks* 2001; **12**(2):228-235.

[7] Sum J, Leung C, Young GH, Kan W. On the Kalman filtering method in neural network training and pruning. *IEEE Trans Neural Networks* 1999; **10**(1):161-166.

[8] Zhang Y, Li XR, A fast U-D factorization-based learning method with applications to nonlinear system modeling and identification. *IEEE Trans Neural Networks* 1999; **10**(4):930-938. time-varying linear models. *IEEE Trans Automatic Control* 2000; **45**(7):1355 - 1358.

[9] Chowdhury FN, Kotta Ü, Nõmm S. On realizability of neural networks based input-output models. *Proc. Differential Equations and Applications* 2000; St. Petersburg, Russia.

Author Biography

Fahmida Chowdhury was educated at Moscow Power Engineering Institute, Moscow, former-USSR, and Louisiana State University, USA. She is currently an Associate Professor in the Electrical and Computer Engineering Department at the University of Louisiana at Lafayette, and holds the Chevron/BoRSF Regents Professorship in Engineering. Her research interests are in fault detection, adaptive control, and neural networks. She was awarded the Fulbright Scholarship in 2000-2001. Her research has been supported by the National Science Foundation, Louisiana Board of Regents, Louisiana Space Consortium, NASA, and private industry. She is a senior member of the IEEE, and has served on the IEEE Control Systems Society's Conference Editorial Board.

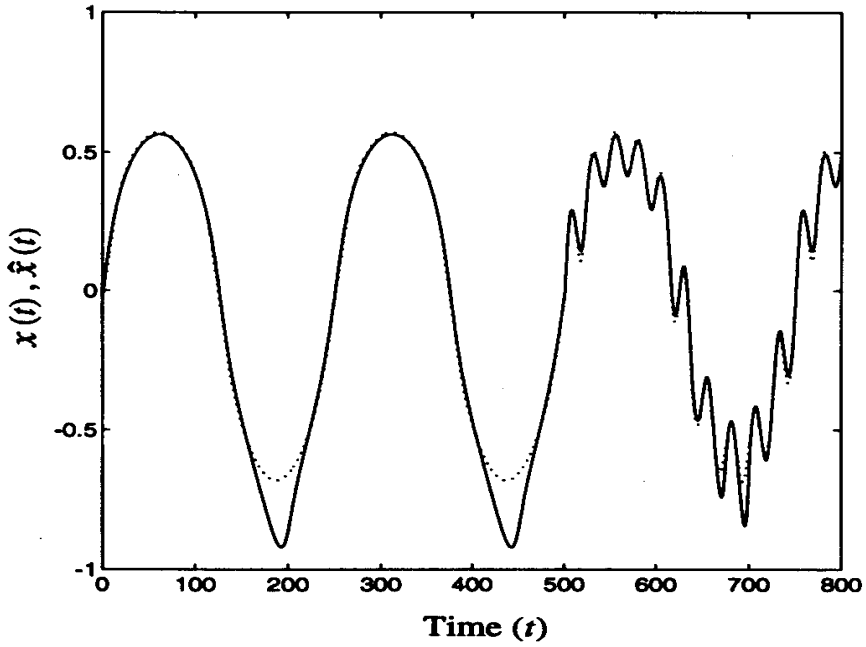

Figure 1: Outputs of the plant (solid line) and the NN (dotted line) – experiment 1, BP training

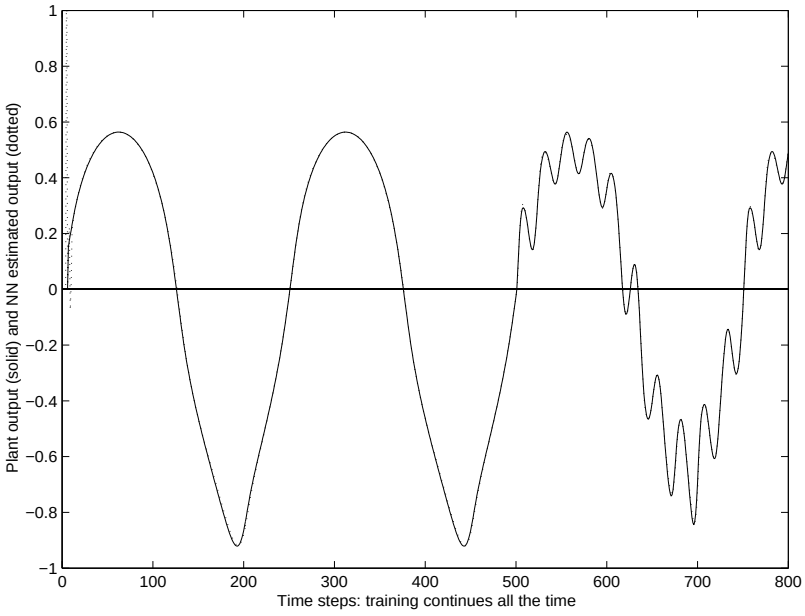

Figure 2: Outputs of the plant (solid line) and the NN (dotted line) – experiment 1, proposed KF training

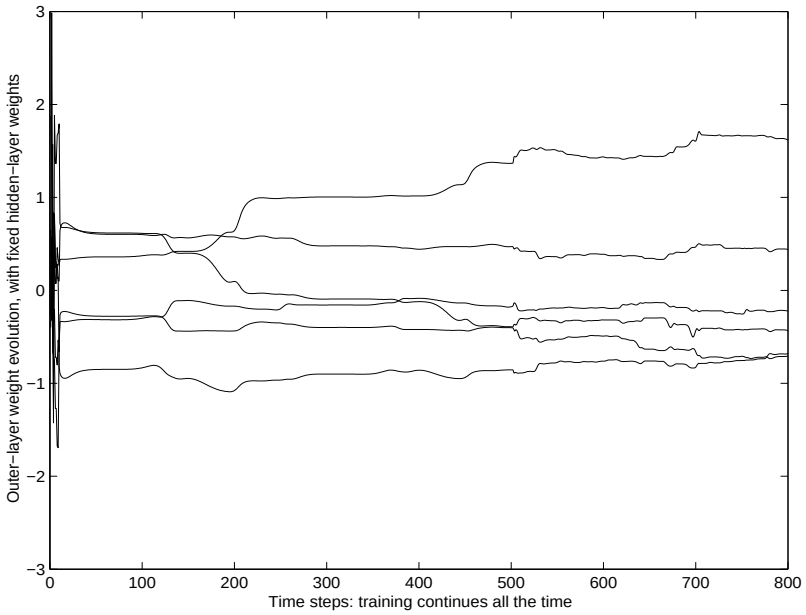

Figure 3: Evolution of the outer layer weights – experiment 1, proposed KF training

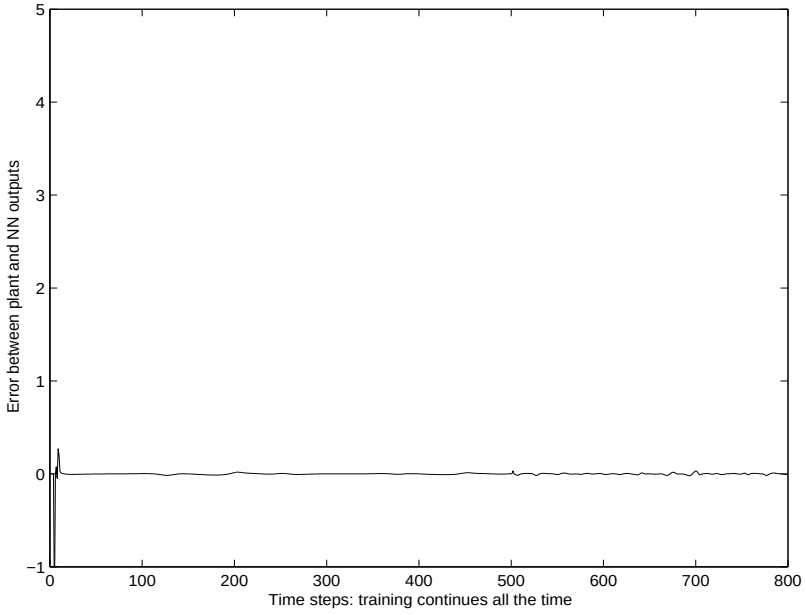

Figure 4: Error between plant output and NN output – experiment 1, proposed KF training

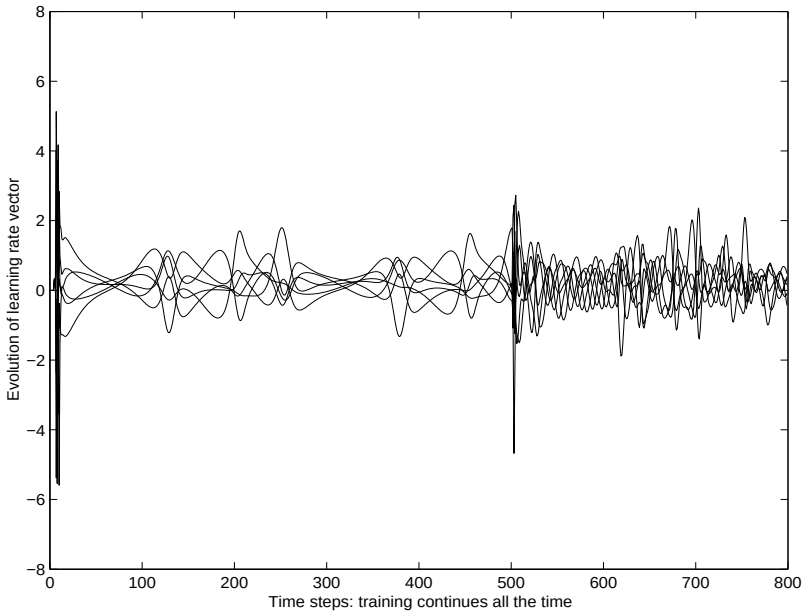

Figure 5: Evolution of the learning rate vector – experiment 1, proposed KF training

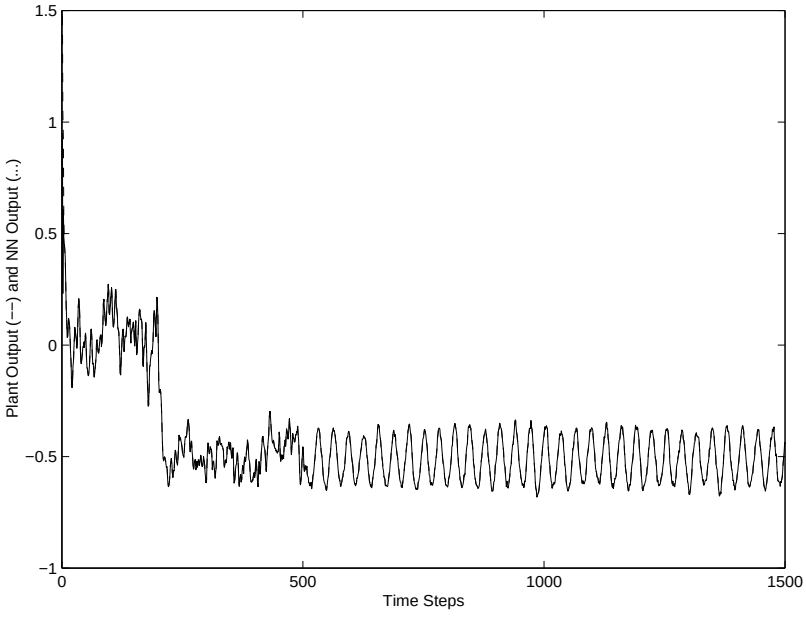

Figure 6: Outputs of the plant (solid line) and the NN (dotted line) – experiment 2

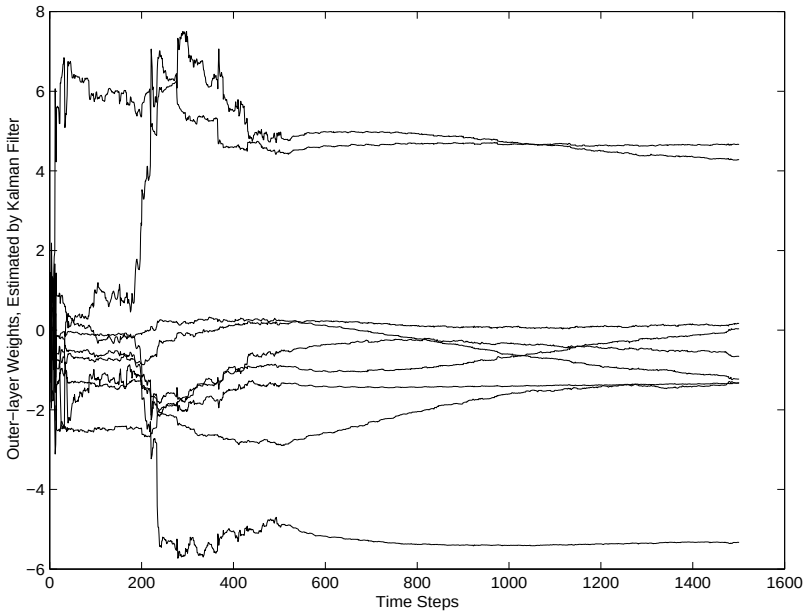

Figure 7: Evolution of the outer-layer weights of the NN – experiment 2

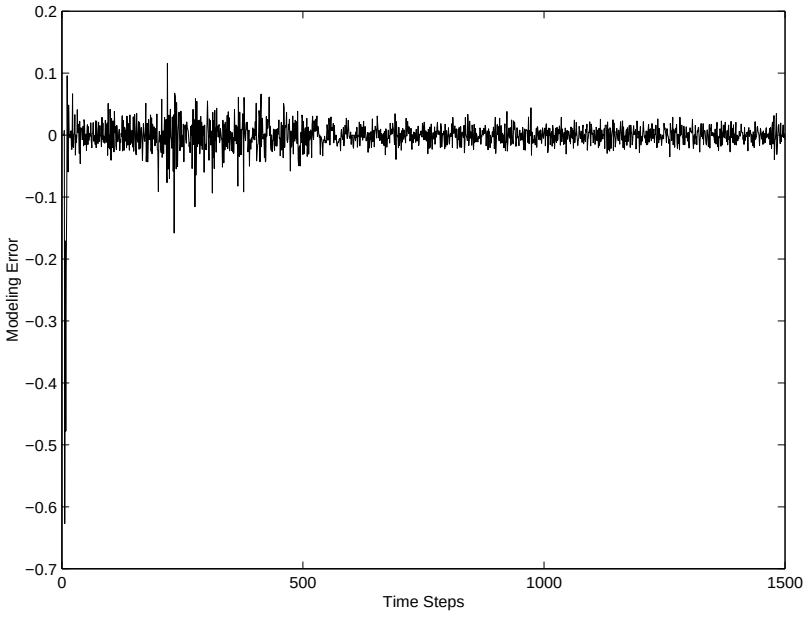

Figure 8: Error between the plant and NN outputs – experiment 2