Training Recurrent Neural Networks: Why and How ?
An Illustration in Dynamical Process Modeling.

O. NERRAND, P. ROUSSEL-RAGOT,
D. URBANI, L. PERSONNAZ, G. DREYFUS, Senior Member, IEEE
Ecole Supérieure de Physique et de Chimie Industrielles
de la Ville de Paris,
Laboratoire d'Electronique
10, rue Vauquelin
75005 PARIS, FRANCE

ABSTRACT

The paper first summarizes a general approach to the training of recurrent neural networks by gradient-based algorithms, which leads to the introduction of four families of training algorithms. Because of the variety of possibilities thus available to the "neural network designer", the choice of the appropriate algorithm to solve a given problem becomes critical. We show that, in the case of process modeling, this choice depends on how noise interferes with the process to be modeled; this is evidenced by three examples of modeling of dynamical processes, where the detrimental effect of inappropriate training algorithms on the prediction error made by the network is clearly demonstrated.

1 INTRODUCTION

During the past few years, there has been a growing interest in the training of recurrent neural networks, either for associative memory tasks, or for tasks related to grammatical inference, time series prediction, process modeling and process control. A general framework for the training of recurrent networks by gradient descent methods, which has been proposed recently [1, 2], is summarized in section 2; it encompasses algorithms which have been used classically in linear filtering, identification and control, and algorithms which have been established in the framework of neural network research; in addition, this general approach leads to original algorithms. The variety of algorithms thus available raises the question of the choice of an appropriate one in a given situation. In section 3, we show, in the framework of non-linear process identification (i.e., of the estimation of the parameters of a model of a non-linear process), that the choice of an appropriate algorithm depends of how *noise* interferes with the process. The striking effect of using either an appropriate algorithm or an inappropriate one for modeling a non-linear process undergoing non-measurable, random perturbations, is shown on examples.

2 A GENERAL FRAMEWORK FOR THE TRAINING OF RECURRENT NETWORKS BY GRADIENT-BASED DESCENT ALGORITHMS

In this section, we summarize a general approach described in more detail in [1]. We first define the terms which will be used in the paper. Some of this terminology is borrowed directly from the fields of filtering and automatic control; since many familiar concepts in the neural network area have been in use in other disciplines, we deem it unnecessary, and in most cases confusing, to coin new words for old concepts; conversely, we shall introduce a few new terms whenever required for clarity. In the second part of this section, we recall the ingredients of the algorithms whose use is illustrated in section 3.

2.1 Some definitions

Because the terminologies used in adaptive filtering, in automatic control, and in the literature on neural networks, are sometimes conflicting, we first define the terms that we use in the paper.

Adaptive vs. non-adaptive training

The training of a network makes use of two sequences, the sequence of inputs and the sequence of corresponding desired outputs. If the network is first trained (with a training sequence of finite length), and subsequently used (with the fixed weights obtained from training), we shall refer to this mode of operation as "non-adaptive". Conversely, we term "adaptive" the mode of operation whereby the network is trained permanently while it is used (with a training sequence of infinite length).

Performance criterion, cost function and training function

The computation of the coefficients during training aims at finding a system whose operation is optimal with respect to some performance criterion which may be either quantitative, e.g., maximizing the signal to noise ratio for spatial filtering, or qualitative, e.g. the (subjective) quality of speech reconstruction. In the following, we assume that we can define a positive *training function* which is such that a decrease of this function through modifications of the coefficients of the network leads to an improvement of the performance of the system.

In the case of non-adaptive training, the training function is defined as a function of all the data of the training set (in such a case, it is usually termed *cost function*); the minimum of the function corresponds to the optimal performance of the system. Training is an optimization procedure, using gradient-based methods.

In the case of adaptive training, it is impossible, in most instances, to define a time-independent cost function whose minimization leads to a system which is optimal with respect to the performance criterion. Therefore, the training function is time-dependent. The modification of the coefficients is computed continually from the gradient of the training function. The latter involves the data pertaining to a time window of finite length, which shifts in time (sliding window), and the coefficients are updated at each sampling time for instance.

Recursive vs. non-recursive algorithms, iterative vs. non-iterative algorithms

A *non-recursive* algorithm makes use of a cost function (i.e. a training function defined on a fixed window). A *recursive* algorithm makes use of a training function defined on a sliding window [3]. Therefore, an adaptive system must be trained by a recursive algorithm, whereas a non-adaptive system may be trained either by a non-recursive or by a recursive algorithm.

An *iterative* algorithm performs coefficient modifications *several times* from a set of data pertaining to a given time window; a *non-iterative* algorithm does this *only once*. The popular LMS (Least Mean Squares) algorithm is thus a recursive, non-iterative algorithm operating on a sliding window of length 1.

In the following, we focus on the computation of the coefficients by gradient-based descent; in the recursive, non-iterative case, the modification of the coefficients at time n can be written as $\Delta C(n) = \mu(n)\ D(n)$ where $\{\mu(n)\}$ is a sequence of positive real numbers and D(n) is a linear transformation of the gradient of the training function; in the simple gradient method, D(n) is just the opposite of the gradient and $\mu(n)$ is constant.

2.2 Training algorithms for recurrent networks

Canonical form

All the computational details on the material presented in this section can be found in reference [1].

It has been shown in [1] that any feedback network can be cast into a canonical form which consists of a feedforward (static) network

- whose outputs are the outputs of the neurons which have desired values, and the values of the state variables,
- whose inputs are the inputs of the network and the values of the state variables, the latter being delayed by one time unit (Figure 1a).

The canonical form is thus expressed as

$S(k) = \varphi_1[S(k\text{-}1), I(k\text{-}1)]$; $z(k\text{-}1) = \varphi_2[S(k\text{-}1), I(k\text{-}1)]$,

where S(k) is the state vector, whose dimension N_r is the order of the network, where z(k-1) is the output, and where I(k) is the vector of non-feedback inputs. The transformation of a non-canonical form to a canonical form is described in [1]. Note that this concept can be used with any type of discrete-time neuron, including for instance the high-order units used for grammatical inference [4]

Training function

The main difficulty in the *recursive* training of recurrent networks arises from the fact that the output of the network and its partial derivatives with respect to the coefficients depend on the values of the inputs since the beginning of the training process, and on the initial state of the network. Therefore, a rigorous computation of the gradient of the training function would imply taking into account all the past inputs, and related desired outputs. This is not practical for two reasons: first, it would require ever increasing computation times; second, in the case of the modeling or control of a non-stationary process, taking the whole past into account would not make sense, since a large part of the past might be irrelevant. Therefore, the estimation of the gradient of the training function is performed by truncating the computations to a fixed number of sampling periods N_t into the past. Thus, at time n, this estimation will involve N_t identical copies of the feedforward part of the canonical form of the network, with coefficients computed at time n-1 (Figure 1b).

The training function at time n is defined on a sliding window of length N_c as a sum of N_c quadratic errors:

$$J(C, n) = \frac{1}{2} \sum_{m= N_t-N_c+1}^{N_t} [e^m(n)]^2 \text{ with } e^m(n) = d(n\text{-}N_t\text{+}m) - y^m(n) \text{ and } 1 \le N_c \le N_t \ ,$$

where $y^m(n)$ is the output of copy m ($1 \le m \le N_t$). $y^m(n)$ is the value that the output of the network would have taken on, at time $n\text{-}N_t\text{+}m$, had the vector of coefficients at that time been equal to C(n-1).

In the case of *non-recursive* training, the training (or cost) function is defined on a fixed window of length N_c ; at iteration i:

$$J(C, i) = \frac{1}{2} \sum_{m= N_t-N_c+1}^{N_t} \left[e^m(i)\right]^2 \text{ with } e^m(i) = d(m) - y^m(i) \text{ and } 1 \le N_c \le N_t \; ,$$

where $y^m(i)$ is the output of copy m ($1 \le m \le N_t$), computed with the weights C(i-1) obtained at iteration i-1.

Algorithms

The computation of the above training function requires the computation of the outputs $y^m(n)$ (or $y^m(i)$), which in turns require the computation of the state $S_{in}{}^m(.)$ of the network (Figure 1b); various algorithms arise from different choices of the values of the state inputs. In [1], four families of algorithms were introduced: undirected, semi-directed, directed, and hybrid. In the following, we restrict our discussion to the case where the desired values of the state variables are available; thus, the first three families only will be considered in the present paper.

3 APPLICATION: NON-LINEAR PROCESS IDENTIFICATION BY NEURAL NETWORKS

3.1 The problem

Assume that a set of measurements can be carried out on a non-linear process. From this data, a predictor model must be derived, whose dynamical behaviour should be as close as possible to that of the process. The identification of the process is the estimation of the parameters of the predictor, based on the available data; if the predictor is implemented as a neural network, the identification is the training of the network. When identifying a non-linear, dynamical process, a recurrent network is a logical candidate. We show in the following how the choice of the appropriate training algorithm results from assumptions made on the role on random noise in the process. We use non-linear generalizations of three popular models corresponding to three different assumptions on the noise; we describe the predictor associated to each model, i.e. the predictor which is such that *the prediction error is the unpredictable part of the process output*. We show, in each case, which of the above algorithms is the most appropriate, if the predictor is implemented as a neural network.

3.2 Three black-box models

Three approaches with black-box models will be considered, depending on the assumptions made on the process [3]: (i) the *output error* model, (ii) the *NARMAX* model, and (iii) the *NARX* (or *equation error*) model .

In the *output error* approach, it is assumed that the output $y_p(k)$ of the process (Figure 2a) obeys the following equations:
$x(k) = \Phi\left[X(k\text{-}1), U(k\text{-}1)\right]$,
$y_p(k) = x(k) + w(k)$,
with $X(k\text{-}1) = \{x(k\text{-}1), x(k\text{-}2), ...x(k\text{-}N)\}$, and $U(k\text{-}1) = \{u(k\text{-}1), u(k\text{-}2), ...u(k\text{-}M\}$.
$\{w(k)\}$ is a white noise sequence.
The output $y(k)$ of the associated predictor (Figure 2b), such that $y_p(k)\text{-}y(k)=w(k)$, is given by:
$y(k) = \Phi\left[y(k\text{-}1), \ldots , y(k\text{-}N), U(k\text{-}1)\right]$.
Therefore, the associated predictor of the output error process is *recurrent* of order N. If there exists a neural network which can approximate function Φ, this network can implement the predictor, *and it must be trained by an undirected algorithm* [1], since it is essential that the predictor be recurrent.

A NARMAX (Non-linear Auto-Regressive Moving Average with eXogeneous inputs) model [5] (Figure 3a) obeys the following equation:
$x_p(k) = \Phi\left[X_p(k\text{-}1), U(k\text{-}1), W(k\text{-}1)\right] + w(k)$,
$y_p(k) = x_p(k)$.
where $X_p(k\text{-}1) = \{x_p(k\text{-}1), x_p(k\text{-}2), ..., x_p(k\text{-}N)\}$ and
$W(k\text{-}1) = \{w(k\text{-}1), w(k\text{-}2), ..., w(k\text{-}P)\}$.
The output $y(k)$ of the associated predictor (Figure 3b) is defined by:
$y(k) = \Phi\left[Y_p(k\text{-}1), U(k\text{-}1), e(k\text{-}1), ..., e(k\text{-}P)\right]$ where $e(k) = y_p(k)\text{-}y(k)$
and $Y_p(k\text{-}1) = \{y_p(k\text{-}1), y_p(k\text{-}2), ..., y_p(k\text{-}N)\}$.
Therefore, the predictor of the NARMAX process is *recurrent* of order P, and, if it is implemented as a neural network, *it must be trained by an undirected algorithm* [1].

In the *equation error* approach (Non-linear Auto-Regressive with eXogeneous inputs, or NARX, model, Figure 4a), it is assumed that the process obeys the following equations:
$x_p(k) = \Phi\left[X_p(k\text{-}1), U(k\text{-}1)\right] + w(k)$,
$y_p(k) = x_p(k)$.
The output $y(k)$ of the associated predictor (Figure 4b) is given by
$y(k) = \Phi\left[Y_p(k\text{-}1), U(k\text{-}1)\right]$ with $Y_p(k\text{-}1) = \left\{y_p(k\text{-}1), y_p(k\text{-}2), \ldots , y_p(k\text{-}N)\right\}$.
Therefore, the predictor of the equation error process is actually a *non-recursive* predictor, whose inputs are the external inputs of the process and the (measured) outputs of the process. If there exists a neural network which can approximate

function Φ, this network can implement the predictor, *and a directed algorithm* [1] is the only suitable choice, since the predictor is not recursive.

To summarize, the algorithms derived for the training of discrete-time recurrent neural networks can readily be applied to the identification of dynamical non-linear processes. Directed algorithms are best suited to the training of neural networks intended to predict the output of processes satisfying the perturbation-free hypothesis or the equation error hypothesis, whereas undirected algorithms are best suited to the NARMAX and output error hypotheses. It is intuitive, and it can be shown analytically in simple cases [6], that semi-directed algorithms bridge the gap between these approaches.

3.3 Illustration: identification of a first-order non-linear process

In this section, we propose several illustrations of the above algorithms. We first train a neural network, both adaptively and non-adaptively, to model a deterministic, noise-free simulated process. In section 3.3.2, we add output noise to the same deterministic equation as above, and we train a network, non-adaptively, to model the resulting process. We pretend that we do not know how noise interferes with the process; we first make the assumption that the process is appropriately described by an output error model, and we train the network accordingly with an undirected algorithm; we subsequently make the assumption that the process is appropriately described by an equation error model, and we train the neural network accordingly with a directed algorithm; we compare the results obtained in these two cases. Finally, in section 3.3.3, we add state noise to the same deterministic equation as above, and we train a network, non-adaptively, to model the resulting process; we make the same two assumptions as above. The detrimental effect of using the wrong algorithm, i.e. of making the wrong assumption on the influence of the noise on the process, is shown clearly on all these examples.

All results presented here were obtained by the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [7], with step adaptation by the method of Wolfe and Powell [8].

3.3.1 - Example 1: perturbation-free process

3.3.1.1 - Simulation equation

A continuous-time process is simulated by the following discrete-time equation

$$y_p(k) \equiv \Psi\left[y_p(k\text{-}1), u(k\text{-}1)\right] = \left[1 - \frac{T}{a+by_p(k\text{-}1)}\right] y_p(k\text{-}1) + \left[T\,\frac{c+dy_p(k\text{-}1)}{a+by_p(k\text{-}1)}\right] u(k\text{-}1) \ ,$$

where $y_p(k)$ is the output of the process at time k, and u(k) is the external input at time k. In the following, the values of the parameters are:
a=-0.139, b=1.2, c=5.633, d=-0.326, sampling period T= 0.1 sec.

3.3.1.2 - Adaptive identification of the noise-free process

We first identify the process adaptively, making the assumption that it can be adequately described in the vicinity of an operating point by a linear first-order model; this approach is useful if the model is to be used, with small input and ouput signals, within an adaptive control system as an alternative to gain scheduling. It can be implemented by a "neural network" made of a single, linear neuron; in the absence of perturbations, the appropriate training algorithm is a directed algorithm. The behaviour of the adaptive predictor, and the prediction error, are illustrated on Figure 5.

3.3.1.3 - Non-adaptive identification of the noise-free process

The process can also be identified non-adaptively. The predictor must then be valid in a suitable region of state space; it can be used in the case of large input and output signals. This can be achieved, in the present case, by a feedforward neural network with one hidden layer of five neurons. The training set is a sequence of 100 steps of random amplitude. Training has been performed by a non-recursive, iterative, directed algorithm with $N_c(=N_t)=2000$. Figure 6 illustrates the behaviour of the non-adaptive predictor, and the prediction error.

3.3.2 - Example 2: process with additive output noise

The simulated process that we consider now is described by the same equation as in the previous section, with additive noise on the output:

$$x_p(k) = \Psi\left[x_p(k-1),\ u(k-1)\right]$$
$$y_p(k) = x_p(k) + w(k)\ .$$

w(k) is white noise with maximum amplitude 0.5.

The goal of identification is to find a (neural network) predictor that implements a function as close as possible to Ψ in a bounded domain of state space; therefore, the prediction error should be as close as possible to the noise w(k) once training is completed.

We first make the (correct) assumption that an output error model is appropriate. Thus, we use a recurrent predictor of the type shown on Figure 2b; the feedforward part of the neural network has the same architecture as in the perturbation-free case (since we know from the previous section that such a network can approximate function Ψ with satisfactory accuracy), and we train it with an *undirected* algorithm (undirected, non recursive, iterative with N_t=N_c=2000). Figure 7a show the response of the network at the end of training, and Figure 7b shows the prediction error. As expected, the latter is just white noise of amplitude 0.5, which shows that (i) the feedforward part of the predictor network is appropriate for the approximation of function Ψ, that (ii) the undirected training algorithm is the appropriate algorithm for training the predictor, or, in other words, that the assumption that the process can be described by an output error model is correct, and that (iii) the quasi-Newton gradient method (of constant use in recursive identification [3]) allows a very efficient optimization of the cost function; this may seem to be a side issue, but it is worth pointing out that the results presented in this paper would not have been obtained in any reasonable time otherwise.

We now make the (wrong) assumption that the process can be described by a NARX model. Accordingly, we choose a neural network predictor of the type shown on Figure 4, which we train with a *directed* algorithm on the same data as before. Figure 8 shows the prediction error after training (directed, non recursive, iterative algorithm with N_c=2000), with the same inputs as shown on Figure 7a: the vairance of the prediction error is much larger than in the previous case, thereby showing that the training algorithm is inappropriate for extracting the model in the presence of the additive output noise.

3.3.3 - Example 3: process with additive state noise

In this section, we consider again the same simulation equation as in section 3.3.1.1, but we add white noise, with amplitude 0.5, to its state:

$x_p(k) = \Psi\left[x_p(k\text{-}1), u(k\text{-}1)\right] + w(k)$,

$y_p(k) = x_p(k)$.

We first make the (correct) assumption that a NARX model is appropriate. Thus, we use a predictor as shown on Figure 4, with five hidden neurons, trained by a *directed* algorithm (non recursive, iterative with N_c=2000). The result after training is exactly as shown on Figure 7b: the prediction error is just white noise, which shows that the identification of the process by the neural network has been perfectly successful.

If we now make the (wrong) assumption that the process can be represented by an output error model, we take a predictor as shown on Figure 2, we use five hidden neurons in the feedforward part of the network, and we train the model with an undirected algorithm; the resulting prediction error is as shown on Figure 9: the error is clearly not white noise, thereby showing that the use of an *undirected* algorithm with additive state noise prevents the network from correctly extracting the model, although we know from the previous examples that the network has the appropriate number of hidden units for approximating function Ψ.

4 CONCLUSION

We have shown the importance of choosing an appropriate training algorithm for the modeling of a dynamical system in the presence of noise. Directed (teacher forcing) algorithms are appropriate for the modeling of noiseless dynamical systems, or for systems in which random perturbations can be considered as white noise added to the state variables of the black-box model, whereas undirected algorithms are appropriate for predicting the output of systems in which random perturbations can be considered as white noise added to the output of the black-box model. Although the architecture of the feedforward part of the neural predictors is the same in all the above examples, and is known to be appropriate for describing the non-linearity of the process, very different results can be obtained, depending on the algorithm used. Within the appropriate family, other choices (recursive or non-recursive algorithm, iterative or non-iterative algorithm, values of N_c and N_t, ...) are important but less critical; they will be made on the basis of the stationarity time of the process, of the computer time available, etc...

In the above examples, semi-directed algorithms have not been used because no stability problem was encountered with undirected algorithms: semi-directed algorithms are useful when an output error model describes the process appropriately, but when the corresponding predictor is unstable. Detailed stability analyses of undirected algorithms have been performed in simple cases [9].

REFERENCES

[1] O. Nerrand, P. Roussel-Ragot, L. Personnaz, G. Dreyfus, S. Marcos, "Neural Networks and Non-linear Adaptive Filtering: Unifying Concepts and New Algorithms", *Neural Computation,* vol. 5, pp. 165-197 (1993).

[2] S. Marcos, P. Roussel-Ragot, L. Personnaz, O. Nerrand, G. Dreyfus, C. Vignat, "Réseaux de Neurones pour le Filtrage Non-linéaire Adaptatif", *Traitement du Signal*, vol. 8, pp. 409-422 (1993).

[3] L. Ljung, T. Söderström, *Theory and Practice of Recursive Identification,* MIT Press (1983).

[4] C.L. Giles, G.Z. Sun, H.H. Chen, Y.C.Lee, D. Chen, "Higher Order Recurrent Networks and Grammatical Inference", *Advances in Neural Information Processing Systems 2,* D.S. Touretzky, ed., pp. 380-387 (1990).

[5] S. Chen, S.A. Billings, "Representations of Non-Linear Systems: the NARMAX Model". *Int. J. Control,* vol 49, pp. 1013-1032 (1989).

[6] G. Dreyfus, O. Macchi, S. Marcos, O. Nerrand, L. Personnaz, P. Roussel-Ragot, D. Urbani, C. Vignat, "Adaptive Training of Feedback Neural Networks for Non-linear Filtering", *Neural Networks for Signal Processing II,* S.Y. Kung, F. Fallside, J. Aa. Sorenson, C.A.Kamm, eds (1992).

[7] W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, *Numerical Recipes: The Art of Scientific Computing,* Cambridge University Press (1986).

[8] P. Wolfe, *Convergence Conditions for Ascent Methods,* S.I.A.M. Review vol. 11, pp. 226-235 (1969).

[9] C. Vignat, "Convergence des Approches Filtrage Adaptatif et Réseaux de Neurones Formels. Cas des Systèmes Non-Linéaires Bouclés". Thèse de l'Université de Paris-Sud, Orsay (1993).

FIGURE CAPTIONS

Figure 1:
a. The canonical form of a discrete-time recurrent network.
b. Copy m at time n of the feedforward part of the canonical form.

Figure 2:
a. Structure of the model of the process under the *output error* hypothesis.
b. Associated neural predictor.

Figure 3:
a. Structure of the model of the process under the *NARMAX* hypothesis.
b. Associated neural predictor.

Figure 4:
a. Structure of the model of the process under the *equation error* hypothesis.
b. Associated neural predictor.

Figure 5:
Example 1: adaptive identification of the perturbation-free process. Training with a directed recursive algorithm (N_c=20).
a. Input and output of the adaptive predictor.
b. Prediction error.

Figure 6:
Example 1: non-adaptive identification of the perturbation-free process. Training with a directed iterative algorithm (N_c=2000).
a. Input and output of the non-adaptive predictor after training.
b. Prediction error.

Figure 7:
Example 2: non-adaptive identification of the process with additive output noise . Training with an undirected iterative algorithm (N_t=N_c=2000), corresponding to the correct hypothesis (output error model).
a. Input and outputs of the simulated process and of the predictor.
b. Prediction error.

Figure 8:
Example 2: non-adaptive identification of the process with additive output noise. Prediction error after training with a directed iterative algorithm (N_c=2000), corresponding to a wrong hypothesis (equation error model).

Figure 9:
Example 3: non-adaptive identification of the process with additive state noise. Prediction error after training with an undirected iterative algorithm (N_t=N_c=2000), corresponding to a wrong hypothesis (output error model).

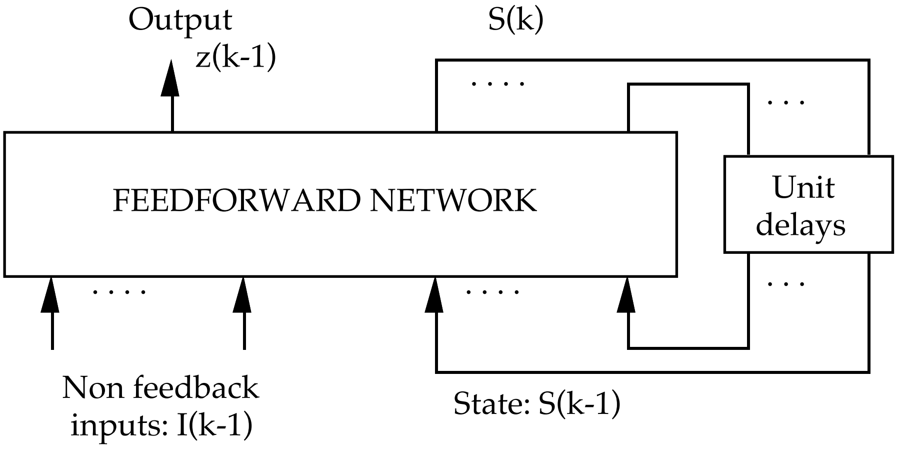

FIGURE 1a

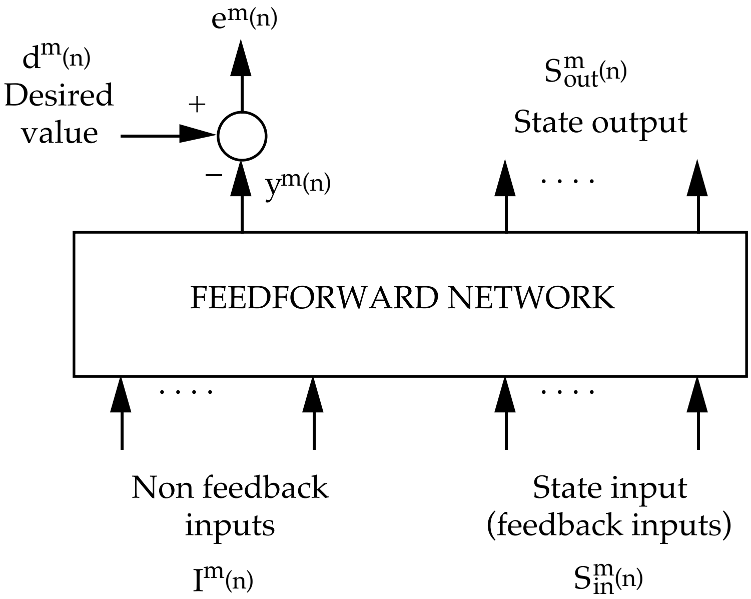

FIGURE 1b

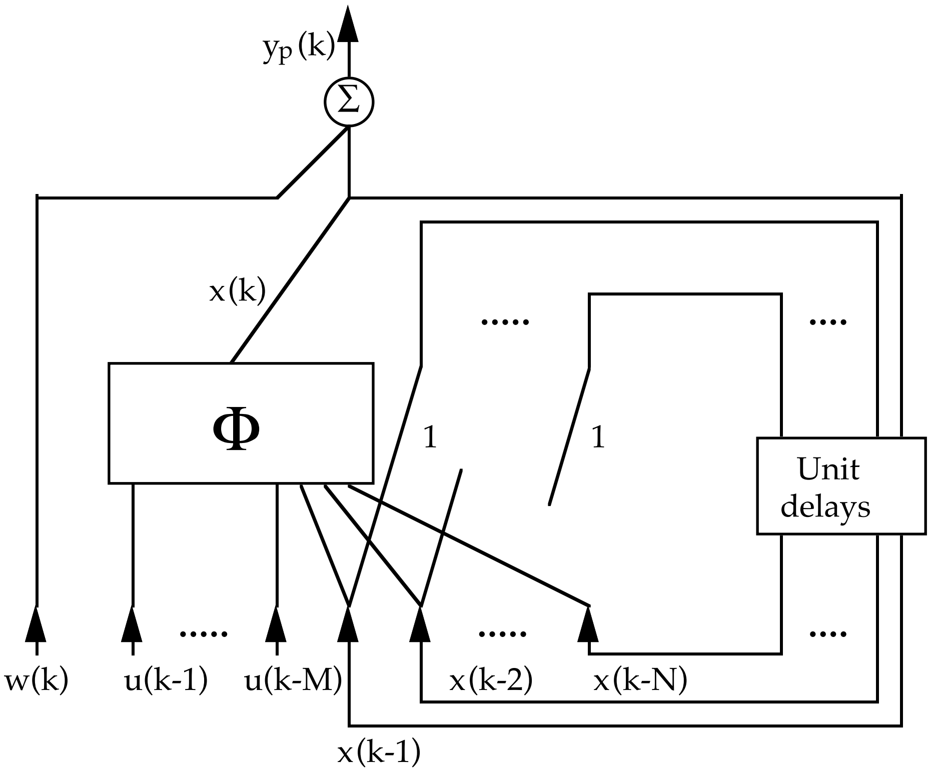

FIGURE 2a

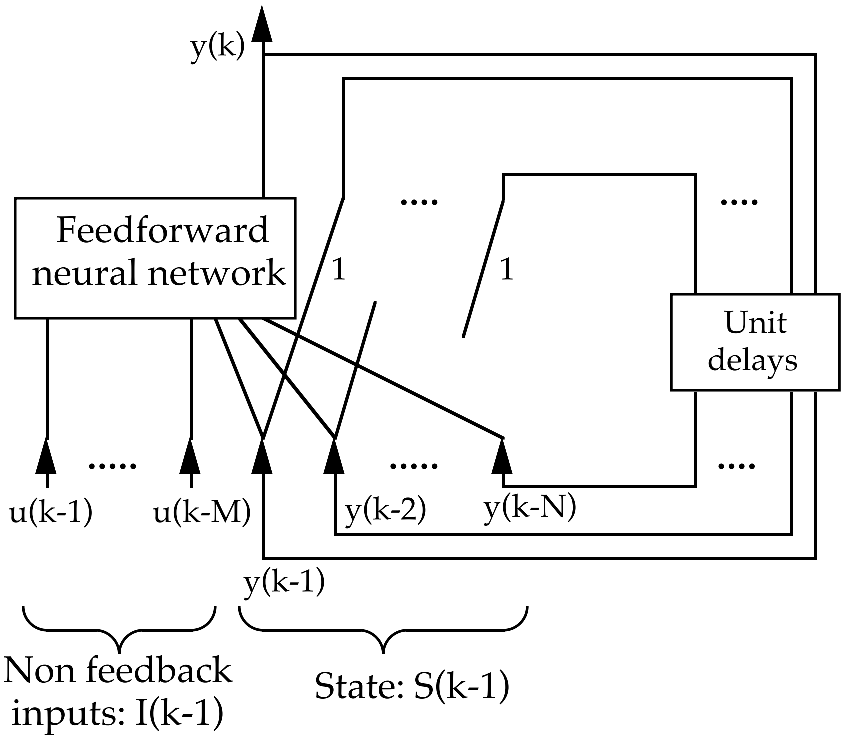

FIGURE 2b

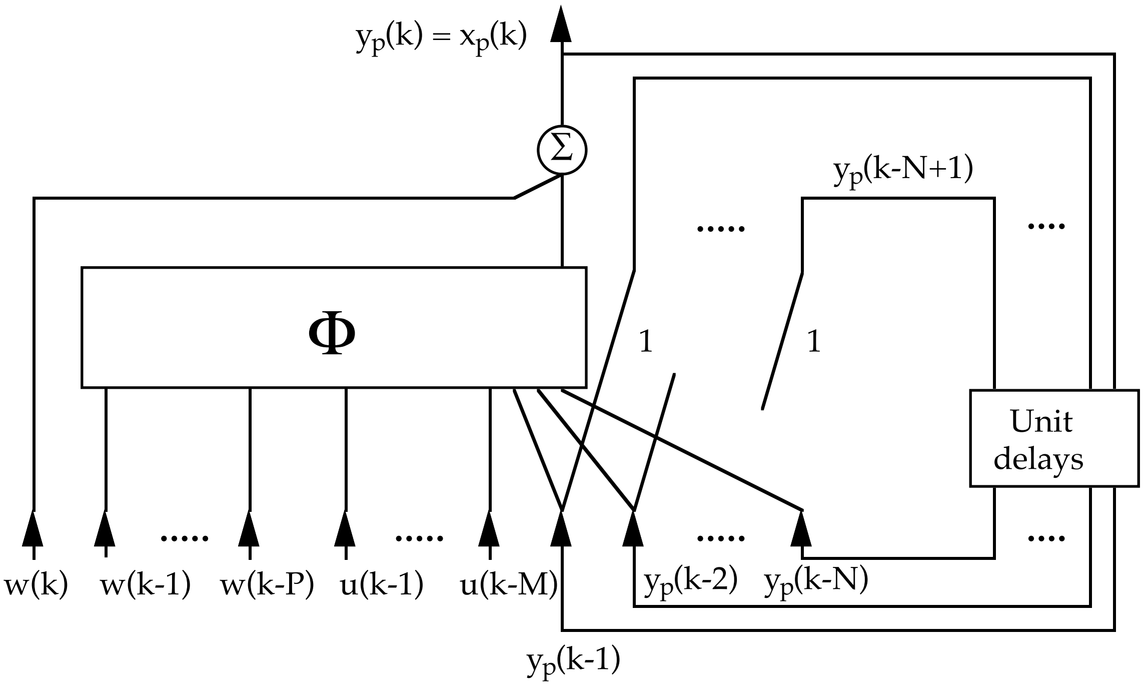

FIGURE 3a

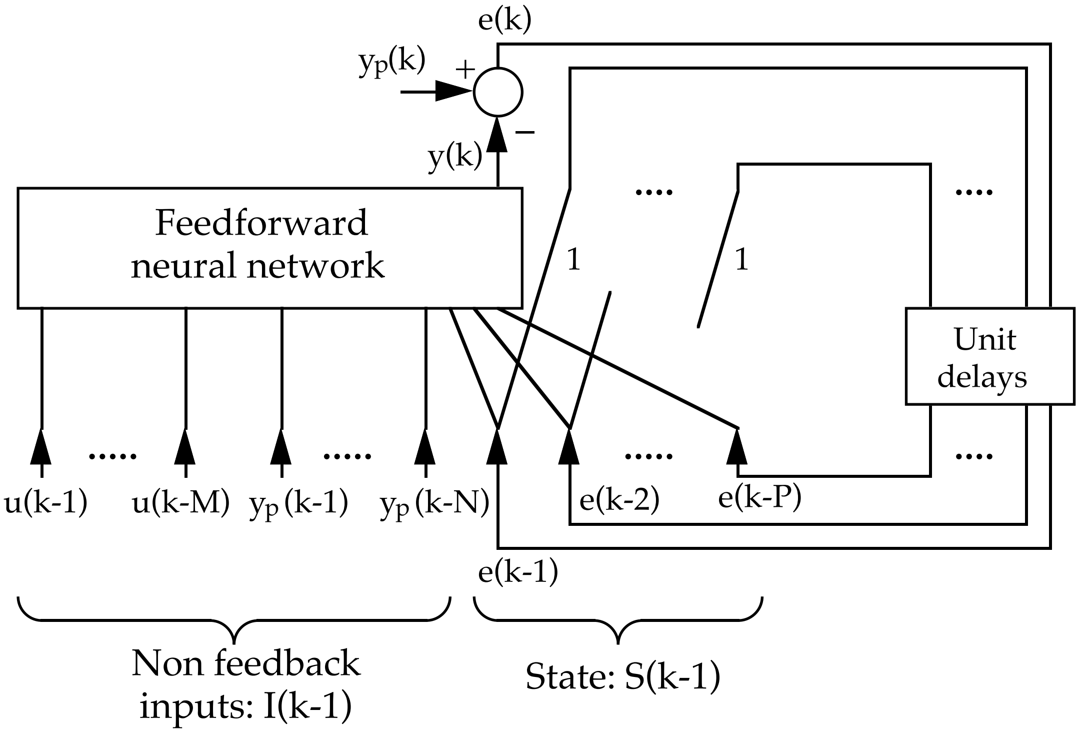

FIGURE 3b

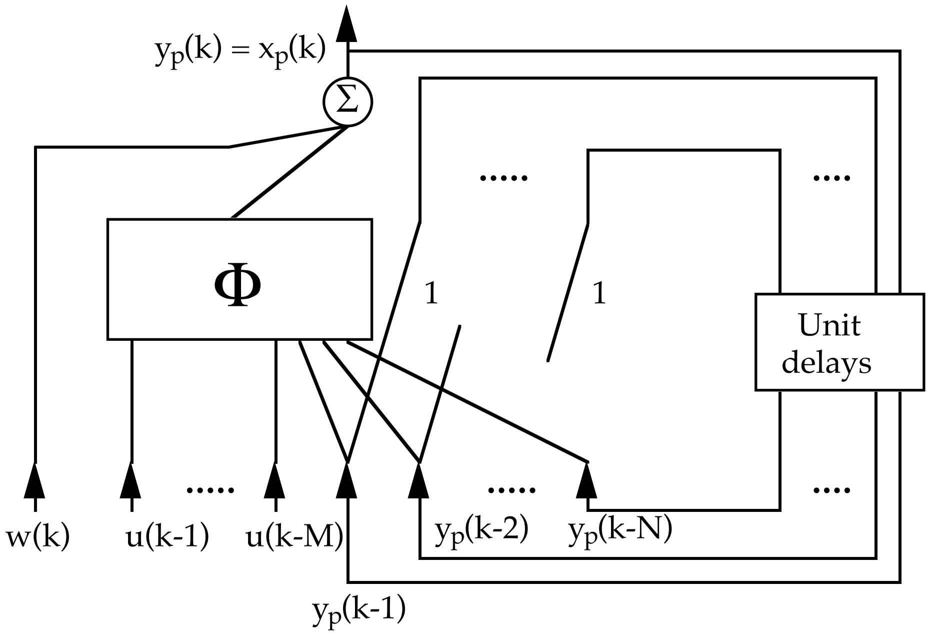

FIGURE 4a

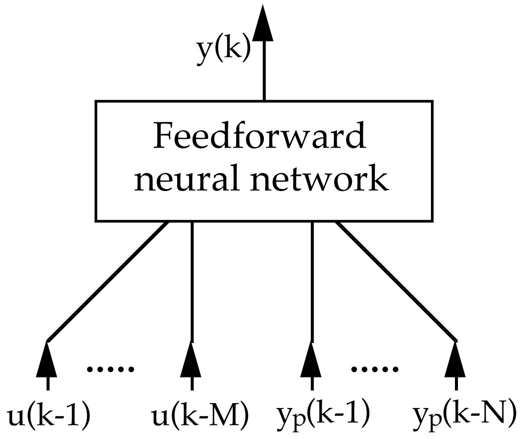

FIGURE 4b

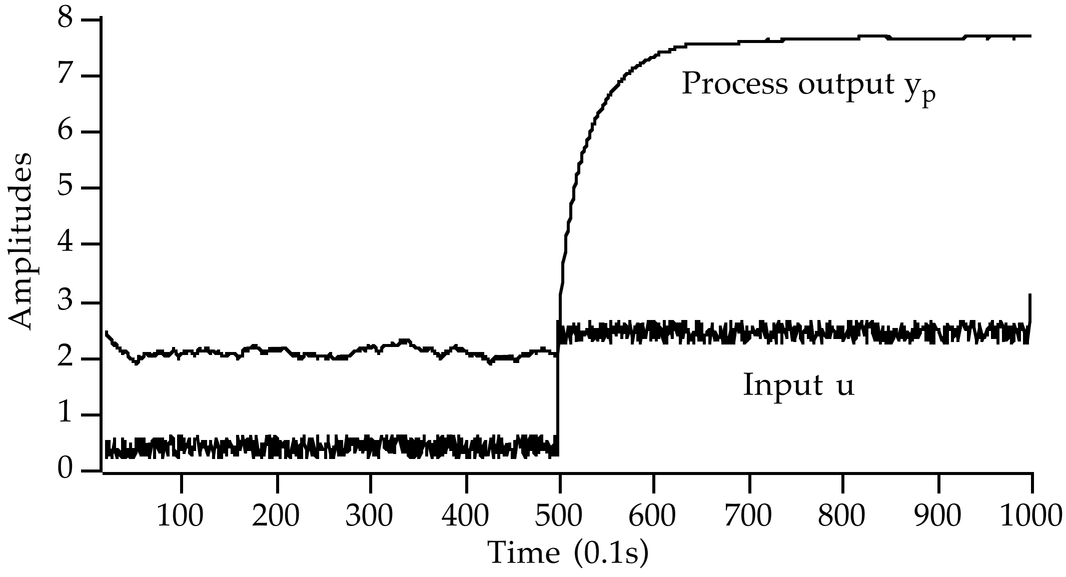

FIGURE 5a

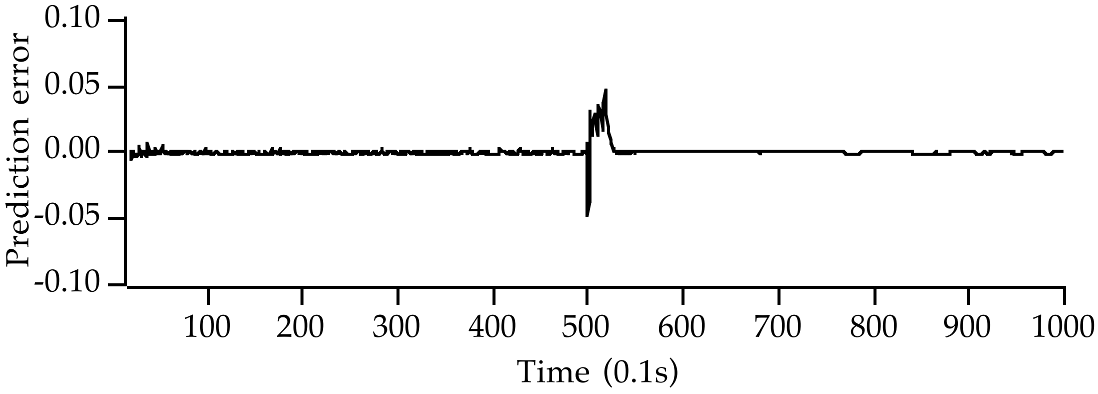

FIGURE 5b

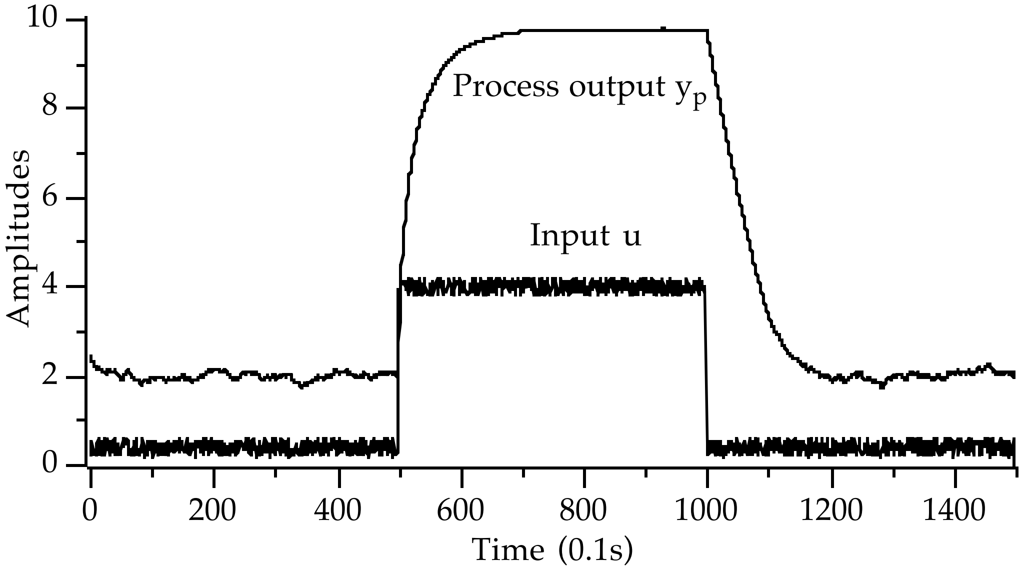

FIGURE 6a

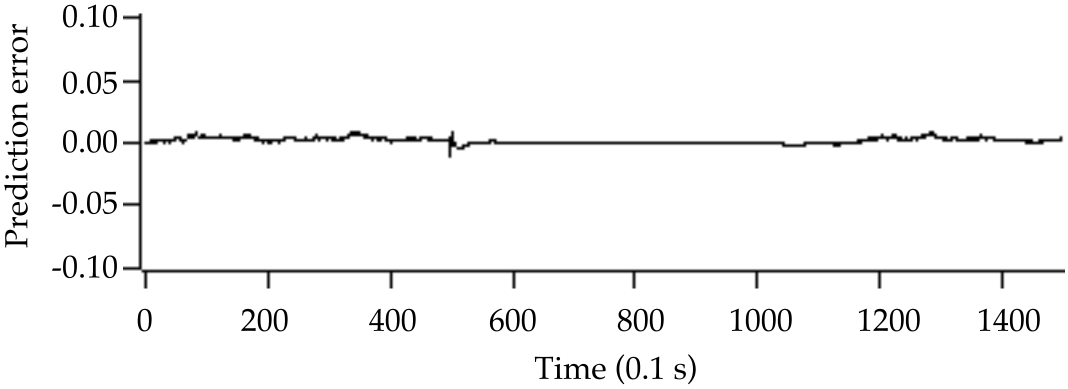

FIGURE 6b

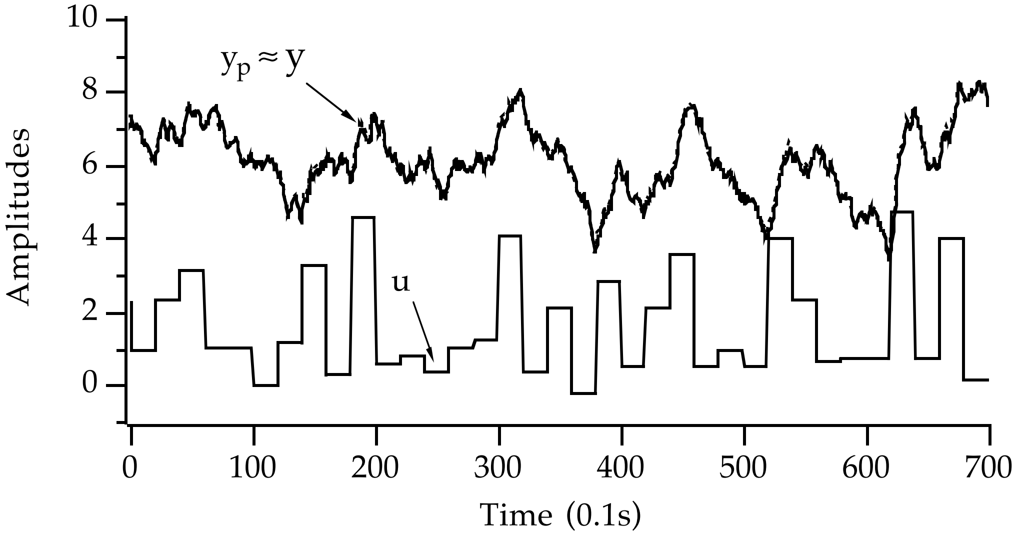

FIGURE 7a

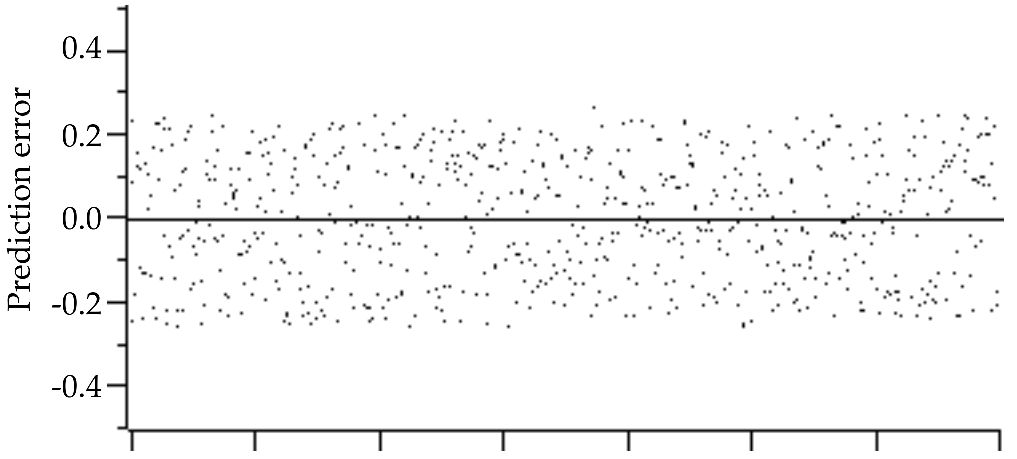

FIGURE 7b

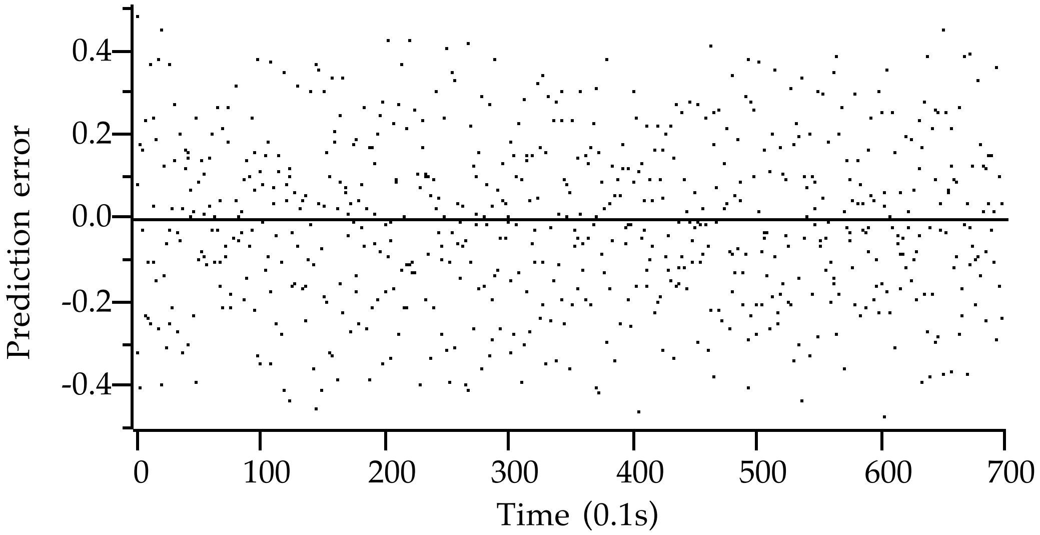
0.4
0.2
0.0
-0.2
-0.4
Prediction error
0
100
200
300
400
500
600
700
Time (0.1s)

FIGURE 8

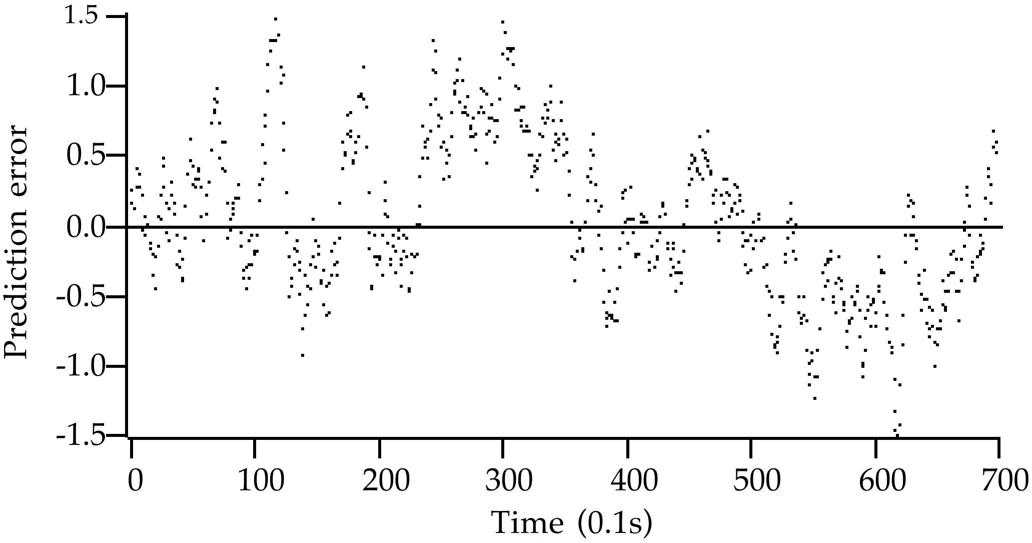
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
Prediction error
0
100
200
300
400
500
600
700
Time (0.1s)

FIGURE 9