In : "Neural Adaptive Control Technology", R. Zbikowski and K. J. Hunt eds., World Scientific (1996), pp. 237-264.

BLACK-BOX MODELING WITH STATE-SPACE NEURAL NETWORKS

ISABELLE RIVALS and LéON PERSONNAZ

ESPCI, Laboratoire d'Électronique, 10 rue Vauquelin
75231 Paris Cedex 05, France.
E-mail: rivals@neurones.espci.fr, personna@neurones.espci.fr

ABSTRACT

Neural network black-box modeling is usually performed using nonlinear input-output models. The goal of this paper is to show that there are advantages in using nonlinear state-space models, which constitute a larger class of nonlinear dynamical models, and their corresponding state-space neural predictors. We recall the fundamentals of both input-output and state-space black-box modeling, and show the state-space neural networks to be potentially more efficient and more parsimonious than their conventional input-output counterparts. This is examplified on simulated processes as well as on a real one, the hydraulic actuator of a robot arm.

1. Introduction

During the past few years, several authors [Narendra and Parthasarathy 1990, Nerrand et al. 1994] have suggested neural networks for nonlinear dynamical black-box modeling. The problem of designing a mathematical model of a process using only observed data has attracted much attention, both from an academic and an industrial point of view. Neural models can be used either as simulators (for fault detection, personnel training, etc., see for example [Ploix et al. 1994]), as models for the training of a neural controller, and/or as models to be used within control systems [see for instance the robust "neural" internal model control of a 4WD Mercedes vehicle in Rivals et al. 1994].

To date, most of the work in neural black-box modeling has been performed making the assumption that the process to be modeled can be described accurately by input-output models, and using the corresponding input-output neural predictors. We show in the present paper that, since state-space models constitute a larger class of nonlinear dynamical models, there is an advantage in making the assumption of a state-space description, and in using the corresponding state-space neural predictors. In section 2, we give some useful definitions, and show state-space neural networks to be potentially more efficient than their conventional input-output counterparts. In section 3, in order to enlighten the links and the differences between state-space and

input-output black-box modeling, we recall some results in linear modeling. In section 4.1, we present a family of nonlinear input-output models and their associated predictors, and in section 4.2, a family of nonlinear state-space models is presented in the same manner. In section 5, we describe the training of the input-output and state-space neural predictors in a unified fashion. State-space modeling is then illustrated and compared to input-output modeling on simulated processes in section 6, and on the hydraulic actuator of a robot arm in section 7. In the latter case, our results are compared to those obtained by other groups on the same data.

2. Some Definitions

The problem of modeling a process is to find a mathematical model which is able to describe its dynamic behavior, given some prior knowledge about the process and input-output measurements. Two types of models can be chosen according to the prior knowledge one has about the internal behavior of the process: knowledge-based models and black-box models. We are interested in the latter type.

2.1. Knowledge-Based versus Black-Box Modeling

A model is termed knowledge-based if it has been possible to construct it entirely from prior knowledge and physical insight, or if the physical insight suggests a model structure where the only unknowns are parameters which will be estimated from measurements. Knowledge-based models are usually in the state-space form, their state variables and the relationships between them having a physical meaning.

If no or unsufficient physical insight is available, but only observed data, no model structure can be defined *a priori*. In this case, one must resort to black-box models. A discrete-time black-box model is a recursive filter whose outputs are functions of its past external inputs and state variables; these functions are defined by a set of parameters (polynomial functions, or neural networks, radial basis networks, wavelet networks…). For nonlinear modeling, the family of parameterized functions should be as flexible as possible, in order to be able to describe any complex dynamical process. Black-box models are usually chosen to be input-output models; our aim is precisely to show that it may be advantageous to use state-space models as black-box models.

2.2. Input-Output versus State-Space Black-Box Models

We are interested in black-box state-space models, for the following reasons:

- First, state-space models can describe a broader class of dynamical systems than input-output models: whereas it is always possible to rewrite a nonlinear input-

output model in a state-space representation, conversely, an input-output model globally equivalent to a given state-space model might not exist [Leontaritis and Billings 1985];

- Second, even when an input-output representation does exist, it may have a higher order. Let us consider the following deterministic SISO state-space model:

$$\begin{cases} x(k+1) = f\left(x(k),\, u(k)\right) & (\textit{state equation}) \\ y(k) = g\left(x(k)\right) & (\textit{output equation}) \end{cases} \tag{2.2.1}$$

where $u(k)$ is the scalar external input, $y(k)$ is the scalar output, and $x(k)$ is the n-state vector of the model at time k; f and g are nonlinear functions. It has been shown that, under fairly general conditions on the observability of (2.2.1), an equivalent input-output model does exist, and that it is given by:

$$y(k) = h\left(y(k\text{-}1),\, \ldots,\, y(k\text{-}r),\, u(k\text{-}1),\, \ldots,\, u(k\text{-}r)\right) \tag{2.2.2}$$

with $n \le r \le 2\,n + 1$ [Levin 1992, Levin and Narendra 1995].

Therefore, state-space models are likely to have a lower order and to require a smaller number of past inputs (i.e. a smaller number of regressors) than input-output models, and hopefully a smaller number of parameters. This is of importance especially when a limited amount of data is available.

2.3. Black-Box Modeling

The black-box modeling of a process is achieved in three steps: 1) choice of a set of candidate models; 2) derivation of the associated predictors; 3) selection of the best model.

Step 1: choice of a set of candidate models.

The analysis of the process behavior leads to a set of control inputs and measured disturbances - the external inputs - acting on the observed outputs. In this paper, all unmeasured disturbances are supposed to be stochastic processes. This analysis also gives possible values for the order of a black-box model, either input-output or state-space. Candidate models are thus characterized by their input-output or state-space structure, their order and number of external inputs, and by the functional relationships between their inputs and their output.

In this paper, we will consider time-invariant stochastic SISO models (the extension to MISO models is straightforward):

- Input-output candidate models:

$$y_p(k) = h\left(y_p(k\text{-}1),\, \ldots,\, y_p(k\text{-}n),\, u(k\text{-}1),\, \ldots,\, u(k\text{-}m),\, w(k),\, \ldots,\, w(k\text{-}p)\right) \tag{2.3.1}$$

where $y_p(k)$ denotes the output of the process and $u(k)$ the known external input at time k; $\{w(k)\}$ is a sequence of zero mean, independent and identically distributed (i.i.d.) random variables; h is a nonlinear function.

- State-space candidate models:

$$\begin{cases} x_p(k+1) = f\left(x_p(k),\, u(k),\, v_1(k)\right) \\ y_p(k) = g\left(x_p(k),\, v_2(k)\right) \end{cases} \tag{2.3.2}$$

where $y_p(k)$ denotes the output of the process, $u(k)$ the known external input, and $x_p(k)$ the n-state vector at time k; $\{v_1(k)\}$ is a sequence of zero mean, i.i.d. random vectors (state noise), and $\{v_2(k)\}$ is a sequence of zero mean, i.i.d. random variables (output noise); f and g are nonlinear functions. Since we deal with black-box modeling, the state x_p is not measured.

A candidate model, built upon a set of assumptions about the process, is called an *assumed model*.

Step 2: derivation of the associated predictors.

We define the *theoretical predictor associated to a given assumed model* as the recursion giving the conditional expectation $E\left(y_p(k+1) \mid k\right)$ of the output $y_p(k+1)$ given the past observations $\{y_p(k),\, y_p(k\text{-}1),\, \ldots,\, y_p(0)\}$ if the assumed model is true.

Theoretical predictor associated to an input-output assumed model:

$$y(k+1) = h_{pred}\left(y_p^k,\, u^k,\, y^k,\, k\right) \tag{2.3.3}$$

where y is the predictor output, and y_p^k denotes a finite set of past outputs of the process (the same notation is used for u, y, x). h_{pred} is a nonlinear function, generally time-varying, depending on h in (2.3.1), and whose expression will be given in section 4 for some particular input-output assumed models.

Theoretical predictor associated to a state-space assumed model:

$$\begin{cases} x(k+1) = f_{pred}\left(x^k,\, u^k,\, y_p^k,\, k\right) \\ y(k+1) = g_{pred}\left(x^k,\, u^k,\, y_p^k,\, k\right) \end{cases} \tag{2.3.4}$$

where y is the predictor output, and x its state vector. f_{pred} and g_{pred} are nonlinear functions depending on f and g in (2.3.2), whose expressions will be given in section 5 for some particular models.

The theoretical predictor associated to the true model is thus the *minimal variance* - or *optimal - predictor*. At this step of the modeling procedure, the goal is to find an *empirical predictor* as close as possible to the optimal predictor.

Once candidate assumed models have been chosen, empirical predictors giving an estimate of $E\left(y_p(k+1) \mid k\right)$ are then designed, using the theoretical predictors associated to the candidate models: the functions h_{pred} , or f_{pred} and g_{pred}, of each theoretical predictor are replaced by *functions parameterized by a set of parameters* θ. Any universal function approximator can be used. In the modeling examples of sections 6 and 7, we use neural networks with sigmoidal hidden neurons: these networks are known to be universal [Hornik et al. 89] and parsimonious [Hornik et al. 94] approximators. For a review of other possible approximators, see [Sontag 93, Sjöberg et al. 1995]. The training of a candidate neural predictor is achieved

using input-output sequences. Its weights can be estimated with a *prediction error* (P.E.) method, or possibly an *extended Kalman filter* (E.K.F.) method.

Step 3: selection of the best model.

Eventually, the performances of the candidate predictors are estimated with the system the model is being designed for (predictor, simulator, control system…). According to these performances, the "best" predictor is chosen, and hence the best model.

Parts 3 and 4 are devoted to the theoretical predictors associated to various assumed models, which will be needed for the design of neural empirical predictors.

3. Linear Modeling

In order to show the links and the differences between state-space and input-output modeling, some results in linear modeling are recalled. These results are mostly of tutorial nature, and we will generally refer to [Goodwin and Sin 1984].

3.1. The Linear State-Space Assumed Model

The general SISO linear time-invariant stochastic model is:

$$\begin{cases} x_p(k+1) = F\,x_p(k) + G\,u(k) + v_1(k) \\ y_p(k) = H\,x_p(k) + v_2(k) \end{cases} \tag{3.1.1}$$

with dim $(F) = n \times n$, dim $(G) = n \times 1$, and dim $(H) = 1 \times n$.

Let us assume that the initial state $x_p(0)$ is gaussian with mean μ_0, and that the noises are also gaussian. The conditional expectation of the state $x_p(k+1)$ and of the output $y_p(k+1)$ of model (3.1.1) given observations $y_p(0)$ up to $y_p(k)$ can be estimated by the *Kalman predictor*. Let $x(k+1)$ and $y(k+1)$ denote these predictions. They satisfy the following recursion:

$$\begin{cases} x(0) = \mu_0 \\ x(k+1) = F\,x(k) + G\,u(k) + K(k)\,(y_p(k) - H\,x(k)) \\ y(k+1) = H\,x(k+1) \end{cases} \tag{3.1.2}$$

where $K(k)$ is the time-varying Kalman gain. Under certain stabilizability and observability conditions, the error covariance and the Kalman gain $K(k)$ converge to *steady-state* values as k tends to infinity.

If the gaussian assumption is removed, (3.1.2) is the minimum variance *linear* predictor.

In the modeling problem, the matrices F, G, H, K are parameterized by an unknown vector θ, which can be estimated using a P.E. or an E.K.F. method:

- Prediction Error method: the steady-state form of predictor (3.1.2) ($K(k) = K \; \forall k$) is used. The estimation of θ is obtained by minimizing the mean square prediction error (MSPE):

$$J(\theta) = \frac{1}{N} \sum_{k=0}^{N-1} e(k+1)^2 = \frac{1}{N} \sum_{k=0}^{N-1} (y_p(k+1) - y(k+1))^2 \qquad (3.1.3)$$

- Extended Kalman Filter method: the unknown parameter vector θ is included in an augmented state vector. The E.K.F. gives a suboptimal estimation of the state vector (and therefore of θ) using a linearization of the model around the current state estimate.

3.2. The Innovations Model

Using the *innovations* $\varepsilon(k)$, defined as $\varepsilon(k) = y_p(k) - H\,x(k) = y_p(k) - y(k)$, the Kalman predictor can be rewritten as what is called the innovations model, disturbed by the random innovations sequence only:

$$\begin{cases} x(k+1) = F\,x(k) + G\,u(k) + K(k)\,\varepsilon(k) \\ y_p(k) = H\,x(k) + \varepsilon(k) \end{cases} \qquad (3.2.1)$$

The latter is equivalent to the following *time-varying input-output model*:

$$A(q^{-1})\,y_p(k) = B(q^{-1})\,u(k) + C(k, q^{-1})\,\varepsilon(k) \qquad (3.2.2)$$

where q^{-1} is the backward shift operator, A, B and C are polynomials of degree n, A and C being monic, and with $B(q^{-1}) = b_1\,q^{-1} + \ldots + b_n\,q^{-n}$. The time-varying nature of C in (3.2.2) arises from the fact that the Kalman gain is time-varying. However, if the Kalman gain is asymptotically time-invariant, C is also asymptotically time-invariant. We can thus replace the time-varying model (3.2.2) by its steady-state form:

$$A(q^{-1})\,y_p(k) = B(q^{-1})\,u(k) + C(q^{-1})\,\varepsilon(k) \qquad (3.2.3)$$

3.3. The ARMAX Assumed Model

The equivalence between the state-space model (3.1.1) and the input-output model (3.2.3) in the steady state motivates the choice of time-invariant input-output assumed models:

$$A(q^{-1})\,y_p(k) = B(q^{-1})\,u(k) + C(q^{-1})\,w(k) \qquad (3.3.1)$$

where A, B and C are polynomials of degree n, m and p, A and C being monic, and where $B(q^{-1}) = b_1\,q^{-1} + \ldots + b_m\,q^{-m}$; the noise $\{w(k)\}$ is a sequence of zero mean, i.i.d. random variables. This model is called the ARMAX (AutoRegressive Moving Average with eXogenous input) model.

The designer thus has the choice between two assumed models which are equivalent in the steady state, the model in the state-space form (3.1.1) and the ARMAX model (3.3.1). The latter is often chosen for the simplicity of the modeling procedure. Consider the following one-step-ahead predictor:

$$C(q^{-1})\, y(k+1) = \left(C(q^{-1}) - A(q^{-1})\right) y_p(k+1) + B(q^{-1})\, u(k+1) \qquad (3.3.2)$$

or equivalently:

$$y(k+1) = \left(1 - A(q^{-1})\right) y_p(k+1) + B(q^{-1})\, u(k+1) + \left(C(q^{-1}) - 1\right) e(k+1) \qquad (3.3.3)$$

where $e(k+1) = y_p(k+1) - y(k+1)$ is the prediction error*. Provided that the assumed model is true, and that the polynomial C has no zeros on or outside the unit circle, the effect of arbitrary initial conditions will diminish exponentially, and the prediction error e becomes equal to the noise w: predictor (3.3.3) is *asymptotically* optimal. Therefore, we will consider (3.3.3) as the theoretical predictor associated to the assumed model (3.3.1).

The process identification can thus be performed by minimizing the MSPE of a predictor parameterized by θ:

$$\begin{aligned} y(k+1) = &\left(1 - A(\theta, q^{-1})\right) y_p(k+1) + B(\theta, q^{-1})\, u(k+1) \\ &+ \left(C(\theta, q^{-1}) - 1\right) e(k+1) \end{aligned} \qquad (3.3.4)$$

Several particular cases of the ARMAX model are well known:

The ARX (AutoRegressive with eXogenous input) model.

It is also called the "*Equation-Error*" model. It is obtained by taking $C(q^{-1}) = 1$:

$$A(q^{-1})\, y_p(k) = B(q^{-1})\, u(k) + w(k) \qquad (3.3.5)$$

The theoretical predictor associated to this ARX model is:

$$y(k+1) = \left(1 - A(q^{-1})\right) y_p(k+1) + B(q^{-1})\, u(k+1) \qquad (3.3.6)$$

The "Output-Error" model.

It corresponds to the assumption of *additive output noise* (i.e. $C(q^{-1}) = A\,(q^{-1})$):

$$A(q^{-1})\, y_p(k) = B(q^{-1})\, u(k) + A(q^{-1})\, w(k) \qquad (3.3.7)$$

If the assumption is true, and if A has no zeros on or outside the unit circle, the following theoretical predictor is asymptotically optimal:

$$y(k+1) = \left(1 - A(q^{-1})\right) y(k+1) + B(q^{-1})\, u(k+1) \qquad (3.3.8)$$

* It is important to distinguish between the noise sequence occuring in a given assumed model ($\{v_1(k)\}$, $\{v_2(k)\}$, or $\{w(k)\}$), the prediction error sequence $\{e(k)\}$ of some predictor, and the innovations sequence $\{\varepsilon(k)\}$, which is the prediction error sequence of the optimal predictor.

In : "Neural Adaptive Control Technology", R. Zbikowski and K. J. Hunt eds., World Scientific (1996), pp. 237-264.

4. Nonlinear Modeling

As recalled in section 2 for deterministic models, in contradistinction to the linear case, there is no simple equivalence between nonlinear input-output and state-space models. Since the latter build a larger class of nonlinear models, it would be too restrictive to make only input-output assumptions. Furthermore, even when there exists an input-output model equivalent to a given state-space assumed model, its function h_{pred} in (2.3.3) might require many more arguments than the functions f_{pred} and g_{pred} of the equivalent state-space model (2.3.4). Therefore, if input-output modeling is unsatifactory (typically if the training becomes uneasy because too many inputs and/or neurons are needed), one should resort to neural state-space models. Their state variables not being imposed to be delayed values of their output, these are more flexible.

As in the linear case, a possible presentation would start with state-space modeling, and deal with input-output modeling as a particular case. We choose first to recall the input-output approach, which should be tried in the first place, and then to emphasize the new aspects of neural state-space modeling.

4.1. Input-Output Modeling

The input-output modeling approach for neural networks has been partly introduced and illustrated in [Narendra and Partasarathy 1990, Narendra and Partasarathy 1991, Nerrand et al. 1994]. In this subsection, we summarize its basic principles.

The NARMAX assumed model.

The most general input-output model whose associated predictor can be easily expressed is the NARMAX model, which is an extension of the linear ARMAX model to the nonlinear case [Leontaritis and Billings 1985, Chen and Billings 1989]. It is given by:

$$y_p(k) = h\left(y_p(k\text{-}1), ..., y_p(k\text{-}n), u(k\text{-}1), \ldots, u(k\text{-}m), w(k\text{-}1), \ldots, w(k\text{-}p)\right) + w(k) \tag{4.1.1}$$

Let us consider the following theoretical feedback predictor of order p (its state variables are the p past prediction errors $e = y_p - y$):

$$y(k+1) = h\left(y_p(k), ..., y_p(k\text{-}n+1), u(k), \ldots, u(k\text{-}m+1), e(k), \ldots, e(k\text{-}p+1)\right) \tag{4.1.2}$$

If the assumed model is true, and if the p first prediction errors are equal to the noise, $e(k+1) = w(k+1)\ \forall\ k$; therefore, the variance of the prediction error e is minimal. The effect of arbitrary initial conditions will vanish depending on the unknown function h. We will consider (4.1.2) as the theoretical predictor associated to the

assumed model (4.1.1). Note that the function h_{pred} defining the associated predictor is the function h of the assumed model.

In order to implement the associated predictor, one must therefore train a feedback neural network of the form:

$$y(k+1) = \phi\left(y_p(k), ..., y_p(k-n+1), u(k), \ldots, u(k-m+1), e(k), \ldots, e(k-p+1); \theta\right) \tag{4.1.3}$$

where ϕ is the nonlinear function implemented by the feedforward part of the network with weights θ. If the assumed model is true, and if ϕ approximates h with an arbitrary precision, then predictor (4.1.3) is arbitrarily close to the asymptotically optimal one.

As in the linear case, it is usual in nonlinear modeling to first consider particular cases of the NARMAX model leading to more simple predictors.

The NARMAX assumed model with additive correlated (ARMA) noise.

This model of intermediate complexity is discussed in [Goodwin and Sin 1984]:

$$A(q^{-1})\, y_p(k) = h\left(y_p(k-1), \ldots, y_p(k-n), u(k-1), \ldots, u(k-m)\right) + C(q^{-1})\, w(k) \tag{4.1.4}$$

where A and C are monic polynomials with degree n and p respectively, and h is a nonlinear function. The theoretical predictor associated to (4.1.4) is given by:

$$\begin{aligned} y(k+1) = {} & h\left(y_p(k), \ldots, y_p(k-n+1), u(k), \ldots, u(k-m+1)\right) \\ & + \left(1 - A(q^{-1})\right) y_p(k+1) + \left(C(q^{-1}) - 1\right) e(k+1) \end{aligned} \tag{4.1.5}$$

If C has no zeros outside or on the unit circle, the effect of arbitrary initial conditions will die away.

The neural network used for training thus imposes a linear dependency between its output and the past prediction errors:

$$y(k+1) = \phi\left(y_p(k), \ldots, y_p(k-n+1), u(k), \ldots, u(k-m+1); \theta\right) + \Pi\,(q^{-1})\, e(k+1) \tag{4.1.6}$$

where Π is a polynomial with adjustable coefficients.

The NARX assumed model.

The NARX model assumes *pseudo-white additive state noise*, and is given by:

$$y_p(k) = h\left(y_p(k-1), ..., y_p(k-n), u(k-1), \ldots, u(k-m)\right) + w(k) \tag{4.1.7}$$

The associated theoretical predictor is feedforward:

$$y(k+1) = h\left(y_p(k), ..., y_p(k-n+1), u(k), \ldots, u(k-m+1)\right) \tag{4.1.8}$$

If the assumed model is true, the prediction error $e(k+1) = y_p(k+1) - y(k+1)$ is equal to the noise $w(k+1)$, and its variance is minimal.

Thus, the neural predictor must be made of a feedforward network:

$$y(k+1) = \phi\left(y_p(k), ..., y_p(k-n+1), u(k), \ldots, u(k-m+1); \theta\right) \tag{4.1.9}$$

The Output-Error assumed model.

As in the linear case, the Output-Error model assumes *pseudo-white additive output noise*. It is given by:

$$\begin{cases} z_p(k) = h\left(z_p(k\text{-}1), \ldots, z_p(k\text{-}n), u(k\text{-}1), \ldots, u(k\text{-}m)\right) \\ y_p(k) = z_p(k) + w(k) \end{cases} \tag{4.1.10}$$

Let us consider the following theoretical feedback predictor of order n:

$$y(k+1) = h\left(y(k), \ldots, y(k\text{-}n+1), u(k), \ldots, u(k\text{-}m+1)\right) \tag{4.1.11}$$

If the assumed model is true, and if the n first prediction errors are equal to the noise, then the prediction error is also equal to the noise and the prediction error variance is minimal. The effect of arbitrary initial conditions will vanish depending on the unknown function h.

In order to implement the associated predictor (4.1.11), one must train a feedback neural network of the form:

$$y(k+1) = \phi\left(y(k), \ldots, y(k\text{-}n+1), u(k), \ldots, u(k\text{-}m+1); \theta\right) \tag{4.1.12}$$

4.2. State-Space Modeling

State-space models can be used either as *knowledge-based* models if enough prior knowledge on the physics of the process is available, or as *black-box* models when input-output models prove to be inefficient; we are concerned with the latter case [for a presentation including knowledge-based models, see Rivals et al. 1995].

The specificity of black-box state-space modeling arises from the fact that only the outputs of the state-space predictors have desired values $\{y_p(k)\}$, while the internal state variables are not imposed. This gives state-space predictors more flexibility (i.e. the ability to have a more complex input-output behavior), but the price to be paid is that different state-space representations might display the same input-output behavior. To present this specificity, we start with deterministic state-space assumed models; we then consider stochastic models.

4.2.1. Determistic State-Space Models

The assumed model is:

$$\begin{cases} x_p(k+1) = f\left(x_p(k), u(k)\right) \\ y_p(k) = g\left(x_p(k)\right) \end{cases} \tag{4.2.1.1}$$

where x_p denotes the n-state vector of the assumed model. Consider the predictor:

$$\begin{cases} x(k+1) = f\left(x(k), u(k)\right) \\ y(k+1) = g\left(x(k+1)\right) \end{cases} \tag{4.2.1.2}$$

where the n-vector x is the prediction of the state x_p. If the assumed model is true, and if $x(k) = x_p(k)$, the prediction error $e(k+1) = y_p(k+1) - y(k+1)$ is zero.

But, consider any predictor with n-state vector $\tilde{x} = \Phi(x)$, where Φ is invertible, and such that:

$$\begin{cases} \tilde{x}(k+1) = \tilde{f}(\tilde{x}(k),\, u(k)) \\ \tilde{y}(k+1) = \tilde{g}(\tilde{x}(k+1)) = y(k+1) \end{cases} \text{ with } \begin{cases} \tilde{f}(.\ ,\ .) = \Phi\left(f\left(\Phi^{-1}(.),\ .\right)\right) \\ \tilde{g}(.) = g\ \left(\Phi^{-1}(.)\right) \end{cases} \tag{4.2.1.3}$$

Any such predictor has the same input-output behavior as predictor (4.2.1.2).

Thus, if the following state-space neural predictor, with n-state vector ξ and output y, is trained:

$$\begin{cases} \xi(k+1) = \phi(\xi(k),\, u(k);\, \theta_\phi) \\ y(k+1) = \omega(\xi(k+1);\, \theta_\omega) \end{cases} \tag{4.2.1.4}$$

where ϕ and ω are nonlinear functions implemented by cascaded subnetworks with weights θ_ϕ and θ_ω, the training might lead to any of the predictors defined by (4.2.1.3). Thus, its fonctions ϕ and ω need not be approximations of the functions f and g of the assumed model. Nevertheless, since the goal of black-box modeling is to construct a predictor that displays an input-output behavior as close as possible to the process behavior, any such predictor is satisfactory.

Remark.

In practice, the following state-space network can be used instead of (4.2.1.4):

$$\begin{cases} \xi(k+1) = \phi(\xi(k),\, u(k);\, \theta_\phi) \\ y(k+1) = \psi(\xi(k),\, u(k);\, \theta_\psi) \end{cases} \tag{4.2.1.5}$$

The predictors (4.2.1.4) and (4.2.1.5) will have the same input-output behavior if $\psi = \omega\ o\ \phi$. The advantage of (4.2.1.5) is that, ϕ and ψ being functions of the same arguments, they can be implemented by a single network with weights θ. For convenience, such networks are considered throughout sections 6 and 7.

4.2.2. Stochastic State-Space Models

General nonlinear assumed model.

Both state and output noise are assumed to be present, i.e. the assumed model is given by:

$$\begin{cases} x_p(k+1) = f(x_p(k),\, u(k),\, v_1(k)) \\ y_p(k) = g(x_p(k),\, v_2(k)) \end{cases} \tag{4.2.2.1}$$

where $\{v_1(k)\}$ is a sequence of zero mean, i.i.d. random vectors, and $\{v_2(k)\}$ is a sequence of zero mean, i.i.d. random variables.

If the functions f and g are known, the extended Kalman predictor gives a *suboptimal* solution using a linearization of the model around the current state estimate. The one-step-ahead predictions of the state and of the output are computed using the following recursion:

$$\begin{cases} x(k+1) = f(x(k),\, u(k),\, 0) + K(k,\, x(k))\,(y_p(k) - g(x(k),\, 0)) \\ y(k+1) = g(x(k+1),\, 0) \end{cases} \tag{4.2.2.2}$$

where $K(k, x(k))$ is the time-varying and state-dependent extended Kalman gain, which is computed using the linearizations of f and g [Goodwin and Sin 1984]. But, in the modeling problem, the functions f and g are partially or completely unknown. There is thus no reason to restrict the complexity of the associated predictor structure to the form (4.2.2.2). We propose a time-invariant associated predictor with the same arguments, but of the more general form:

$$\begin{cases} x(k+1) = f_{pred}(x(k),\, u(k),\, y_p(k)) \\ y(k+1) = g_{pred}(x(k+1)) = g(x(k+1),\, 0) \end{cases} \tag{4.2.2.3}$$

A state-space neural predictor corresponding to predictor (4.2.2.3):

$$\begin{cases} \xi(k+1) = \phi(\xi(k),\, u(k),\, y_p(k);\, \theta_\phi) \\ y(k+1) = \omega(\xi(k+1);\, \theta_\omega) \end{cases} \tag{4.2.2.4}$$

with n-state vector ξ will be trained to minimize the MSPE on the training set.

Remark.

As in the deterministic case, the following state-space network can also be used:

$$\begin{cases} \xi(k+1) = \phi(\xi(k),\, u(k),\, y_p(k);\, \theta) \\ y(k+1) = \psi(\xi(k),\, u(k),\, y_p(k);\, \theta) \end{cases} \tag{4.2.2.5}$$

where ϕ and ψ are nonlinear functions implemented by a single feedforward network with weights θ.

As in the input-output case, one would like to consider particular cases of the general model that lead to less complex predictors. This can be done essentially when making the assumption of additive output noise:

"Additive Output Noise" state-space assumed model.

In the case where additive output noise only is assumed:

$$\begin{cases} x_p(k+1) = f(x_p(k),\, u(k)) \\ y_p(k) = g(x_p(k)) + v_2(k) \end{cases} \tag{4.2.2.6}$$

an associated predictor takes a more simple form (it does not use the observations $\{y_p(k)\}$):

$$\begin{cases} x(k+1) = f(x(k),\, u(k)) \\ y(k+1) = g(x(k+1)) \end{cases} \tag{4.2.2.7}$$

If the assumed model is true, and if it is stable, the prediction error of predictor (4.2.2.7) is asymptotically equal to the noise.

The following state-space neural predictor will be trained:

$$\begin{cases} \xi(k+1) = \phi\,(\xi(k),\, u(k);\, \theta_\phi) \\ y(k+1) = \omega\,(\xi(k+1);\, \theta_\omega) \end{cases} \qquad (4.2.2.8)$$

where ξ is the n-state vector of the predictor, and where ϕ and ω are nonlinear functions implemented by cascaded subnetworks with weights θ_ϕ and θ_ω .

Remark.

Again, the following state-space neural network can also be used:

$$\begin{cases} \xi(k+1) = \phi\,(\xi(k),\, u(k);\, \theta) \\ y(k+1) = \psi\,(\xi(k),\, u(k);\, \theta) \end{cases} \qquad (4.2.2.9)$$

where ϕ and ψ are nonlinear functions implemented by a single feedforward network with weights θ.

5. Training of the Neural Predictors

Let us consider a set of candidate neural predictors, each candidate being characterized by the assumed model it is associated to, and by its architecture (connectivity and number of neurons). The goal is now to obtain the best neural predictor among the candidates. As in the linear case considered in section 3.1, the training of each candidate (the estimation of its weights θ) is achieved using a P.E. method, or possibly an E.K.F. method in the case of a state-space neural network; the candidate with the best performance is then selected.

The E.K.F. training method consists in including the weights of the neural network in an augmented state vector, and to estimate this state recursively using the extended Kalman predictor associated to the augmented model. But it should be noted that the E.K.F. method requires the knowledge of the noise covariance matrices, which is unrealistic in black-box modeling. The E.K.F. algorithm might thus diverge, whereas the P.E. method will always provide the best predictor of a given arbitrary structure. As a consequence, P.E. methods are preferred for the training of neural networks. For a presentation of E.K.F. methods applied to neural network training, see [Singhal and Wu 1989, Matthews and Moschytz 90].

In the framework of a P.E. method, a candidate network is trained by minimizing its MSPE on input-output *training sequences*. Since we are concerned with time-invariant models, the MSPE on the training sequences can be minimized in a non recursive, iterative fashion ("batch" training). The best candidate is the predictor with the smallest MSPE on appropriate *test sequences*.

More precisely, the training and the selection use:
- sequences applied to the external inputs of the neural predictor $\{I(k)\}$, and sequences of the corresponding desired values $\{y_p(k+1)\}$ for the outputs (the training and test sequences);
- a cost function defined as the MSPE on the training sequence of size N, which is to be minimized iteratively. At iteration i, its value is:

$$J(\theta^i) = \frac{1}{N}\sum_{k=0}^{N-1}\left(e^i(k+1)\right)^2 = \frac{1}{N}\sum_{k=0}^{N-1}\left(y_p(k+1) - y^i(k+1)\right)^2 \qquad (5.1)$$

  where $y^i(k+1)$ is the output of the predictor at time k and iteration i.
- the MSPE on the test sequence.

Training of a neural candidate with a given architecture.

At each iteration, the computation of the cost function $J(\theta^i)$ is performed using N copies of the feedforward part of the neural predictor (see Figure 5.1). In the following, we will omit the index i of the current iteration.

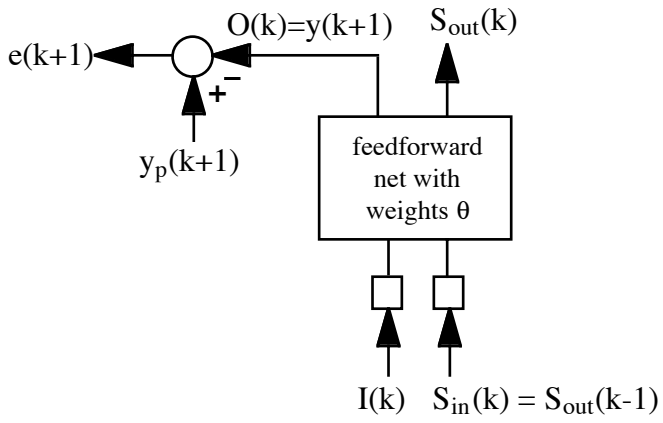

Figure 5.1.
Copy k of a trained neural predictor.

In the general case of a feedback predictor, the inputs of the copy k are:
- $I(k)$, the nonfeedback inputs of the predictor (external inputs u and possibly the measured outputs y_p);
- $S_{in}(k)$, the input state variables of the predictor at time k (past outputs y , or state ξ, or prediction errors e), which verify $S_{in}(k) = S_{out}(k\text{-}1)$ for k > 0; for copy k=0, the initial state must be fixed arbitrarily.

The outputs of the copy k are:
- $O(k)$, the output of the predictor $y(k+1)$;
- $S_{out}(k)$, the output state variables of the predictor at time $k+1$.

The cost function $J(\theta^i)$ can be minimized with any gradient method (the gradient being computed by the Backpropagation algorithm); quasi-Newtonian methods are particularly efficient. Feedforward predictors are trained in a directed fashion [Nerrand et al. 1993], i.e. using the Teacher Forcing algorithm [Jordan 1985], and feedback predictors are trained in a semi-directed fashion, i.e. using the Backpropagation-Through-Time algorithm [Rumelhart et al. 1986]. For further details about the training of state-space neural networks, see [Rivals 1995].

Example 1.

Let us make the NARX assumption for a given process; the associated predictor is given by (4.1.8). A feedforward neural predictor given by (4.1.9) will therefore be used:

$$y(k+1) = \phi\left(y_p(k), ..., y_p(k-n+1), u(k), \ldots, u(k-m+1); \theta\right) \qquad (5.2)$$

The external inputs $I(k)$ of copy k used for training are:

$$I(k) = \left[y_p(k), \ldots, y_p(k-n+1), u(k), \ldots, u(k-m+1)\right] \qquad (5.3)$$

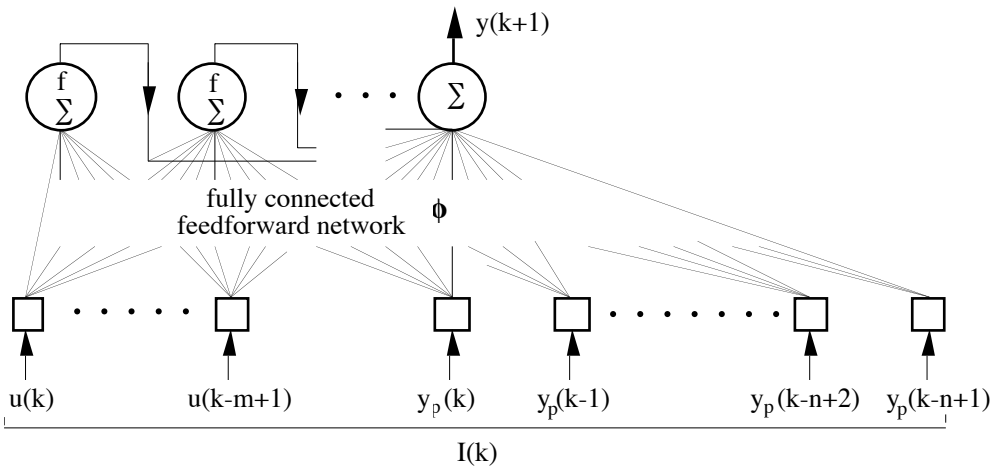

Figure 5.2.
Copy k of a neural NARX predictor.

Figure 5.2 shows a possible implementation of predictor (5.2) where the function ϕ is computed by a fully connected feedforward network with weights θ.

Example 2.

The predictor associated to a NARMAX assumed model is given by (4.1.2). A feedback neural predictor given by (4.1.3) must therefore be used:

$$y(k+1) = \phi\left(y_p(k), ..., y_p(k-n+1), u(k), \ldots, u(k-m+1), e(k), \ldots, e(k-p+1); \theta\right) \qquad (5.4)$$

The external and state inputs, and the state outputs of copy k, are:

$$\begin{aligned} I(k) &= [y_p(k), \ldots, y_p(k\text{-}n\text{+}1), u(k), \ldots, u(k\text{-}m\text{+}1)] \\ S_{in}(k) &= S_{out}(k\text{-}1) = [e(k), \ldots, e(k\text{-}p\text{+}1)] \end{aligned} \tag{5.5}$$

The initial state can for instance be taken equal to: $S_{in}(0) = [0, \ldots, 0]$. Figure 5.3 shows a possible implementation of predictor (5.4), the function ϕ being computed by a fully connected feedforward network with weights θ.

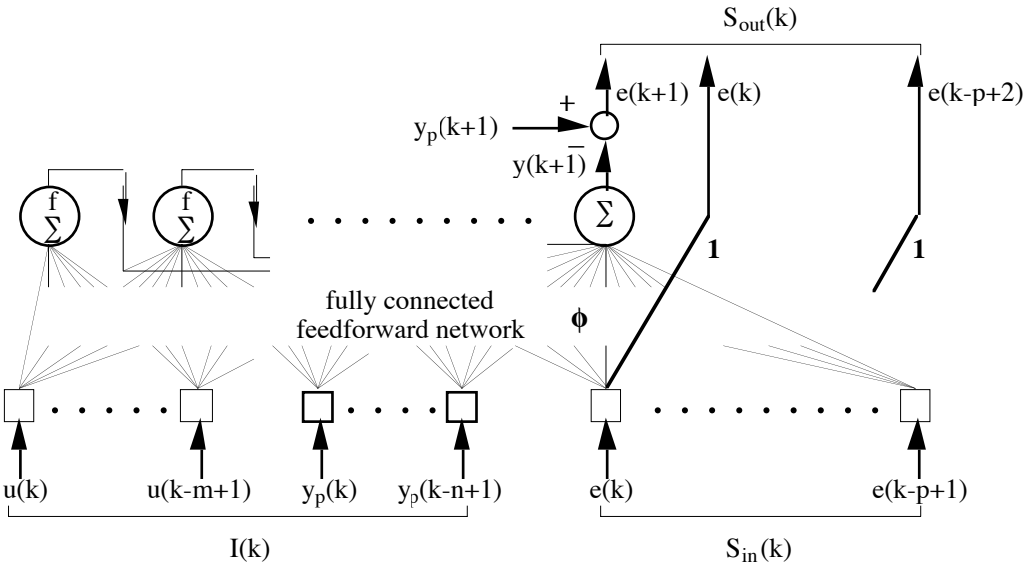

Figure 5.3.
Copy k of a neural NARMAX predictor.

Example 3.

A predictor associated to a stochastic state-space assumed model of order n is given by (4.2.2.3). A state-space neural network is used, possibly (4.2.2.4):

$$\begin{cases} \xi_1(k\text{+}1) = \phi_1\,(\xi_1(k), \ldots, \xi_n(k), u(k), y_p(k); \theta_{\phi_1}) \\ \quad\ldots \\ \xi_n(k\text{+}1) = \phi_n\,(\xi_1(k), \ldots, \xi_n(k), u(k), y_p(k); \theta_{\phi_n}) \\ y(k\text{+}1) = \omega\,(\xi_1(k\text{+}1), \ldots, \xi_n(k\text{+}1); \theta_\omega) \end{cases} \tag{5.6}$$

The external and state inputs, and the state outputs of copy k, are:

$$\begin{aligned} I(k) &= [u(k), y_p(k)] \\ S_{in}(k) &= S_{out}(k\text{-}1) = [\xi_1(k), \ldots, \xi_n(k)] \end{aligned} \tag{5.7}$$

The state may be initialized with: $S_{in}(0) = [0, \ldots, 0]$.
(5.6) is represented in Figure 5.4. The n functions $\phi_1, \ldots, \phi_n$ and the output function ω are implemented by n+1 one-layer feedforward subnetworks.

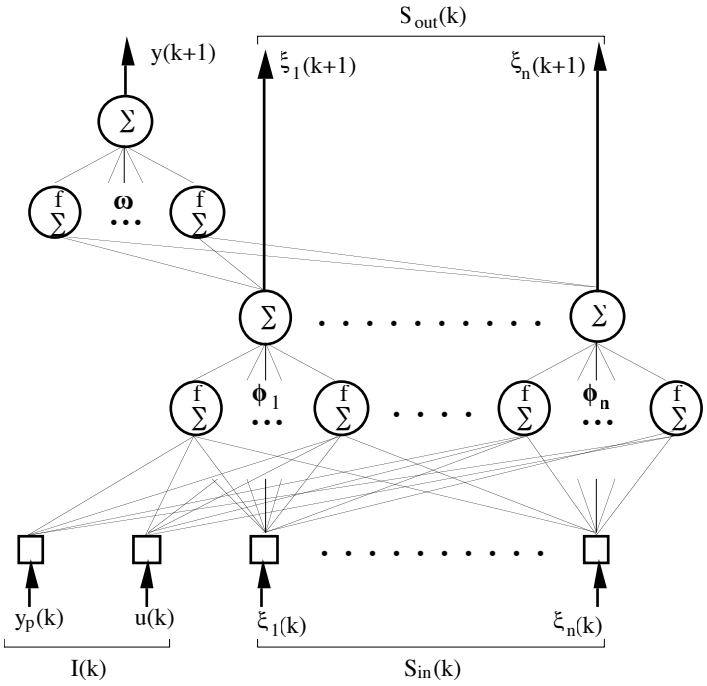

Figure 5.4.
Copy *k* of a predictor (4.2.2.4) associated to a stochastic state-space model.

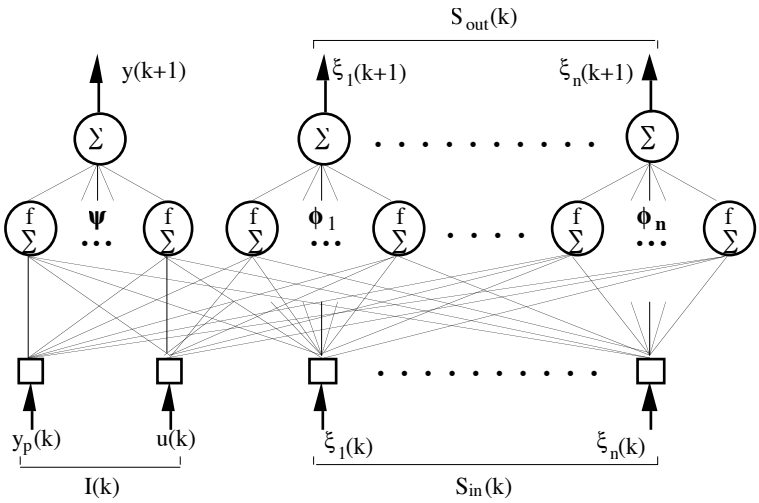

Figure 5.5.
Copy *k* of a predictor (4.2.2.5) associated to a stochastic state-space model.

Another possible neural predictor for (4.2.2.3) is of form (4.2.2.5):

$$\begin{cases} \xi_1(k+1) = \varphi_1\left(\xi_1(k),\ \ldots,\ \xi_n(k),\ u(k),\ y_p(k);\ \theta_{\varphi_1}\right) \\ \qquad \ldots \\ \xi_n(k+1) = \varphi_n\left(\xi_1(k),\ \ldots,\ \xi_n(k),\ u(k),\ y_p(k);\ \theta_{\varphi_n}\right) \\ y(k+1) = \psi\left(\xi_1(k),\ \ldots,\ \xi_n(k),\ u(k),\ y_p(k);\ \theta_{\psi}\right) \end{cases} \tag{5.8}$$

The external and state inputs, and the state outputs of copy k, are also given by (5.7). A possible implementation of predictor (5.8) is shown in Figure 5.5.

Remark.

The state-space networks proposed here are minimal state-space representations, as opposed to those used in [Elman 1990], or more recently in [Lo 1994, Connor et al. 1994], which have also been shown to be universal identification models [Sontag 1993]. In the present paper, the state variables are computed by a neural network with as many nonlinear hidden neurons as required, whereas in Elman's networks, each function ϕ_i is constrained to be computed with a *single* hidden neuron. This constraint leads generally to a non minimal state-space representation.

To summarize:

Let the training and test sequences be chosen according to the future use of the model (type of inputs, sample size, etc.): if *the assumptions underlying the selected neural predictor are true,* if *the family of functions defined by the feedforward part of the network is rich enough with respect to the complexity of the process behavior (i.e.* if *the number of neurons is sufficient), and* if *the training algorithm is efficient,* then *the trained predictor will be arbitrarily close to the optimal predictor in the state domain covered by the training and test sequences.*

6. Modeling Simulated Processes

To illustrate the neural black-box modeling approach we have outlined, we first model processes simulated by deterministic and stochastic state-space systems.

6.1. Modeling a Deterministic Process

We consider a process simulated by the deterministic second-order state-space system:

$$\begin{cases} x_{1p}(k+1) = 1.145\ x_{1p}(k) - 0.549\ x_{2p}(k) + 0.584\ u(k) \\ x_{2p}(k+1) = \dfrac{x_{1p}(k)}{1 + 0.01\ x_{2p}(k)^2} + 0.330\ u(k) \\ y_p(k) = 4\ tanh\left(\dfrac{x_{1p}(k)}{4}\right) \end{cases} \tag{6.1.1}$$

The behavior of this process is oscillatory with unity static gain around zero; the denominator of the second state equation, together with the *tanh* in the output equation, increases the damping and decreases the static gain in the large-amplitude regime. The training and test input sequences consist of steps of random amplitude in [-5, +5] with a duration of 20 sampling periods; the total length of the sequences is

1200. Figure 6.1 shows the first third of the test sequence (for clarity, the graphs will always display the first third of the sequences only).

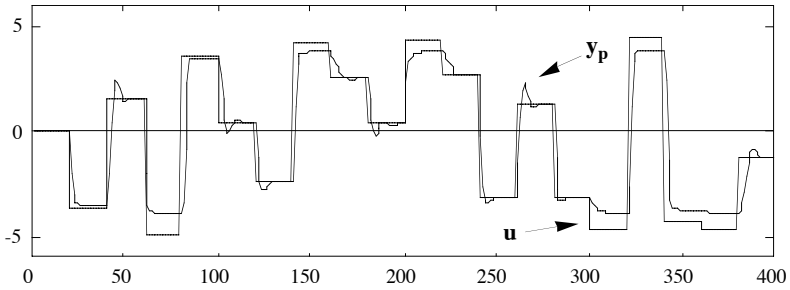

Figure 6.1.
Test sequences.

A predictor associated to the true model is given by Eq. (4.2.1.2). We use a second-order fully connected network of form (4.2.1.5):

$$\begin{cases} \xi_1(k+1) = \varphi_1\left(\xi_1(k),\, \xi_2(k),\, u(k);\, \theta\right) \\ \xi_2(k+1) = \varphi_2\left(\xi_1(k),\, \xi_2(k),\, u(k);\, \theta\right) \\ y(k+1) = \psi\left(\xi_1(k),\, \xi_2(k),\, u(k);\, \theta\right) \end{cases} \tag{6.1.2}$$

With only 2 hidden neurons and 27 weights (see Figure 6.2), the MSPE equals 1.6×10^{-7} on the training sequence, and 3.8×10^{-7} on the test sequence.

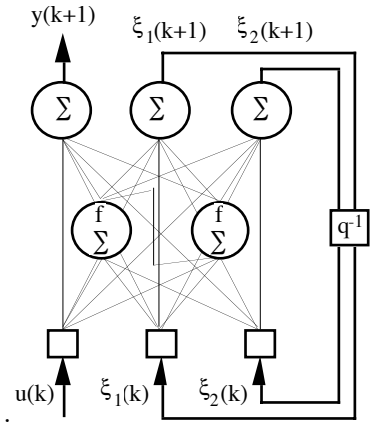

Figure 6.2.
Neural state-space predictor associated to deterministic or to "Additive Output Noise" state-space assumed models (*f* is the *tanh* function).

Figure 6.3a shows the outputs of the process and of the neural predictor; the prediction error is displayed on Figure 6.3b. Figures 6.3c and 6.3d show the state variables of the process and of the model. As expected, there is no straightforward relationship between them.

In : "Neural Adaptive Control Technology", R. Zbikowski and K. J. Hunt eds., World Scientific (1996), pp. 237-264.

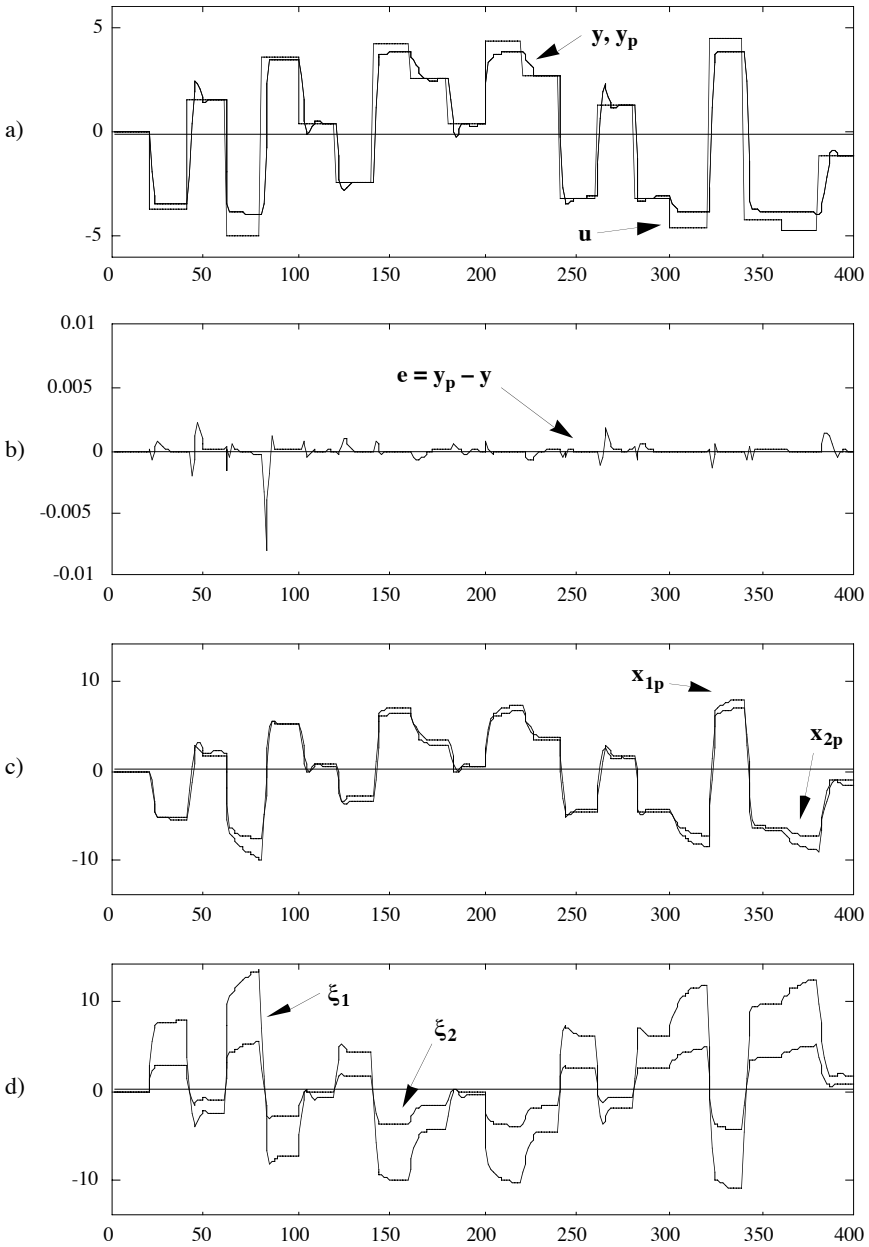

Figure 6.3.
State-space modeling of a deterministic process.

Input-output candidate predictors of the following form were also trained:

$$y(k+1) = \varphi\left(y(k),\ \ldots,\ y(k-n+1),\ u(k),\ \ldots,\ u(k-m+1);\ \theta\right) \qquad (6.1.3)$$

For n=m=2, 10 hidden neurons are necessary to obtain a MSPE of 10^{-4} on the test sequence; higher values of n and m do not lead to better results.

The comparison of the complexity and of the performances of the neural input-output and state-space predictors shows the difficulty that may arise when modeling an unknown process with an input-output predictor.

6.2. Modeling Stochastic Processes

Let $\{v_1(k)\}$ and $\{v_2(k)\}$ be gaussian white noise sequences with zero mean and variance $8{,}3\ 10^{-2}$. The input sequences $\{u(k)\}$ for training and testing are the same as without noise. Since the predictors presented in section 3 and 4 are only *asymptotically* optimal, and since, for general stochastic models, their error variance is state-dependent, the error variance on the test sequence will be estimated using 100 realizations of the noise.

6.2.1. Modeling a Process with Additive Output Noise

The process is simulated by the following stochastic system:

$$\begin{cases} x_{1p}(k+1) = 1.145\ x_{1p}(k) - 0.549\ x_{2p}(k) + 0.584\ u(k) \\ x_{2p}(k+1) = \dfrac{x_{1p}(k)}{1 + 0.01\ x_{2p}(k)^2} + 0.330\ u(k) \\ y_p(k) = 4\ tanh\left(\dfrac{x_{1p}(k)}{4}\right) + v_2(k) \end{cases} \qquad (6.2.1.1)$$

A predictor associated to the true model is given by (4.2.2.7). Network (6.1.2) was thus trained to be a predictor for process (6.2.1.1). The best performance (i.e. the smallest MSPE on the test set) is obtained with 2 hidden neurons (27 weights). The MSPE equals $8.2\text{x}10^{-2}$ on the training sequence, and $8.4\text{x}10^{-2}$ on the test sequence, while the variance of $v_2(k)$ is $8.3\text{x}10^{-2}$. The variance of the prediction error on the test, estimated with 100 realizations of the noise, is shown on Figure 6.4. In the steady state, the variance estimation is $8.3\text{x}10^{-2}$, which is equal to the noise variance.

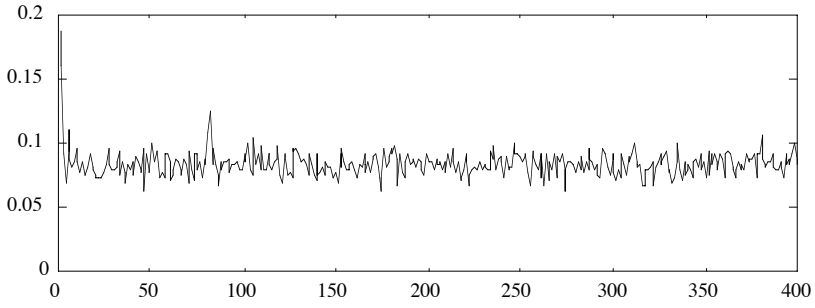

Figure 6.4.
Prediction error variance of a neural state-space predictor associated to an "Additive Output Noise" state-space assumed model.

In : "Neural Adaptive Control Technology", R. Zbikowski and K. J. Hunt eds., World Scientific (1996), pp. 237-264.

6.2.2. Modeling a Process with Additive State and Output Noise

The simulated process has also additive noise on the first state variable:

$$\begin{cases} x_{1p}(k+1) = 1.145\ x_{1p}(k) - 0.549\ x_{2p}(k) + 0.584\ u(k) + v_1(k) \\ x_{2p}(k+1) = \dfrac{x_{1p}(k)}{1 + 0.01\ x_{2p}(k)^2} + 0.330\ u(k) \\ y_p(k) = 4\ tanh\left(\dfrac{x_{1p}(k)}{4}\right) + v_2(k) \end{cases} \qquad (6.2.2.1)$$

Modeling with a predictor associated to a stochastic state-space assumed model.

As shown in section 4.2.2, the theoretical predictor (4.2.2.3) will lead to minimum variance prediction. The best MSPE on the test is obtained with a second-order fully connected network of form (4.2.2.5):

$$\begin{cases} \xi_1(k+1) = \varphi_1\left(\xi_1(k),\ \xi_2(k),\ u(k),\ y_p(k);\ \theta\right) \\ \xi_2(k+1) = \varphi_2\left(\xi_1(k),\ \xi_2(k),\ u(k),\ y_p(k);\ \theta\right) \\ y(k+1) = \psi\left(\xi_1(k),\ \xi_2(k),\ u(k),\ y_p(k);\ \theta\right) \end{cases} \qquad (6.2.2.2)$$

with 2 hidden neurons (32 weights), which is shown on Figure 6.5.

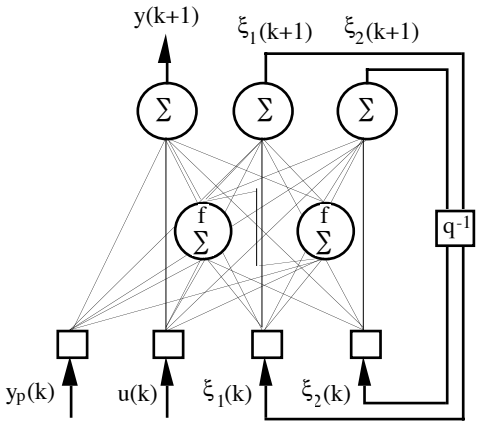

Figure 6.5.
Neural state-space predictor associated to a stochastic state-space assumed model.

The variance of the prediction error on the test sequence, estimated with 100 realizations of the noise, is shown on Figure 6.6. As opposed to the case of additive output noise, the variance is state-dependent. The MSPE equals 1.2×10^{-1} on the training sequence and 1.3×10^{-1} on the test sequence.

The value of the minimal variance is unknown; nevertheless, in order to evaluate the optimality of the neural predictor, we computed the MSPE of the extended Kalman predictor using the exact model (6.2.1.1): it is equal to 1.4×10^{-1}. This demonstrates the ability of the trained state-space neural predictor (*without*

knowledge of the exact model of the process) to perform at least as well as the extended Kalman predictor.

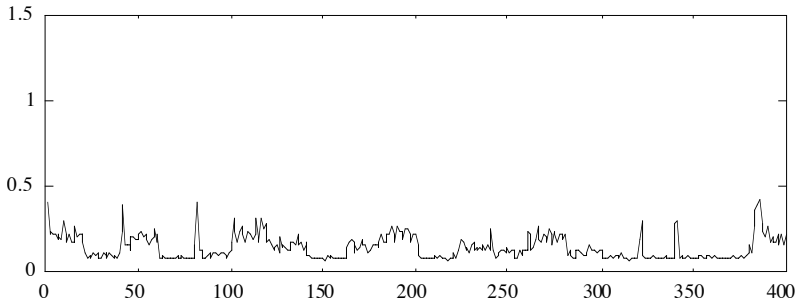

Figure 6.6.
Prediction error variance of a neural state-space predictor associated to a stochastic state-space assumed model.

Modeling with a NARMAX predictor.

We also tried to model the process (6.2.2.1) with neural NARMAX predictors, with n=m=2, of order p=1:

$$y(k+1) = \varphi\left(y_p(k),\, y_p(k-1),\, u(k),\, u(k-1),\, e(k);\, \theta\right) \qquad (6.2.2.3)$$

and order p=2:

$$y(k+1) = \varphi\left(y_p(k),\, y_p(k-1),\, u(k),\, u(k-1),\, e(k),\, e(k-1);\, \theta\right) \qquad (6.2.2.4)$$

The best results were obtained with a NARMAX predictor of order 2 with 5 hidden neurons (51 weights). The MSPE equals $1.5\text{x}10^{-1}$ on the training sequence, and $1.6\text{x}10^{-1}$ on the test sequence. The prediction error variance on the test sequence, estimated with 100 realizations of the noise, is shown on Figure 6.7.

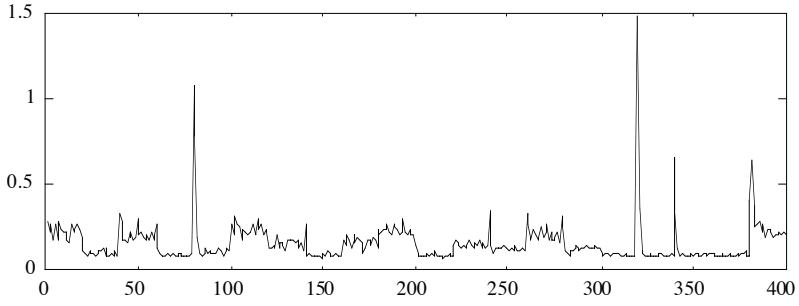

Figure 6.7.
Prediction error variance of a neural input-output predictor associated to a NARMAX assumed model.

This result confirms the fact that a NARMAX predictor might require many more arguments, neurons and weights than a state-space predictor to achieve a comparable (though not as good) performance.

In : "Neural Adaptive Control Technology", R. Zbikowski and K. J. Hunt eds., World Scientific (1996), pp. 237-264.

7. Modeling a Hydraulic Actuator

We consider the hydraulic actuator of a robot arm, whose position depends on the valve-controlled oil pressure. The available input (valve position u) and output (oil pressure y_p) sequences were split in two for training and testing, and are shown on Figure 7.1. The training and test sequences have 512 samples each.

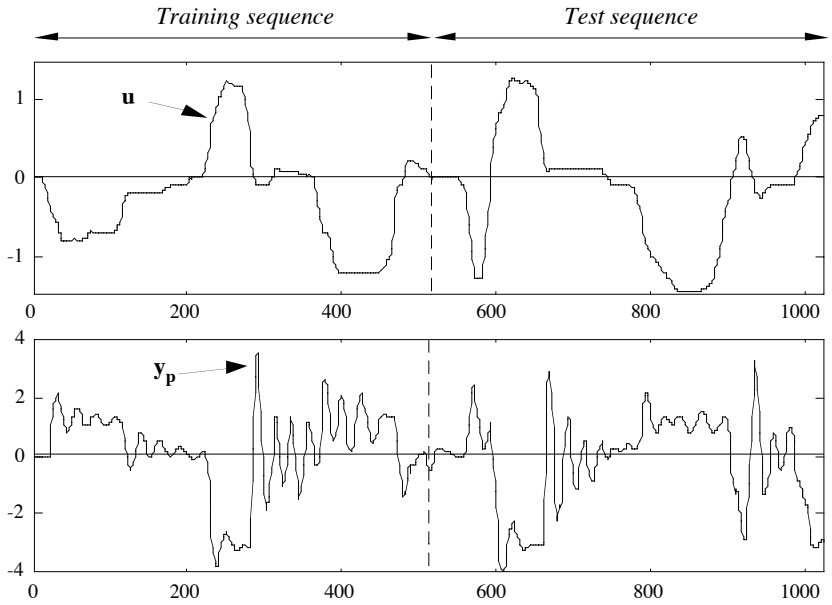

Figure 7.1.
Training and test sequences for the modeling of the hydraulic actuator.

The control and output sequences have roughly the same amplitude in the training sequence as in the test sequence, but the maximum amplitude of the control sequence is larger in the latter than in the former. Furthermore, the response of the system seems to undergo more sustained oscillations, for similar control sequences, in the training sequence than in the test sequence.

The goal of the modeling procedure is here to design a simulator of the process. The simulation model corresponding to a given predictor is a feedback model, which is obtained by replacing the process output y_p (if any) acting as external input on the predictor by the predictor output y. The performance (the MSPE on the test sequences) will be estimated with the simulation models corresponding to the candidate predictors used for training.

In : "Neural Adaptive Control Technology", R. Zbikowski and K. J. Hunt eds., World Scientific (1996), pp. 237-264.

7.1. Input-Output Modeling of the Actuator

Under the Output-Error assumption, the input-output predictor giving the best performance without overfitting is a fully connected second-order network:

$$y(k+1) = \varphi\left(y(k),\, y(k\text{-}1),\, u(k);\, \theta\right) \qquad (7.1.1)$$

with 2 hidden neurons (15 weights), which is shown on Figure 7.2.

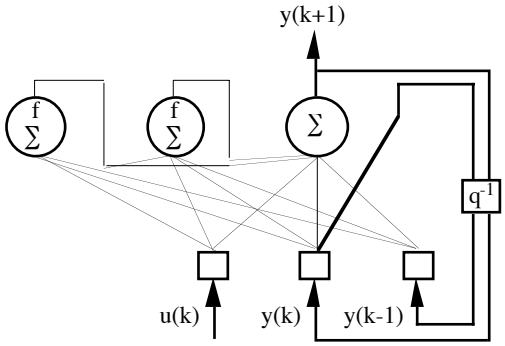

Figure 7.2.
Input-output neural predictor associated to an Output-Error assumed model.

Since the predictor (7.1.1) does not use the output of the process y_p, the corresponding simulation model is identical to the predictor. The result obtained on the test sequence is shown on Figure 7.3. The MSPE are 0.085 on the training sequence and 0.30 on the test sequence. Adding hidden neurons or extra inputs leads to overfitting.

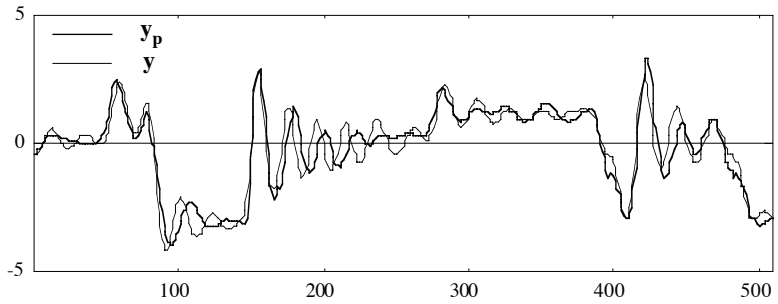

Figure 7.3.
Input-output modeling of the actuator (Output-Error assumption).

Making the NARX assumption leads to lower quality results (the NARX predictors being tested as simulation models, i.e. as feedback models). NARMAX predictors do not give better results than Output-Error predictors.

7.2. State-Space Modeling of the Actuator

Under the "Additive Output Noise" assumption, the best results were obtained with a second-order network:

$$\begin{cases} \xi_1(k+1) = \varphi_1\left(\xi_1(k),\, \xi_2(k),\, u(k);\, \theta\right) \\ \xi_2(k+1) = \varphi_2\left(\xi_1(k),\, \xi_2(k),\, u(k);\, \theta\right) \\ y(k+1) = \psi\left(\xi_1(k),\, \xi_2(k),\, u(k);\, \theta\right) \end{cases} \tag{7.2.1}$$

with 2 hidden neurons (27 weights), which is the same network as in Figure 6.2.

Again, since the predictor (7.2.1) does not use the output of the process y_p, the corresponding simulation model is identical to the predictor. The result on the test sequence is shown on Figure 7.4. The MSPE is 0.10 on the training sequence, and 0.11 on the test sequence, which is much better than under input-output assumptions. This clearly illustrates the advantage of using state-space predictors when small training sets only are available: since they require a smaller number of inputs, they are more parsimonious, hence less prone to overfitting.

A neural state-space predictor associated to a general stochastic assumed model, i.e. given by (4.2.2.4) or (4.2.2.5), did not give a better performance.

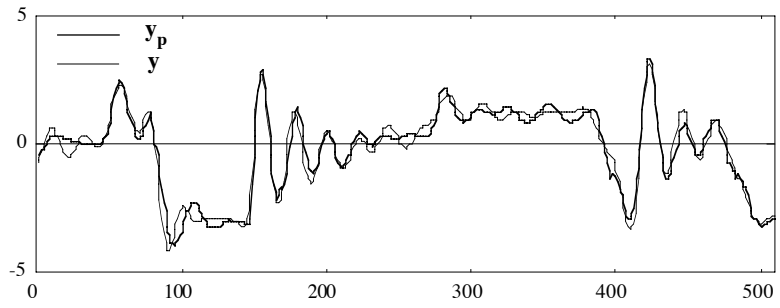

Figure 7.4.
State-space modeling of the actuator (state-space "Additive Output Noise" assumption).

Nevertheless, the results presented above compare very favorably to those obtained on the same data by other groups: in [Sjöberg 1993] a MSPE on the test sequence of 0.46 is obtained with a NARX second-order input-output multilayer Perceptron with 8 hidden neurons; similar results are obtained using wavelets in [Benveniste et al. 1994].

8. Conclusion

We have given an overview of the neural network approach to nonlinear modeling, from the choice of the candidate models to the estimation of the parameters of their corresponding neural predictors. We have emphasized the importance of

state-space candidate models, which are likely to need less arguments than input-output models: this is clearly an advantage when small data sets only are available. The trained state-space neural predictors were shown to be able to perform at least as well as the extended Kalman predictor, but without the knowledge of a model of the process. The modeling approach has been illustrated on several simulated examples and on a real process.

In the case where an accurate nonlinear model of the process is available, another possible application of neural state-space predictors is the synthesis of an optimal filter using training sequences generated by computer simulations with the model: a trained neural predictor is likely to outperform the extended Kalman filter, or even the iterated extended Kalman filter, as shown in [Lo 1994].

An additional attractive feature of state-space neural modeling is the fact that it is well suited to the introduction of domain knowledge, when the latter is in the form of state equations; this is illustrated in [Ploix et al. 1994] by the modeling of a large industrial process, a distillation column, described by more than one hundred state equations.

9. Acknowledgements

The authors are deeply indebted to Pierre-Yves Glorennec for providing the data on the hydraulic actuator.

10. References

Benveniste A., Juditsky A., Delyon B., Zhang Q., Glorennec P.-Y. Wavelets in identification, *10th IFAC Symposium on Identification* (Copenhagen, 1994).

Chen S., Billings S. A. Recursive prediction error parameter estimation for non linear models, *Int. J. Control* **49** (1989), pp. 569-594.

Connor J. T., Martin R. D., Atlas L. E. Recurrent neural networks and robust time series prediction, *IEEE Trans. on Neural Networks* **5** (1994), pp. 240-254.

Elman J. L. Finding structure in time, *Cognitive Science* **14** (1990), pp. 179-221.

Goodwin G. C., Sin K. S. *Adaptive filtering prediction and control* (Prentice-Hall, New Jersey, 1984).

Hornik K., Stinchcombe M., White H. Multilayer feedforward networks are universal approximators, *Neural Networks* **2** (1989), pp. 359-366.

Hornik. K., Stinchcombe M., White H., Auer P. Degree of approximation results for feedforward networks approximating unknown mappings and their derivatives, *Neural Computation* **6** (1994), pp. 1262-1275.

Jordan M. I. *The learning of representations for sequential performance*, Doctoral Dissertation, University of California, San Diego (1985).

Leontaritis I. J., Billings S. A. Input-output parametric models for non-linear systems. Part I: deterministic non-linear systems. Part II: stochastic non-linear systems, *Int. J. Control* **41** (1985), pp. 303-344.

Levin A. U. *Neural networks in dynamical systems; a system theoretic approach*, PhD Thesis, Yale University (1992).

Levin A. U., Narendra K. S. Identification using feedforward networks, *Neural Computation* **7** (1995), pp. 349-357.

Lo J. T.-H. Synthetic approach in optimal filtering, *IEEE Trans. on Neural Networks* **5** (1994), pp. 803-811.

Matthews M. B., Moschytz G. S. Neural-network nonlinear adaptive filtering using the extended Kalman filter algorithm, *International Neural Network Conference*, (Paris, 1990), pp. 115-118.

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

Narendra K. S., Parthasarathy K. Gradient methods for the optimization of dynamical systems containing neural networks, *IEEE Trans. on Neural Networks* **2** (1991), pp. 252-262.

Nerrand O., Roussel-Ragot P., Personnaz L., Dreyfus G. Neural networks and non-linear adaptive filtering: unifying concepts and new algorithms, *Neural Computation* **5** (1993), pp. 165-199.

Nerrand O., Roussel-Ragot P., Urbani D., Personnaz L., Dreyfus G. Training recurrent neural networks: why and how? An Illustration in Process Modeling, *IEEE Trans. on Neural Networks* **5** (1994), pp. 178-184.

Ploix J.-L., Dreyfus G., Corriou J.-P., Pascal D. From knowledge-based models to recurrent networks: an application to an industrial distillation process, *Neural Networks and their Applications*, J. Hérault ed.(1994).

Rivals I., Canas D., Personnaz L., Dreyfus G. Modeling and control of mobile robots and intelligent vehicles by neural networks, *IEEE Conference on Intelligent Vehicles* (Paris, 1994), pp. 137-142.

Rivals I. *Modélisation et commande de processus par réseaux de neurones; application au pilotage d'un véhicule autonome*, Thèse de Doctorat de l'Université Paris 6 (1995).

Rivals I., Personnaz L., Dreyfus G., Ploix J.-L. Modélisation, classification et commande par réseaux de neurones: principes fondamentaux, méthodologie de conception, et illustrations industrielles, in *Récents progrès en génie des procédés* **9** (Lavoisier technique et documentation, Paris, 1995).

Rumelhart D. E., Hinton G. E., Williams R. J. Learning internal representations by error back-propagation, in *Parallel Distributed Processing: explorations in the microstructure of cognition. Vol.1 : Foundations*, D. E. Rumelhart, J. L.

McLelland and the PDP Research Group eds. (MIT Press, Cambridge MA, 1986), pp. 318-362.

Singhal S., Wu L. Training multilayer perceptrons with the extended Kalman algorithm, *Advances in Neural Information Processing Systems* **I** (Morgan Kaufmann, San Mateo 1989), pp. 133-140.

Sjöberg J. *Regularization issues in neural network models of dynamical systems*, LiU-TEK-LIC-1993:08, ISBN 91-7871-072-3, ISSN 0280-7971 (1993).

Sjöberg J., Zhang Q., Benveniste A., Deylon B., Glorennec P.-Y., Hjalmarsson H., Juditsky A., Ljung L. Nonlinear black-box modelling in system identification: model structures and algorithms, submitted to *Automatica* (1995).

Sontag E. D. Neural networks for control, in *Essays on control: perspectives in the theory and its applications*, Trentelman H. L. and Willems J. C. eds. (Birkhäuser , Boston 1993), pp. 339-380.