*UROPEAN CONTROL CONFERENCE (1993)*

# NON-LINEAR RECURSIVE IDENTIFICATION AND CONTROL BY NEURAL NETWORKS: A GENERAL FRAMEWORK

O. NERRAND, L. PERSONNAZ, G. DREYFUS
Ecole Supérieure de Physique et de Chimie Industrielles de la Ville de Paris
Laboratoire d'Electronique
10, rue Vauquelin
F - 75005 PARIS - FRANCE
Phone: 33 1 40 79 45 41 ; Fax: 33 1 40 79 44 25; E-mail: uifr000@frors31.bitnet

**Keywords:** Neural networks, non-linear process identification, non-linear adaptive control, recurrent neural networks, adaptive training

## 1 INTRODUCTION

The development of engineering applications of neural networks makes it necessary to clarify the similarities and differences between the concepts and methods developed for neural networks and those used in more classical fields such as filtering and control. In previous papers [Nerrand et al. 1993], [Marcos et al. 1993], the relationships between non-linear adaptive filters and neural networks have been investigated, and a general framework has been introduced, which encompasses the *recursive* training of neural networks and the adaptation of non-linear filters. Out of this approach, three new families of training algorithms for feedback networks emerged; algorithms used routinely in adaptive filtering and in the training of neural networks were shown to be specific cases of this general approach.

The adaptive identification of non-linear processes is a natural field of application of these algorithms. The first part of the paper will be devoted to a short survey of the recursive training of *feedback* (also termed *recurrent*) discrete-time neural networks for non-linear identification; the algorithms presented in that section can be used either for adaptive or for non-adaptive systems.

Pursuing our effort along the same lines,we show that algorithms for the adaptive control of non-linear processes by neural networks can be derived from the above approach. However, process control has its own goals and constraints: therefore, the algorithms must be tuned to such specificities. The second part of the paper is devoted to the presentation of these algorithms, which are illustrated in detail in the third part.

## 2 RECURSIVE TRAINING OF FEEDBACK NEURAL NETS

### Network

A neural network architecture of the type shown on Figure 1, featuring M external inputs, N feedback inputs and one output, can implement a fairly large class of non-linear functions; the most general form for the feedforward part is a *fully-connected net.* The basic building block of the network is a "neuron", which performs a weighted sum of its inputs and computes an "activation function" f - usually non linear - of the weighted sum:

$$z_i = f_i[v_i] \quad \text{with } v_i = \sum_j c_{ij}\, x_j$$

where $z_i$ denotes the output of neuron i, and $x_j$ denotes the j-th input of neuron i; $x_j$ may be an external input, a state input, or the output of another neuron.

The task of the network is determined by a (possibly infinite) set of inputs and corresponding desired outputs. At each sampling time n, an error e(n) is defined as the difference between the desired output d(n) and the actual output of the network y(n): e(n)=d(n)-y(n). The network training algorithms aim at finding the synaptic coefficients which minimize a given satisfaction criterion involving, usually, the squared error $e(n)^2$ [Widrow 1985]

Thus, it is clear that filters and neural networks are formally equivalent, and that neural networks, which are potentially capable of realizing *non-linear* input-output relations, define a class of non-linear filters.

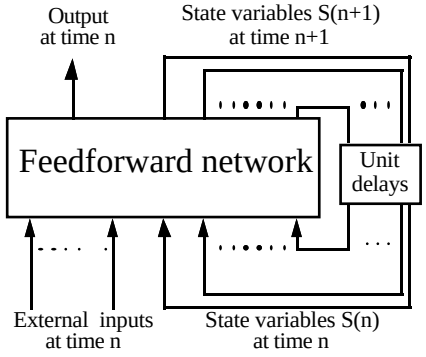


FIGURE 1.
Any discrete-time recurrent neural network of order N can be cast into the above form (termed *canonical form),* featuring M external inputs and N state variables S(n).

### Three families of algorithms for the recursive training of neural networks

The present paper focusses on gradient-based methods using a sliding window of length $N_C$, whereby the updating of the synaptic coefficients is given, at time n, by

$$\Delta c_{ij}(n) = -\mu\, D_n \left[ \frac{\partial}{\partial c_{ij}} \left( \frac{1}{2} \sum_{k=n-N_c+1}^{n} e(k)^2 \right) \right]_{C(n-1)} \qquad (1)$$

where $\mu$ is the gradient step and D[.] stands for a linear transformation of the gradient. These algorithms aim at diminishing, on the average, the value of the "training function"

$$T(n) = \frac{1}{2} \sum_{k=n-N_c+1}^{n} e(k)^2$$

It is well known that, for linear FIR filters with stationary input and output sequences, this algorithm, with $N_c=1$, produces a sequence of coefficients which converges to the coefficients which minimize the expectation value of the squared error (LMS algorithm).

For the computation of the gradient to be meaningful, the coefficients must be considered as being constant on a window of length $N_t \geq N_c$, corresponding to the last $N_t$ sampling times. Thus, *for the updating at time n*, the $N_c$ errors {e(k)} and their partial derivatives, appearing in relation (1), are computed from $N_t$ *computational blocks*; the values of the coefficients used for all $N_t$ blocks are the coefficients C(n-1) which were updated at time n-1. We denote by $S_{in}^m(n)$ the value of the state input of block m at time n, by $S_{out}^m(n)$ the state output, and by $e^m$ the error computed by block m. The choice of $N_c$ and $N_t$ depends on several factors, including the typical time scale of the non-stationarity of the signals.

The choice of the values of the state inputs and of their partial derivatives, as inputs of each block, gives rise to a variety of algorithms. These algorithms fall into three categories depending on the choice of the state inputs.

1) Directed algorithms (Figure 2) : the values of the state inputs $S_{in}(n)$ are taken equal to their desired values D(n), for all blocks; therefore, the network is trained as though it were a *feedforward* network obtained by suppressing the feedback loops of the canonical form of the network. Thus, the error $e^m(n)$ does not depend on the errors computed previously {$e^k(n)$, k=1 to m-1).

2) Semi-directed algorithms (Figure 3) : the values of the state inputs of the first block at time n are taken equal to their desired values, and the values of the state inputs of the other blocks are taken equal to the state outputs of the previous block.

3) Undirected algorithms (Figure 4) : the values of the state inputs of the first block at time n are taken equal to the corresponding states computed at time n-1, and the values of the state inputs of the other blocks are taken equal to the state outputs of the previous block; therefore, errors are computed recursively: $e^m(n)$ depends on the errors computed previously.

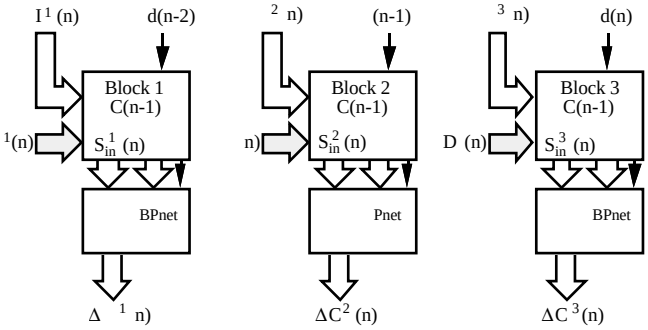


FIGURE 2
Directed algorithm (with $N_t=N_c=3$). The blocks perform all operations necessary for the computation, at time n, of $\Delta C^1(n)$, $\Delta C^2(n)$, $\Delta C^3(n)$; the total coefficient modification is given by: $\Delta C(n) = \Delta C^1(n)+\Delta C^2(n)+\Delta C^3(n)$. For more details see [Nerrand et al. 1993{ EX "Nerrand et al. 1993" }].

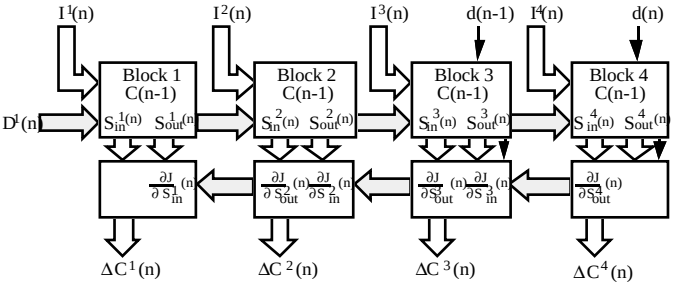


FIGURE 3
Semidirected algorithm (with $N_t=4$ and $N_c=2$). The blocks perform all operations necessary for the computation of $\Delta C^1(n)$, $\Delta C^2(n)$, $\Delta C^3(n)$; the total coefficient modification is given by: $\Delta C(n) = \Delta C^1(n)+\Delta C^2(n)+\Delta C^3(n)+\Delta C^4(n)$. For more details see [Nerrand et al. 1993].

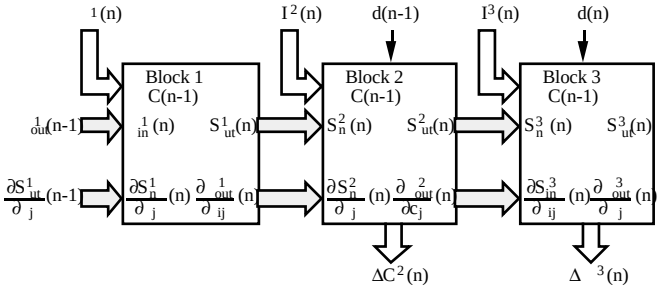


FIGURE 4
Undirected algorithm (with $N_t=3$ and $N_c=2$). The blocks perform all operations necessary for the computation, at time n of $\Delta C^1(n)$, $\Delta C^2(n)$, $\Delta C^3(n)$; the total coefficient modification is given by: $\Delta C(n) = \Delta C^2(n)+\Delta C^3(n)$. For more details see [Nerrand et al. 1993]

| Algorithm | $S^1_{in}(n)$ | $S^m_{in}(n)$ | $\frac{\partial S^1_{in}}{\partial c_{ij}}(n)$ | $\frac{\partial S^m_{in}}{\partial c_{ij}}(n)$ |
|---|---|---|---|---|
| Directed (D) | Des. val. | Des. val. | zero | zero |
| D-SD | Des. val. | Des. val. | zero | $\frac{\partial S^{m-1}_{out}}{\partial c_{ij}}(n)$ |
| D-UD | Des. val. | Des. val. | $\frac{\partial S^1_{out}}{\partial c_{ij}}(n-1)$ | $\frac{\partial S^{m-1}_{out}}{\partial c_{ij}}(n)$ |
| Semi-Directed (SD) | Des. val. | $S^{m-1}_{out}(n)$ | zero | $\frac{\partial S^{m-1}_{out}}{\partial c_{ij}}(n)$ |
| SD-D | Des. val. | $S^{m-1}_{out}(n)$ | zero | zero |
| SD-UD | Des. val. | $S^{m-1}_{out}(n)$ | $\frac{\partial S^1_{out}}{\partial c_{ij}}(n-1)$ | $\frac{\partial S^{m-1}_{out}}{\partial c_{ij}}(n)$ |
| Undirected (UD) | $S^1_{out}(n-1)$ | $S^{m-1}_{out}(n)$ | $\frac{\partial S^1_{out}}{\partial c_{ij}}(n-1)$ | $\frac{\partial S^{m-1}_{out}}{\partial c_{ij}}(n)$ |
| UD-D | $S^1_{out}(n-1)$ | $S^{m-1}_{out}(n)$ | zero | zero |
| UD-SD | $S^1_{out}(n-1)$ | $S^{m-1}_{out}(n)$ | zero | $\frac{\partial S^{m-1}_{out}}{\partial c_{ij}}(n)$ |

Table 1.
Summary of algorithms
Des. val. = desired value

Directed and semi-directed algorithms can be used only if all state variables have desired values, as is the case in black-box models, where the state variables are the output and its delayed values. If some, but not all, state inputs do not have desired values, hybrid versions of the above algorithms can be used: those state inputs for which no desired values are available are taken equal to the corresponding computed state variables (as in an undirected algorithm), whereas the other state inputs may be taken equal to their desired values (as in a directed or semi-directed algorithm).

In each category, three algorithms are defined, depending on the choice of the partial derivatives of the state inputs. This is summarized in Table 1. Note that backpropagation cannot be used with UD, D-UD and SD-UD algorithms.

Illustrative examples of the application of these algorithms to problems in adaptive filtering and in process identification can be found in [Dreyfus et al. 1992{ EX "Dreyfus et al. 1992" }].

## 3 NON-LINEAR IDENTIFICATION BY NEURAL NETWORKS

The application of these algorithms to identification is straightforward: the desired output of the network whose synaptic coefficients must be computed is the output of the process to be modelled. In the following, we show how the above algorithms are related to classical approaches in system identification [Ljung 1987].

Three approaches with black-box models ($S(k) = \{y(k), y(k-1), ...y(k-N+1\}$)will be considered, depending on the assumptions made on the process : (i) the NARX (*equation error*) model , (ii) the NARMAX model, and (iii) the *output error* model .

The first initials of the acronyms refer to the choice of the state inputs; the second initial refers to the choice of the partial derivatives of the state inputs. Backpropagation cannot be used with D-UD, SD-UD and UD algorithms.In the *equation error* approach (Non-linear Auto-Regressive with eXogeneous inputs, or NARX, model), it is assumed that the process obeys the following equations:

$$s(k+1) = \varphi\left[U(k), S(k)\right] + w(k+1) \ ,$$
$$y(k) = s(k) \ ,$$

where w(k) is white noise and U(k) stands for the external inputs. The associated predictor of the output y(k) is given by

$$y_m(k+1) = \varphi\left[U(k), S(k)\right] \ ,$$

where $y_m(k)$ is the predicted value of y(k). Thus, 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. This has an important consequence: assume that there exists a neural network which can realize function φ exactly; this network can therefore implement the predictor, *and it has precisely the structure of the network which is trained by a Directed algorithm,* as defined above.

A NARMAX (Non-linear Auto-Regressive Moving Average with eXogeneous inputs) model obeys the following equations:

$$s(k+1) = \varphi\left[U(k), S(k), W(k)\right] + w(k+1),$$
$$y(k) = s(k) \ ,$$

where $W(k) = \{w(k), w(k-1), ..., w(k-N_W+1)$. The associated predictor is defined by:

$$y_m(k+1) = \varphi\left[U(k), S(k), E(k)\right] \ ,$$

with $e(k)=y(k)-y_m(k)$ and $E(k)=\{e(k), e(k-1), ..., e(k-N_w+1)\}$ .
Thus, the predictor of the NARMAX process is *recurrent* of order $N_W$.

In the *output error* approach, it is assumed that the process obeys the following equations:

$$s(k+1) = \varphi\left[U(k), S(k)\right] \ ,$$
$$y(k) = s(k) + w(k) \ .$$

In this case, the associated predictor for the ouptut y(k) is given by:

$y_m(k+1) = \varphi\left[U(k), Y_m(k)\right]$ ,
where $Y_m(k) = \{y_m(k), y_m(k-1), ..., y_m(k-N+1)\}$ .
Thus, the associated predictor of the output error process is *recurrent* of order N. Assume that there exists a neural network which can realize function φ exactly; this network can therefore implement the predictor, *and it has precisely the structure of the network which is trained by a Undirected algorithm,* as defined above.

To summarize, the algorithms derived for the adaptive training of discrete-time neural networks can readily be applied to the adaptive identification of dynamical non-linear processes. Directed algorithms are best suited to the identification of equation error models, whereas Undirected algorithms are best suited to the identification of NARMAX and output error

models. It is intuitive, and it can be shown analytically in simple cases [Dreyfus et al. 1992], that Semi-directed algorithms bridge the gap between these approaches; they can be especially useful when dealing with unstable NARMAX predictors [Chen and Billings 1989].

## 4 ADAPTIVE TRAINING OF FEEDBACK NEURAL NETS FOR NON-LINEAR CONTROL

The approach to non-linear control which is presented in this section can be regarded as a non-linear, adaptive version of Linear Quadratic (LQ) state feedback control with a reference model (see for instance [Åström and Wittenmark 1989]). It capitalizes on the fact that the algorithms described above can be used for controlling the system in such a way that the output of the process can be made as close as possible to the output of the (suitably chosen) reference model.

We first make the assumption that the process to be controlled has been identified by one of the above algorithms, so that there exists a neural network predictor model, whose coefficients are known. In addition, assume that a reference model of the process, prescribing the desired output $y_r$ corresponding to the reference sequence {r}, has been designed. We want to find the coefficients of a neural network controller which computes a control sequence {u}, from the reference sequence {r}. Thus, the ingredients of the control system are:

- a reference model,
- a "neural" predictor, with fixed coefficients, which computes its output $y_m$ from the control signal U,
- a "neural" controller, whose coefficients are to be computed, adaptively or non-adaptively.

We denote the output of the process by $y_p$.

### Three families of algorithms

1) R-M, RS-M and U-M algorithms (Figure 5)

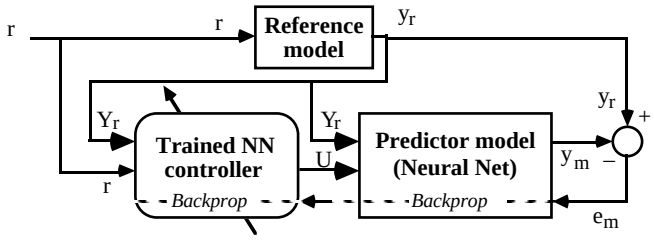


FIGURE 5
Architecture associated to a R-M control algorithm

The training function involves the squared difference between the output of the model (hence the M in the acronym) and the output of the reference model. In a R-M algorithm, the state variables and their partial derivatives are initialized to the values of the state variables of the reference model (hence the R in the acronym), and the algorithm is run in a *Directed* fashion. RS-M and U-M algorithms are similar in spirit except for the fact that they are run in a *Semi-directed* or *Undirected* fashion. The approach of neural control developed in [Narendra et Parthasarathy 1991] lies entirely within the specific framework of R-M, RS-M and U-M algorithms.

The process itself is absent in these algorithms: therefore, they can be useful for validating the structure of the controller by computer simulations, since they do not require any measurement to be taken from the plant.

2) P-M and PS-M algorithms (Figure 6)

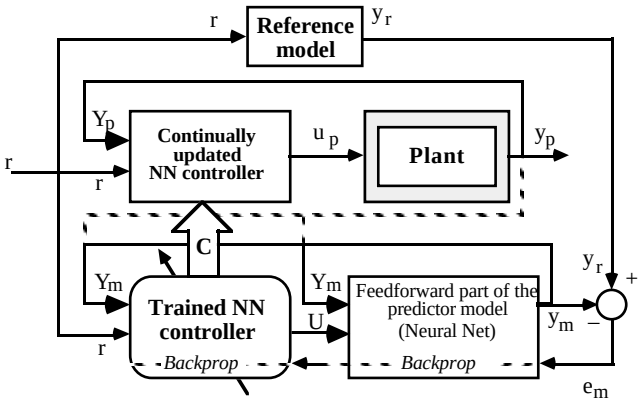


FIGURE 6
Architecture associated to a PS-M control algorithm

The training function involves the squared difference between the output of the model (hence the M in the acronym) and the output of the reference model. In a P-M algorithm, the state variables and their partial derivatives are initialized to the values of the state variables of the process (hence the P in the acronym), and the algorithm is run in a *Directed* fashion. PS-M algorithms are similar in spirit except for the fact that they are run in a *Semi-directed* fashion.

Here, the process itself is used for the state initializations, so that this architecture can actually be used for controlling the process in an adaptive context.

3) P-P, PS-P and U-P algorithms (Figure 7)

The training function involves the squared difference between the output of the process (hence the second P in the acronym) and the output of the reference model. In a P-P algorithm, the state variables and their partial derivatives are initialized to the values of the state variables of the process (hence the first P in the acronym), and the algorithm is run in a *Directed* fashion. PS-P and U-P algorithms are similar in spirit except for the fact that they are run in a *Semi-directed* or *Undirected* fashion.

Here again, the process itself is used for the state initializations, so that this architecture can actually be used for controlling the process in an adaptive context.

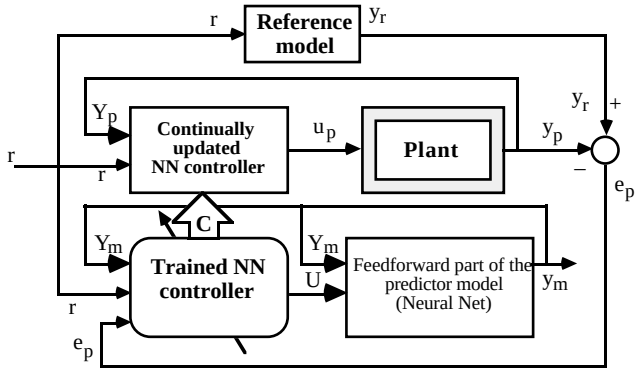


FIGURE 7
Architecture associated with a U-P algorithm.

R-M, P-M and P-P algorithms must involve NARX predictors; all other algorithms must involve recursive predictors.

In addition to the above algorithms, other combinations of state initializations and error computations are possible. For the sake of brevity, they will not be considered here.

## 5 ILLUSTRATION: IDENTIFICATION AND CONTROL OF A NON-LINEAR PROCESS

We consider a non-linear first-order process simulated by

$$y_p(n+1) = \left[1 - \frac{T}{a_1+a_2 y_p(n)}\right] y_p(n) + T\,\frac{b_1+b_2 y_p(n)}{a_1+a_2 y_p(n)}\,u(n)$$

with $a_1$=-0.14 , $a_2$=1.2 , $b_1$=5.63 , $b_2$=-0.33 , T=0.1 sec.

**Identification** can be performed adaptively with a simple first-order model which is locally accurate in state space, or non-adaptively with a more complex first-order model which is globally accurate in the region of state space which is spanned during training. Figure 8 shows the response and the identification error of the adaptive equation error predictor in response to a step function with added noise; function φ is implemented by a linear neuron with inputs u(n), $y_p$(n) and bias. Figure 9 shows the response and the identification error of the non-adaptive model (same inputs, one layer of five hidden neurons and one linear ouput neuron) after training with a Directed algorithm. The adaptive model is more accurate than the non-adaptive one except during transients.

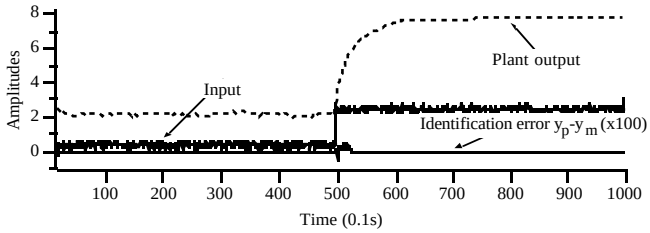


FIGURE 8
Adaptive training of the model by a Directed algorithm with $N_t$=$N_C$=20 using a quasi-Newton method.

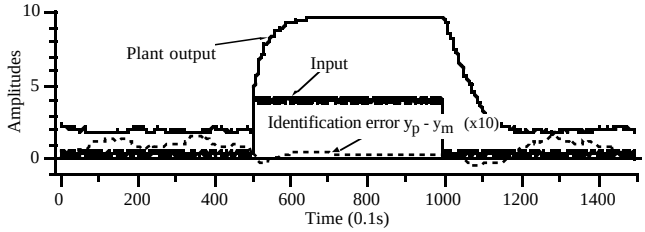


FIGURE 9
Non-adaptive training of the model by a Directed algorithm with $N_t$=$N_C$=2000 using a quasi-Newton method. Training set: input pulses of random amplitude and width.

**Control:** the reference model is a first-order low-pass filter. The predictor model is the above non-adaptive model. The controller is a fully connected feedforward network with four hidden neurons; its inputs are r(n), y(n) and a bias; the amplitude of the control signal u(n) is constrained to be in the interval [0.1, 10]. Training takes place in two steps: first, a R-M algorithm ($y(n)=y_r(n)$) is used non-adaptively in order to get a first approximation of the coefficients; the resulting network is subsequently adapted with a P-M algorithm ($y(n)=y_p(n)$). Figure 10 shows the error during the latter phase.

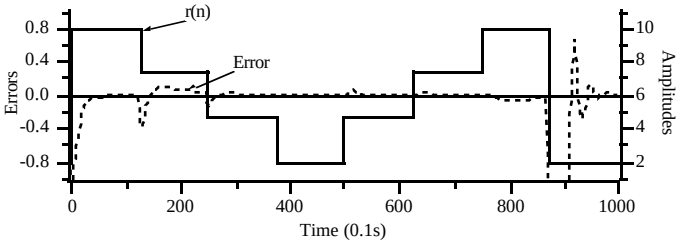


FIGURE 10
Error during the final adaptive training of the controller by a P-M algorithm ($N_t$=$N_C$ =1, quasi-Newton method). Reference model: $y_r(n+1) = 0.9\ y_r(n) + 0.1\ r(n)$.The large errors on the left corresponds to the beginning of adaptation; the large errors on the right are due to the saturation of the control signal.

## 6 CONCLUSION

We have presented a general approach to the identification and training of non-linear dynamical systems with neural networks. The general framework for recursive identification, which had been proposed earlier, has been briefly summarized and put into the perspective of familiar recursive identification approaches. In the second part of the paper, we have presented three families of new algorithms for control. We have illustrated these approaches with an example of identification and control, both adaptive and non-adaptive.

## REFERENCES


Åström, K.J. and B. Wittenmark 1989. *Adaptive Control.* Addison-Wesley.

Chen, S. and A. Billings 1989. Recursive Prediction Error Parameter Estimator for Non-linear Models. *Int. J. Control*, **49**, 569-594.

Dreyfus, G., O. Macchi, S. Marcos, O. Nerrand, L. Personnaz, P. Roussel-Ragot, D. Urbani, C. Vignat 1992. Adaptive Training of Feedback Neural Networks for Non-linear Filtering. In *Neural Networks for Signal Processing II*, S.Y. Kung, F. Fallside, J. Aa. Sorenson, C.A.Kamm, eds.,550-559 (IEEE).

Ljung, L. 1987. *System Identification: Theory for the User.* Prentice-Hall.

Marcos, S., P. Roussel-Ragot, L. Personnaz, O. Nerrand, G. Dreyfus, C. Vignat 1993. Réseaux de Neurones pour le Filtrage Non Linéaire Adaptatif. *Traitement du Signal* **8**, 409-422.

Narendra, K. S., and K. Parthasarathy. 1991. Gradient Methods for the Optimization of Dynamical Systems Containing Neural Networks. *IEEE Trans. on Neural Networks* **2**, 252-262.

Nerrand O., P. Roussel-Ragot, L. Personnaz, G. Dreyfus 1993. Neural Networks and Non-linear Adaptive Filtering: Unifying Concepts and New Algorithms. *Neural Computation* **5**, 165-197.

Widrow, B., S.D. Stearns, Adaptive Signal Processing (Prentice-Hall, 1985).