In Neurocomputing ***20****(1-3): 279-294 (1998).*

A recursive algorithm based on the extended Kalman filter for the training of feedforward neural models

Isabelle Rivals and Léon Personnaz

Laboratoire d'Électronique, École Supérieure de Physique et Chimie Industrielles,
10 rue Vauquelin, 75231 Paris Cedex 05, France.
Phone: 00 33 1 40 79 45 45 Fax: 00 33 1 40 79 44 25
E-mail: Isabelle.Rivals@espci.fr

Abstract

The Extended Kalman Filter (EKF) is a well known tool for the recursive parameter estimation of static and dynamic nonlinear models. In particular, the EKF has been applied to the estimation of the weights of feedforward and recurrent neural network models, i.e. to their training, and shown to be more efficient than recursive and non recursive *first-order* training algorithms; nevertheless, these first applications to the training of neural networks did not fully exploit the potentials of the EKF. In this paper, we analyze the specific influence of the EKF parameters for modeling problems, and propose a variant of this algorithm for the training of feedforward neural models which proves to be very efficient as compared to non recursive *second-order* algorithms. We test the proposed EKF algorithm on several static and dynamic modeling problems, some of them being benchmark problems, and which bring out the properties of the proposed algorithm..

Keywords: Feedforward Neural Networks; Process Modeling; Second-Order Training Algorithms; Extended Kalman Filter.

1. Introduction

The Kalman Filter (KF) provides a solution to the problem of estimating the state of processes described by linear stochastic dynamic models. In the case of a nonlinear model, the Extended Kalman Filter (EKF), which was proposed in the early sixties, linearizes the model around the current state estimate, and applies the KF to the resulting time-varying linear model. The EKF was then very soon applied to the estimation of unknown parameters of linear dynamic systems, by including the unknown parameters in the state of the system [5]. Recently, the EKF has also been used for the estimation of unknown parameters of nonlinear static or dynamic systems, thus to the training of neural network models, i.e. to the estimation of their weights, both for feedforward networks [14] [11], and for recurrent networks [7] [15]. In these papers, it was mainly shown that the EKF converges faster and to lower minima than

recursive and non recursive *first-order* Prediction Error Methods (PEM) using the backpropagation algorithm to compute the gradient of the cost-function.

Nevertheless, these first attempts to use the EKF for the training of neural networks did not fully exploit the potentials of the EKF. In this paper, we propose a variant of the EKF algorithm which compares with the most efficient non recursive *second-order* algorithms on static and dynamic modeling problems involving the training of feedforward neural networks. The paper is organized as follows: Section 2 describes the black-box modeling problems using feedforward neural networks we are dealing with. Section 3 recalls the classic training of such neural models with a non recursive second-order PEM. Section 4 presents the application of the EKF to the training of feedforward neural models, and, after an analysis of the influence of the algorithm parameters, proposes an efficient choice of their values. Section 5 tests the proposed EKF algorithm on several modeling problems, among them benchmark problems which allow a direct comparison to the previous attempts to use the EKF for the training of neural networks, and to a non recursive second-order algorithm.

2. Process modeling using feedforward neural networks

We deal with the following black-box modeling problems involving the training of feedforward neural models (for the sake of the simplicity and of the clarity of the presentation, we restrict to single-output problems, but the EKF algorithm proposed in section 4 can be easily extended to the multiple-output case):

Static modeling problems

We consider the static modeling of a process with n_x-input vector x and scalar output y_p. In the input domain of interest, the static behavior of the process for a fixed input x^a is assumed to be described by:

$$y_p^a = f(x^a) + w^a \tag{1}$$

where w^a is a random variable with zero mathematical expectation and variance σ_w^2, and where the regression function $f(x) = E(y_p \mid x)$ is nonlinear and unknown. The modeling problem consists in finding a neural network which gives a good estimation of the regression in the input domain of interest.

Let $\varepsilon > 0$ be an arbitrary small scalar. Since feedforward neural networks using ridge functions (for example multi-layer perceptrons (MLP) with *tanh* activation function hidden neurons) have been shown to be universal approximators [3], there exists at least a feedforward neural network $\phi(x\,;\theta)$, where $\theta \in \mathbb{R}^{n_\theta}$ denotes the weights of the network, and a value θ_ε of its weights, such that:

$$\left| f(x) - \phi(x\,;\theta_\varepsilon) \right| < \varepsilon \tag{2}$$

in the input domain of interest. A neural network with a sufficient number of neurons can thus theoretically implement a good estimation of the regression in this domain.

NARX modeling problems

An important class of dynamic modeling problems also involves the training of feedforward neural models. Let us consider a process with scalar input u and scalar output y_p affected by additive state noise (Nonlinear AutoRegressive with eXogenous input process) whose dynamic behavior is assumed to be described by:

$$y_p(k) = f\left(y_p(k\text{-}1), \ldots, y_p(k\text{-}n_y), u(k\text{-}1), \ldots, u(k\text{-}n_u)\right) + w(k) \tag{3}$$

where $\{w(k)\}$ is a sequence of independent and identically distributed (i.i.d.) random variables with zero expectation and variance σ_w^2. The regression function $f(x^k) = E\left(y_p^k \mid x^k\right)$ is nonlinear and unknown, where y_p^k denotes the value taken by $y_p(k)$, and x^k the vector of $n_x = n_y + n_u$ past outputs and inputs $\left[y_p(k\text{-}1) \ldots y_p(k\text{-}n_y)\ u(k\text{-}1) \ldots u(k\text{-}n_u)\right]^T$. The regression function provides the optimal prediction of the process output [9] [13].

The universal approximation property again ensures that a feedforward neural network with a sufficient number of neurons can in principle implement a predictor which is arbitrarily close to the optimal one in the input domain of interest.

For these two classes of modeling problems, an estimation of the regression f, i.e. of the weights of a neural model, will be obtained using a finite set of N examples $\left\{x^k,\ y_p^k\right\}_{k=1 \text{ to } N}$, or training set, where y_p^k denotes the value taken by $y_p(k)$ in the case of a NARX modeling problem.

3. Feedforward neural model training with non recursive prediction error methods

The estimation of the network weights can be achieved with a classic PEM, that is by minimizing the following quadratic cost-function, in a non recursive, iterative fashion (batch training):

$$J(\theta) = \frac{1}{2} \sum_{k=1}^{N} \left(e^k\right)^2 = \frac{1}{2} \sum_{k=1}^{N} \left(y_p^k - \phi\left(x^k\ ;\ \theta\right)\right)^2 \tag{4}$$

The common principle of second-order algorithms is to compute a descent direction obtained by a linear transformation of the gradient using the Hessian of the cost-function, or an approximation of it. In this paper, a Levenberg-Marquardt algorithm is used: at each epoch, the weights are updated according to:

$$\Delta\theta = -\left[\widetilde{H} + \lambda\, I_{n_x}\right]^{-1} \nabla J(\theta) \tag{5}$$

where $\nabla J(\theta)$ denotes the gradient of the cost-function, I_n denotes the (n, n) identity matrix. The matrix $\widetilde{H}$ is an approximation of the Hessian whose components $(\widetilde{H})_{ij}$ are given by:

$$(\widetilde{H})_{ij} = \sum_{k=1}^{N} \frac{\partial e^k}{\partial \theta_i} \frac{\partial e^k}{\partial \theta_j} \tag{6}$$

where the $\left\{\partial e^k / \partial \theta_i\right\}$ are computed in an economical fashion with the backpropagation algorithm. The training parameter λ is chosen so as to guarantee that the cost-function is diminished, and λ is decreased as much as possible in order to tend to a Newton-like direction

as soon as a minimum is approached. A simple and efficient algorithm for setting λ is given in [10].

With a sufficiently large number of hidden neurons, it is possible to guarantee that the mean square error on the training set, the TMSE = $\frac{2}{N} J(\theta)$, becomes arbitrarily small. Nevertheless, the problem of finding a function from a finite set of points, the training set, is an ill-posed problem. Thus, even if for a given ε the TMSE obtained after training is smaller than ε^2 , (2) might not be satisfied in the whole input domain of interest, due to overfitting.

In practice, overfitting can be avoided by the successive trainings of neural models with an increasing number of neurons, and by selecting the minimal size neural model with the smallest mean square error on a performance estimation set (the PMSE), which is independent from the training set. We will refer to this type of procedure as "Stepwise Modeling Procedure", as illustrated in Fig. 1.

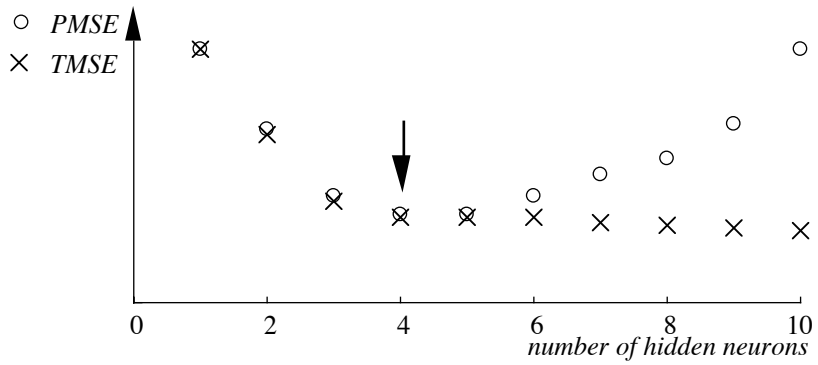

Fig. 1. Stepwise Modeling Procedure: the arrow indicates the selected neural network.

In order to avoid overfitting, it is possible to use a third independent set, or validation set, to perform an early stopping of each training: this is sometimes called "implicit regularization", as opposed to an "explicit regularization". A simple form of explicit regularization is the so-called "weight decay" [1], which consists in adding the sum of the squares of the network parameters, weighted by a regularization parameter $\rho > 0$, to the cost-function (4):

$$J_\rho(\theta, \rho) = \frac{1}{2} \sum_{k=1}^{N} \left(y_p^k - \phi\left(x^k\ ;\ \theta\right)\right)^2 + \rho\, |\theta|^2 \tag{7}$$

A large value of ρ constrains the weights to be small, and thus prevents overfitted regions with large curvatures. Nevertheless, a performance estimation set is still needed to select a neural model among the candidates.

4. Proposed algorithm for the training of feedforward neural models based on the extended Kalman filter

We consider a modeling problem defined by a training set $\left\{x^k,\ y_p^k\right\}_{k=1 \text{ to } N}$ and a candidate neural model $\phi\left(x\ ;\ \theta\right)$. Let us make the assumption (which is not necessarily true) that the

family of functions defined by the neural network contains the regression function f present in (1) or (3), i.e. that there exists a value θ_p of θ such that:

$$\phi\,(x\,;\,\theta_p) = f(x) \tag{8}$$

The EKF (for a general presentation of the EKF, see for instance [2]) can then be used as a recursive algorithm for the estimation of the weights of the neural model by considering the following dynamic system, which is equivalent to (1) or (3):

$$\begin{cases} \theta_p(k+1) = \theta_p(k) & (= \theta_p) \\ y_p(k) = \phi\,(x(k)\,;\,\theta_p(k)) + w(k) \end{cases} \tag{9}$$

where $x(k)$ and $y_p(k)$ are the values taken by the training pair x^k and y_p^k, the order being arbitrary in the case of a static modeling problem; $\{w(k)\}$ denotes a white noise sequence with zero expectation and variance σ_w^2. The EKF applied to system (9) gives at time k an estimate $\theta(k)$ of the weight vector θ_p by the following recursion:

$$\begin{cases} H(k) = \left.\dfrac{\partial \phi\,(x(k)\,;\,\theta)}{\partial \theta}\right|_{\theta\,=\,\theta(k\text{-}1)} \\ P(k) = P(k\text{-}1) - \dfrac{P(k\text{-}1)\,H(k)\,H(k)^T\,P(k\text{-}1)}{H(k)^T\,P(k\text{-}1)\,H(k) + r} \\ K(k) = \dfrac{P(k)\,H(k)}{r} \\ \theta(k) = \theta(k\text{-}1) + K(k)\left(y_p(k) - \phi\,(x(k)\,;\,\theta(k\text{-}1))\right) \end{cases} \tag{10}$$

$H(k)$ is the gradient vector of the output of the neural network with respect to the weights around their available value $\theta(k\text{-}1)$, gradient which is computed with the backpropagation algorithm. In this parameter estimation frame, the matrix $P(k)$ is a recursive estimate of the covariance matrix of the error on the weight vector $\theta_p - \theta(k)$. $K(k)$ is the Kalman gain vector. At the first time step $k = 1$, $\theta(0)$ and $P(0)$ are fixed to arbitrary values θ_0 and P_0. This procedure can be iterated by presenting the training sequence as often as necessary. The EKF recursion is based on a linearization, and there are no established conditions for its convergence.

Let us now discuss the influence of the algorithm parameters: the initial weight vector θ_0, the initial matrix P_0, and the scalar r.

The initial weight vector θ_0, the initial matrix P_0, and the scalar r

One would like to initialize the weight vector close to its "true" value θ_p using *a priori* knowledge, which is not possible in the case of a neural black-box model. For the networks with *tanh* hidden neurons used in this paper, the weights should thus be initialized in the same way as when the network is to be trained by a PEM method, i.e. such that the initial output values of the hidden neurons are in the linear part of the *tanh*. We propose to initialize the weights with small random values, a choice which is also made in most EKF approaches (in [4], [11] and [12] for example).

Initially, since $H(0)$ is fixed by the problem and by θ_0, the weight changes are governed by the quantity $\dfrac{P_0}{r}$. If $\dfrac{P_0}{r}$ is large, the weight estimates are allowed to change much and fast

during the first time instants. As opposed to the case of a linear model, such large changes in the weights of a neural model are not desirable for two reasons: (i) due to the linearization around the initial weight estimates, which are very inaccurate, these changes may be inappropriate and drive the algorithm to a local minimum; (ii) the weights of a neural model with *tanh* hidden neurons and normalized input and output values implementing a reasonably smooth function are usually of the order of 1. In order to avoid large initial weight changes, we thus propose to initialize P_0 to $p_0 I_{n_\theta}$, with $\frac{p_0}{r}$ small, typically $\frac{p_0}{r} = 10^{-2}$. This choice differs from that made in [4] and [11], where the constraints imposed by the use of neural network are not particularly taken into account, and where the diagonal terms of P_0 are chosen of the order of 100, r being also set to value close to 1.

In the theoretical EKF formulation, the scalar r is σ_w^2, the variance of the measurements in the assumed models (1) or (3). Thus, the smaller the value of r, the more the measurements are taken into account by the EKF algorithm; as a matter of fact, the parameter r can be considered as an inverse learning rate [12]. We propose here to decrease r from a value r_0 equal to the initial TMSE (which is of the order of 1 in the case of normalized process outputs $\{y_p^k\}$ and small initial weights θ_0) to a small value r_f, with for instance an exponential decay:

$$r(i) = (r_0 - r_f)\ exp\ (-\alpha\ i) + r_f \qquad (11)$$

where i denotes the number of the epoch, and $\alpha > 0$. The greater the value of α, the faster the convergence, with a problem-dependent upper limit due to the possible numerical instability at the beginning of the training. Note that the parameter r can also be decreased between time instants, as proposed in [12], [14] and [15]. The parameter r_0 being now fixed to a value close to 1, we can take $p_0 \approx 10^{-2}$. Let us show on an example the influence of r_f.

We consider the static modeling of a process with scalar input x and output y_p. The training set consists of $N = 40$ pairs with inputs in the range $[-1\ ;\ 1]$, the process behavior being described by the following model:

$$y_p^k = f(x^k) + w^k \qquad (12)$$

where f is the discontinuous function $f(x) = -1$ if $x \leq 0$, $f(x) = sinc\,(10\ x)$ if $x > 0$, $sinc$ denoting the cardinal sine function, and $\{w(k)\}$ is a set of gaussian variables with $\sigma_w^2 = 5\ 10^{-2}$. An independent set of 1000 examples is used for performance estimation. In order to make overfitting possible, a MLP with one layer of 10 *tanh* hidden neurons and a linear output neuron is chosen, which is indeed large with respect to N, to σ_w^2, and to the complexity of f. As a matter of fact, training the network with the Levenberg-Marquardt algorithm without regularization leads to a TMSE of $3\ 10^{-2} < \sigma_w^2$, and a PMSE of $3\ 10^{-1} >> \sigma_w^2$, the weights being initialized with a uniform distribution in $[-0.17\ ;\ 0.17]$ (i.e. with a variance of 10^{-2}).

With the same weight initialization, and $r_0 = TMSE\ (0) = 0.6$, $p_0 = 10^{-2}$, $\alpha = 0.5$, we then study the influence of the value of r_f on the TMSE and PMSE obtained with the EKF, as

shown on Fig. 2. Note that the minimal PMSE is obtained for $r_f = 10^{-5}$, and not for the value of the measurement noise $r_f = \sigma_w^2 = 5\ 10^{-2}$.

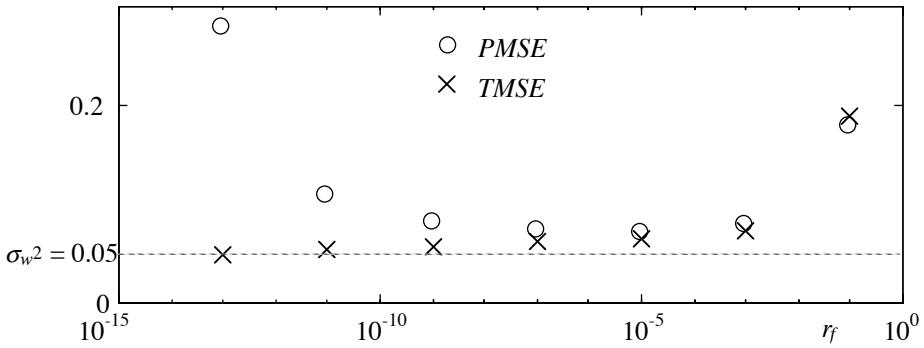

Fig. 2. Discontinuous function problem: Performance of neural models with a hidden layer of 10 sigmoidal neurons and a linear output neuron, trained with the proposed EKF algorithm for different values of r_f.

The functions implemented for three values of r_f are shown on Fig. 3.

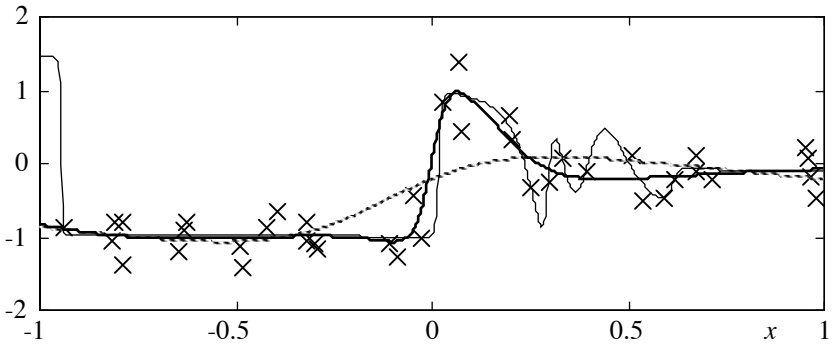

Fig. 3. Discontinuous function problem: Functions implemented by neural models with a hidden layer of 10 sigmoidal neurons and a linear output neuron, trained with the proposed EKF algorithm for three values of r_f: $r_f = 10^{-1}$ (thick dotted line), the optimal value $r_f = 10^{-5}$ (thick solid line), and $r_f = 10^{-13}$ (thin solid line).

As expected, the greater the value of r_f, the less the measurements are taken into account by the EKF. In other words, the parameter r_f acts like a regularization parameter: for a network with a given number of neurons, the greater the value of r_f, the smoother is the function obtained, and the greater is the TMSE. In a situation where one would use a PEM with regularization, typically in the case of very noisy data and/or of a small training set, one should in the same way choose a final value r_f which is not too small. Nevertheless, in the frame of a Stepwise Modeling Procedure, regularization is usually not necessary since this procedure selects a neural model which does not overfit. We thus propose to use the EKF with $r_f \approx 0$ in order to obtain a solution corresponding to a minimum of the cost-function (4) at least as good as that obtained with a Levenberg-Marquardt algorithm without regularization.

Remark

The EKF is able to take state noise into account by considering for example the following state equation:

$$\theta_p(k+1) = \theta_p(k) + v(k) \tag{13}$$

where $\{v(k)\}$ denotes a random vector with zero mean and covariance matrix Q. In the frame of parameter estimation, state noise is usually introduced in order to model a possible drift of the parameters [6]. However, in the case of a time-invariant neural model, i.e. for which the weights must converge to constant values, they should not be allowed to drift. Nevertheless, a diagonal matrix Q with positive components is sometimes added in the update equation for $P(k)$ both to avoid numerical instability, and to help to avoid local minima [11] [12]. But the adequate order of magnitude of the elements of Q which simultaneously makes the weights converge to constant values seems to be highly problem-dependent. In subsection 5.4, we compare the proposed EKF algorithm to the EKF variant using a non zero matrix Q on a benchmark problem.

Summary

The proposed EKF algorithm uses the classic recursion (10) corresponding to a dynamic model without state noise, with initial weights θ_0 fixed to small random values, the parameter r decreasing between epochs according to equation (11), where the initial value r_0 is taken equal to the initial TMSE, the final value r_f is chosen close to zero if no regularization is wished, and the decay parameter α is of the order of 1, and finally the scalar matrix P_0 with small diagonal components (for example $P_0 = 10^{-2} I_{n_\theta}$). These choices are valid for neural networks using *tanh* activation functions, and normalized input-output data.

5. Simulation examples

In this section, we deal with static and dynamic modeling problems, some of them being benchmark problems. They serve as a basis for the comparison of the proposed EKF algorithm to the Levenberg-Marquardt algorithm, and to EKF variants proposed in the literature. Three kinds of neural networks are used:

– $FCP\left(n_x, n_h^{tanh}, n_o^{linear}\right)$ denotes a fully connected perceptron with n_x inputs, n_h hidden neurons, and n_o output neurons, the superscript indicating the activation function of the neurons; the output x_i of neuron i is given by:

$$x_i = f_i\left(\sum_{j=0}^{i-1} c_{ij}\, x_j\right) \qquad i = n_x{+}1 \text{ to } n_x{+}n_o \tag{14}$$

where f_i is the activation function of neuron i, x_0 is the bias, and the $\{c_{ij}\}$ are the components of the weight vector θ;

– $MLP\left(n_x, n_1^{tanh}, \ldots, n_h^{tanh}, n_o^{linear}\right)$ denotes a MLP with n_x inputs, h layers of hidden neurons, and n_o output neurons;

– $MLDCP\left(n_x, n_1^{tanh}, \ldots, n_h^{tanh}, n_o^{linear}\right)$ denotes a MLP with additional direct connections from inputs to outputs.

5.1. Modeling a cardinal sine function (static single-input problem with noise)

We consider the static modeling of a process with scalar input x and output y_p. The training set consists of $N = 200$ pairs with inputs in $[-1 ; 1]$, the behavior of the process being simulated by:

$$y_p^k = sinc\left(7\ (x^k + 0.25)\right) + w^k \tag{15}$$

where $\{w(k)\}$ is a set of gaussian variables with $\sigma_w^2 = 10^{-5}$. An independent set of 1000 examples is used for performance estimation. The training set is shown on Fig. 4.

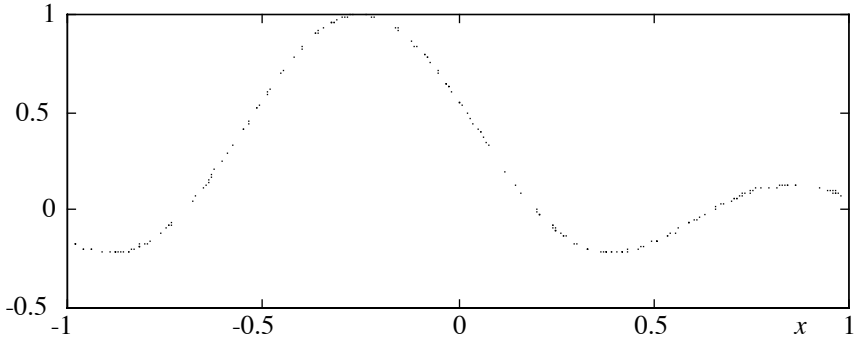

Fig 4. *Sinc* problem: Training set.

A Stepwise Modeling Procedure using $MLP\left(1, n_1^{tanh}, 1^{linear}\right)$ trained with the Levenberg-Marquardt algorithm, their weights being initialized in $[-0.17 ; 0.17]$, shows that the optimal number of neurons is $n_1 = 5$; the corresponding TMSE and PMSE are close to the noise variance (respectively 1.0 10^{-5} and 1.1 10^{-5}).

The proposed EKF algorithm with $p_0 = 10^{-2}$, $r_0 = TMSE\ (0) = 0.2$, $r_f = 10^{-40}$, and a decay factor α in the range $[0.1 ; 5.0]$, leads to the same performance as the Levenberg-Marquardt algorithm. The error of a neural network obtained with the proposed EKF algorithm on the performance estimation set is shown in Fig. 5.

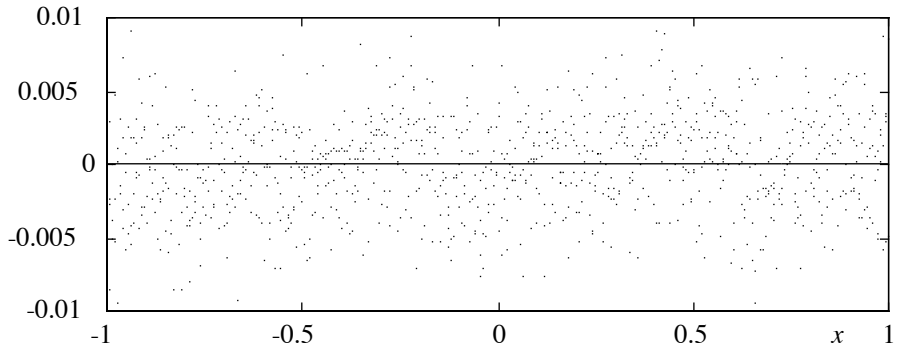

Fig. 5. *Sinc* problem: Error on the performance estimation set obtained with a $MLP\left(1, 5^{tanh}, 1^{linear}\right)$ trained by the proposed EKF algorithm with $r_f = 10^{-40}$.

Using a larger final value of r, $r_f = \sigma_w^2 = 10^{-5}$, and for example $\alpha = 0.5$, the results are less satisfactory: the TMSE equals 3.8 10^{-5} and the PMSE 3.9 10^{-5}. The error of the neural network obtained is shown in Fig. 6.

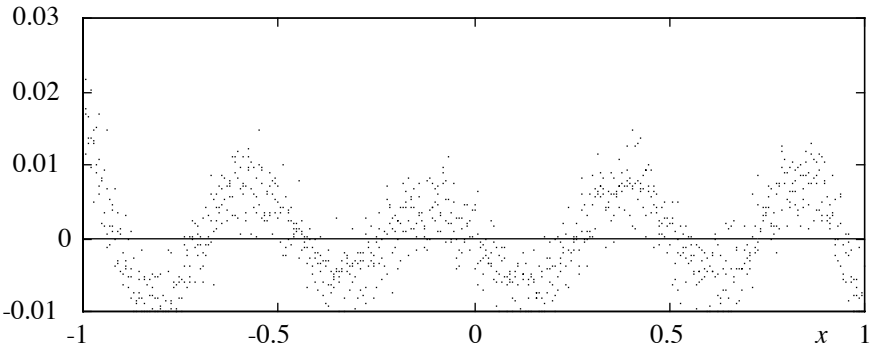

Fig. 6. *Sinc* problem: Error on the performance estimation set obtained with a $MLP\left(1, 5^{tanh}, 1^{linear}\right)$ trained by the proposed EKF algorithm with $r_f = \sigma_w^2 = 10^{-5}$.

The evolution of the TMSE for the different algorithms is shown in Fig. 7: as expected, the greater the value of α, the faster the convergence.

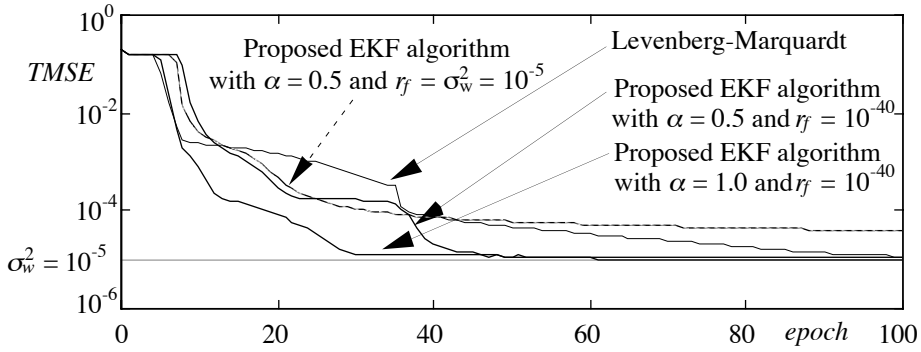

Fig. 7. *Sinc* problem: Evolution of the TMSE.

5.2. The pseudo XOR problem (static 2-input benchmark problem without noise)

The pseudo XOR problem is an academic 2-input classification problem with two non overlapping classes, class 1 occupying the first and third quadrants of the plan, and class 2 the second and fourth quadrants, as shown in Fig. 8.

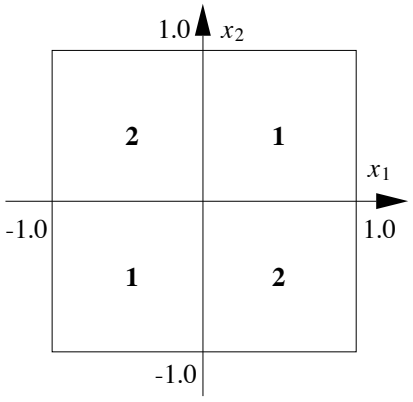

Fig. 8. Pseudo XOR problem: Regions occupied by the two non overlapping classes.

A solution to this problem with the EKF was first proposed in [14]. They chose a training set consisting of $N = 1000$ samples filling the region uniformly. $MLP\left(2, n_1^{tanh}, 1^{tanh}\right)$ were trained. For ten different initialisations of MLPs with up to $n_1 = 4$, a gradient algorithm was never able to achieve correct "decision regions" (i.e. the regions defined by the sign of the

output). The EKF with $r = cte = 1.0$ and $Q = 0$ led a $MLP\,(2,\ 3^{tanh},\ 1^{tanh})$ to correct decision regions six times out of ten, with an average TMSE of 1.1 10^{-1}, and a $MLP\,(2,\ 4^{tanh},\ 1^{tanh})$ nine times out of ten, with an average TMSE of 3.0 10^{-2}, after 645 epochs.

As a matter of fact, this problem should be considered as a static modeling problem:

$$y_p = sign\,(x_1\ x_2)\quad -1 < x_i < 1,\ i{=}1 \text{ to } 2 \tag{16}$$

rather than as a classification problem; since the classes are non overlapping, it is clear that r should tend to a value close to zero. An analysis of the problem shows that the smallest neural networks which are able to approach (16) with a good accuracy are a $MLP\,(2,\ 3^{tanh},\ 1^{tanh})$, and a $MLDCP\,(2,\ 2^{tanh},\ 1^{tanh})$. We trained the latter for ten different initializations of the weights uniformly in $[-0.17\ ;\ 0.17]$, the training being stopped for a TMSE of 10^{-15}. The Levenberg-Marquardt algorithm converged to 10^{-15} seven times out of ten. The proposed EKF algorithm with $p_0 = 10^{-2}$, $r_0 = 1.0$, $r_f = 10^{-40}$ and α in the range $[0.5;\ 2.0]$ led the TMSE to 10^{-15} nine times out of ten, and once to 8.0 10^{-3}, always leading to correct decision regions. The convergence of the different algorithms is shown on Fig. 9; again, increasing α speeds up the convergence of the proposed EKF algorithm.

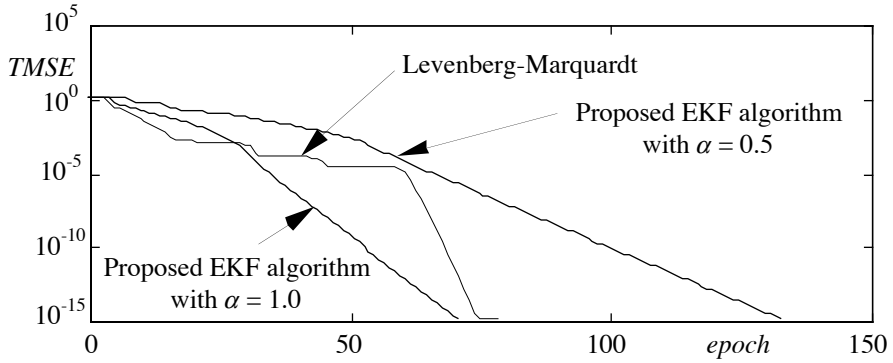

Fig. 9. Pseudo XOR problem: Evolution of the TMSE.

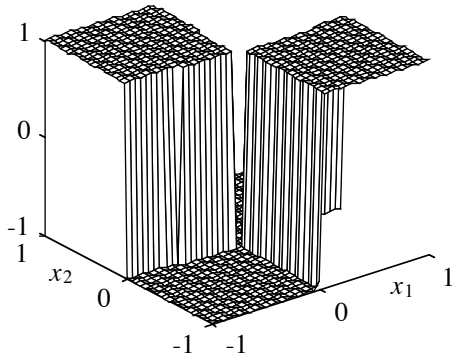

Fig. 10. Pseudo XOR problem: Function implemented by a $MLDCP\,(\ 2,\ 2^{tanh},\ 1^{tanh})$ trained with the proposed EKF algorithm and with a final TMSE of 10^{-15}.

The function implemented by a network trained with the proposed EKF algorithm is shown in Fig. 10: it is remarkable that the proposed EKF algorithm, which is based on the linearization of the function implemented by the neural model around the current weight estimate, converges successfully to a quasi discontinuous function.

5.3. Modeling a first-order NARX process (dynamic 2-input problem with noise)

This example deals with the modeling of a NARX process simulated by the following first-order single-input, single-output equation:

$$y_p(k) = f\left(y_p(k\text{-}1),\, u(k\text{-}1)\right) + w(k)$$
$$= \left(1 - \frac{0.1}{1+5y_p(k\text{-}1)^2}\right) y_p(k\text{-}1) + \frac{exp\left(-\frac{y_p(k\text{-}1)^2}{0.32}\right)}{1+5y_p(k\text{-}1)^2}\, u(k\text{-}1) - exp\left(-\frac{\left(y_p(k\text{-}1)\text{–}0.5\right)^2+\left(u(k\text{-}1)+0.5\right)^2}{0.02}\right) + w(k) \quad (17)$$

where $\{y_p(k)\}$ is the process output sequence, $\{u(k)\}$ the control input sequence, and $\{w(k)\}$ is a sequence of i.i.d. zero mean gaussian variables of variance $\sigma_w^2 = 10^{-3}$. The training control input sequence $\{u(k)\}$ consists of $N = 1000$ random variables with uniform distribution in $[-1\ ;\ 1]$. A sequence for performance estimation of 1000 sampling periods is generated in the same manner. A contour plot of the function to be estimated, and the inputs of the training set are shown on Fig. 11a).

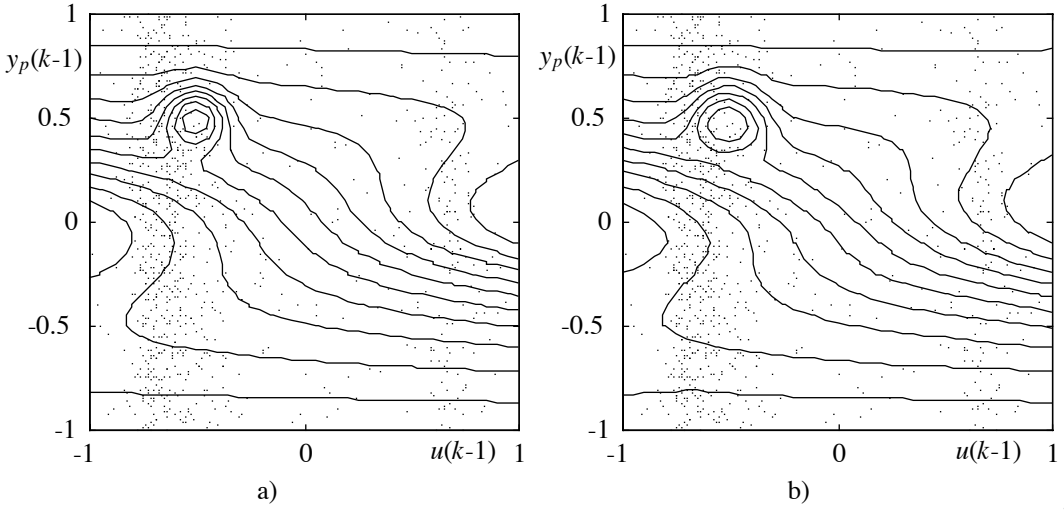

Fig. 11. First-order NARX process problem: Training set inputs and contour plots of: a) Function to be estimated; b) Function implemented by a $FCP\left(2, 7^{tanh}, 1^{linear}\right)$ trained with the proposed EKF algorithm.

The function f being complex, $FCP\left(2, n_h^{tanh}, 1^{linear}\right)$ are used, whose weights are initialized uniformly in $[-0.17\ ;\ 0.17]$. The comparison of the algorithms is performed for $n_h = 7$, the optimal number of neurons found with the Stepwise Modeling Procedure using the proposed EKF algorithm with $p_0 = 10^{-2}$, $r_0 = 0.4$, $r_f = 10^{-40}$, and α in $[0.1\ ;\ 2.0]$: the TMSE equals $9.3\ 10^{-4}$, and the PMSE $1.1\ 10^{-3}$. The function obtained with $\alpha = 0.5$ is shown on Fig. 11b). The Levenberg-Marquardt algorithm leads to less satisfactory results, with a TMSE of $3.2\ 10^{-3}$ and a corresponding PMSE of $4.3\ 10^{-3}$. These results do not vary for different initializations of the weights. The evolution of the TMSE for the different algorithms is shown on Fig. 12.

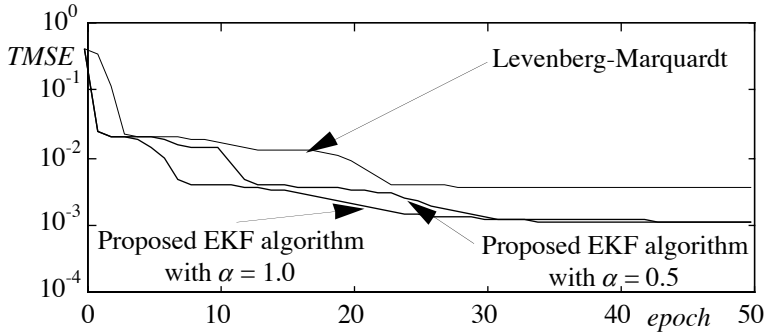

Fig. 12. First-order NARX process problem: Evolution of the TMSE.

5.4. Modeling a third-order deterministic process (dynamic 5-input benchmark problem without noise)

This last example deals with the modeling of a nonlinear simulated process given in [8], and which was also modeled in [11]. The process is described by a deterministic third-order single-input, single-output model:

$$y_p(k) = \frac{y_p(k\text{-}1)\, y_p(k\text{-}2)\, y_p(k\text{-}3)\ \ \left(y_p(k\text{-}3) - 1\right) u(k\text{-}2) + u(k\text{-}1)}{1 + y_p(k\text{-}2)^2 + y_p(k\text{-}3)^2} \tag{18}$$

where $\{y_p(k)\}$ is the process output sequence, and $\{u(k)\}$ the control input sequence. The training control input sequence $\{u(k)\}$ consists of $N = 1000$ random variables with a uniform distribution in $[-1\ ;\ 1]$. A similar input sequence is chosen for performance estimation. As in [11], a $MLP\left(5,\ 12^{tanh},\ 8^{tanh},\ 4^{tanh},\ 1^{linear}\right)$ is used.

In [11], the EKF is used with a diagonal matrix Q whose components are small and non negative and $r = \text{cte} = 1$, and leads to an average TMSE of 1.2 10^{-3}. Note that [11] uses an economic - decoupled - variant of the EKF, which might contribute to the lower quality of the results.

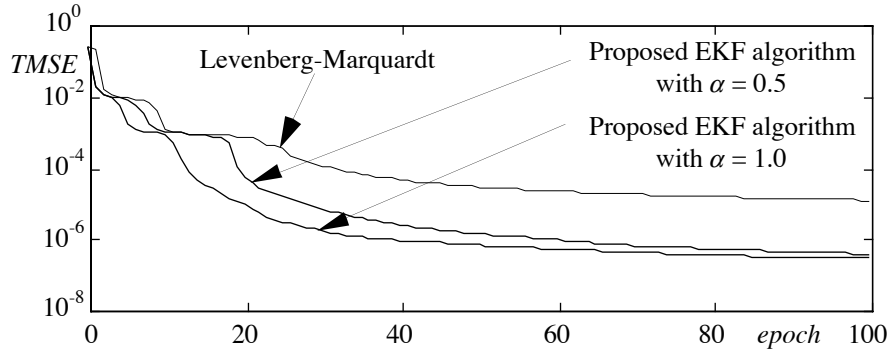

Fig. 13. Third-order deterministic process problem: Evolution of the TMSE.

With our approach, since the process is noise free, it is clear that r should tend to a value close to zero. The proposed EKF algorithm is used with $p_0 = 10^{-2}$, $r_0 = 0.2$, $r_f = 10^{-40}$, α in $[0.1\ ;\ 2.0]$, and the initial weights in $[-0.17\ ;\ 0.17]$. As shown on Fig. 13, a TMSE of 2.6 10^{-7} and a PMSE of 5.9 10^{-7} are reached, whereas the results obtained using a Levenberg-Marquardt algorithm are less satisfactory. These results do not vary for different initializations of the weights. Note that a $FCP\left(5,\ 15^{tanh},\ 1^{linear}\right)$ trained with the proposed EKF algorithm leads to the same performance as the above MLP.

5.5. Discussion

The preceding examples confirm the following facts concerning the choice of the EKF parameters we propose:

– the initialization of the weights of the neural network θ_0 according to the same criteria as when training the neural network with a PEM method and the choice of a small ratio $\frac{p_0}{r_0}$ prevent from too large parameter changes when the algorithm is started;
– the choice of r decreasing between epochs from the initial TMSE to a value close to zero with an exponential decay factor α of the order of 1 leads with robustness to a satisfactory minimum of the cost-function (4); increasing α up to a problem-dependent limit speeds up the convergence.

6. Conclusion

The simulation examples we have presented illustrate the analysis of the influence of the EKF algorithm parameters made in section 4, and show that the choices we propose lead to excellent results. The comparison made with the Levenberg-Marquardt algorithm demonstrate the ability of the proposed EKF algorithm to perform at least as good as the most efficient non recursive second-order algorithms. Finally, the results obtained on benchmark problems show its superiority over EKF variants which have been proposed in the past.

A straightforward extension of this work will be to consider multiple-output problems, i.e. the scalar r becomes a matrix. Another promising extension will be to exploit the regularizing power of r, both in the single- and in the multiple-output case.

Acknowledgements

We thank the anonymous reviewers for their very careful reading of the manuscript and their constructive comments.

References

[1] Bishop M. *Neural Networks for Pattern Recognition* (Clarendon Press, Oxford, 1985), 338-346.

[2] Goodwin G. C., Sin K. S. *Adaptive filtering prediction and control* (Prentice-Hall, New Jersey, 1984), 293-294.

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

[4] Jin L., Nikiforuk P. N., Gupta M. M. Weight-decoupled Kalman filter learning algorithm of multi-layered neural networks, *Neural Network World* **1/95** (1995), 51-70.

[5] Ljung L. Asymptotic behavior of the extended Kalman filter as a parameter estimator for linear systems, *IEEE Trans. on Automatic Control* **24** (1979), 36-50.

[6] Ljung L., Söderström T. *Theory and practice of recursive identification*, (MIT Press, Cambridge, 1987), 54-56.

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

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

[9] Nerrand O., Roussel-Ragot P., Personnaz L., Dreyfus G. Training recurrent neural networks: why and how ? An illustration in process modeling, *IEEE Trans. on Neural Networks* **5** (1994), 178-184.

[10] Press W. H., Teukolsky S. A., Vetterling W. T., Flannery B. P. *Numerical Recipes in C* (Cambridge, University Press, 1992), 684.

[11] Puskorius G. V., Feldkamp L. A. Decoupled extended Kalman filter training of feedforward layered networks, *Proceedings of IJCNN'91 I*, Seattle (1991), 771-777.

[12] Puskorius G. V., Feldkamp L. A. Neurocontrol of nonlinear dynamical systems with Kalman Filter trained recurrent networks, *IEEE Trans. on Neural Networks* **5** (1994), 279-297.

[13] Rivals I., Personnaz L. Black-box modeling with state-space neural networks, in *Neural Adaptive Control Technology I*, R. Zbikowski and K. J. Hunt eds. (World Scientific, 1996), 237-264.

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

[15] Williams R. J. Training recurrent networks using the extended Kalman filter, *Proceedings of IJCNN'92 IV*, Baltimore (1992), 241-246.