Optimized Learning with Bounded Error for Feedforward Neural Networks

A. Alessandri, M. Sanguineti, and M. Maggiore

Abstract

A learning algorithm for feedforward neural networks is presented that is based on a parameter estimation approach. The algorithm is particularly well-suited for batch learning and allows one to deal with large data sets in a computationally efficient way. An analysis of its convergence and robustness properties is made. Simulation results confirm the effectiveness of the algorithm and its advantages over learning based on backpropagation and extended Kalman filter.

Keywords

Feedforward neural networks, learning algorithms, batch learning, nonlinear approximation, optimization.

I. Introduction

After the development of backpropagation (BP, for short) [1], plenty of algorithms have been proposed to train feedforward neural networks. Although BP has been successfully applied in a variety of areas (see, e.g., [2]-[10]), its convergence is slow, thus making high-dimensional problems intractable. Its slowness is to be ascribed to the use of the steepest–descent method, which performs poorly in terms of convergence in high-dimensional settings [11], and to the fixed, arbitrarily chosen step length. For these reasons, algorithms using also the second derivatives have been developed (see, e.g., [12]) and modifications to BP have been proposed (see, e.g., the acceleration technique presented in [13] and the approach described in [14], which is aimed at restoring the dependence of the learning rate on time). The determination of the search direction and of the step length by using methods of nonlinear optimization has been considered, for example, in [15]. The effects of adding noise to inputs, outputs, weights connections and weights changes during backpropagation training have also been studied (see, e.g., [16]).

Further insights can be gained by regarding the learning of feedforward neural networks as a parameter estimation problem. Following this approach, training algorithms based on the extended Kalman filter (EKF, for short) have been proposed (see, e.g., [17]-[19]) that show faster convergence than BP and do not need the tuning of parameters. However, the advantages of EKF-based trainings are obtained at the expense of a notable computational burden (as matrix inversions are required) and a large amount of memory.

In this paper, the learning of feedforward neural networks is regarded as a parameter estimation problem. Some recent results concerning nonlinear estimation [10] are exploited to develop a novel learning algorithm. Its convergence and robustness properties are investigated. Unlike EKF–based training, the proposed algorithm does not require matrix inversions. It is based on the minimization of a cost function composed of two contributions: a fitting penalty term and a term related to changes in the parameters. Bounds on the rate of convergence of the algorithm are provided and its robustness against errors in the minimization of the cost function is investigated. Simulation results show that the optimization of the proposed cost function outperforms backpropagation-based algorithms. The learning based on the extended Kalman filter yields comparable results but involves a much higher computational load. Moreover, as the proposed algorithm works according to a sliding–window scheme, only a portion of the data is processed at each time (the past is summarized in one prediction), thus making the method computationally tractable.

The paper is organized as follows. Section II defines neural-network learning as a parameter estimation problem and describes the basic algorithm. Its convergence and robustness properties are investigated in Sections III and IV, respectively. A formulation of the algorithm tailored to batch learning is illustrated in Section V. Section VI reports simulation results comparing the proposed approach with BP-based and EKF-based learnings.

A. Alessandri is with the Naval Automation Institute (IAN–CNR), National Research Council of Italy, Via De Marini 6, 16149 Genova, Italy (e-mail: angelo@ian.ge.cnr.it).

M. Sanguineti is with Department of Communications, Computer and System Sciences (DIST), University of Genoa, Via Opera Pia 13, 16145 Genova, Italy (e-mail: marcello@dist.unige.it).

M. Maggiore is with the Department of Electrical and Computer Engineering, University of Toronto, 10 King's College Rd., M5S 3G4 Toronto, Canada (e-mail: maggiore@control.toronto.edu).

A. Alessandri and M. Sanguineti were partially supported by the MURST Project "New Techniques for the Identification and Adaptive Control of Industrial Systems" and by the CNR-Agenzia 2000 Project "New Algorithms and Methodologies for the Approximate Solution of Nonlinear Functional Optimization Problems in a Stochastic Environment."

Published in *IEEE Transactions on Neural Networks*, vol. 13, no. 2, 2002, pp. 261–273

II. Algorithm description

We consider *feedforward neural networks* (in the following, for the sake of brevity, often called "neural networks" or simply "networks,") composed of L layers, with ν_s computational units in the layer s ($s = 1, \ldots, L$). The input–output mapping of the q-th unit of the s-th layer is given by

$$\begin{aligned} y_q(s) &= g\left[\sum_{p=1}^{\nu_{s-1}} w_{pq}(s) y_p(s-1) + w_{0q}(s)\right], \\ s &= 1, \ldots, L\,;\, q = 1, \ldots, \nu_s \end{aligned} \tag{1}$$

where $g : \mathbb{R} \to \mathbb{R}$ is called *activation function*. The coefficients $w_{pq}(s)$ and the so-called *biases* $w_{0q}(s)$ are lumped together into the weights vectors $\underline{w}^s$. We let

$$\underline{w} \stackrel{\triangle}{=} \mathrm{col}\left(\underline{w}^1, \underline{w}^2, \ldots, \underline{w}^L\right) \in W \subset \mathbb{R}^n$$

where

$$n \stackrel{\triangle}{=} \sum_{s=0}^{L} \nu_{s+1}\,(\nu_s + 1)$$

is the total number of weights. The function implemented by a feedforward neural network with weights vector $\underline{w}$ is denoted by $\underline{\gamma}(\underline{w}, \underline{u})$, $\underline{\gamma} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^p$, where $\underline{u} \in U \subset \mathbb{R}^m$ is the network input vector.

The data set consists of input/output pairs $(\underline{u}_t, \underline{y}_t)$, $t = 0, 1, \ldots, P-1$, where $\underline{u}_t \in U \subset \mathbb{R}^m$ and $\underline{y}_t \in Y \subset \mathbb{R}^p$ represent the input and the desired output of the neural network at the time t, respectively (hence, $\nu_0 = m$ and $\nu_L = p$). If one assumes the data to be generated by a sufficiently smooth function $\underline{f} : \mathbb{R}^m \longrightarrow \mathbb{R}^p$, i.e., $\underline{y}_t = \underline{f}(\underline{u}_t)$ (suitable smoothness hypotheses on f can be assumed according to the process generating the data), then the approximation properties of feedforward neural networks guarantee the existence of a weights vector $\underline{w}^* \in W \subset \mathbb{R}^n$ such that

$$\underline{f}(\underline{u}) = \underline{\gamma}(\underline{w}^*, \underline{u}) + \underline{\eta}\,,\ \forall \underline{u} \in U \tag{2}$$

where $\underline{\eta}$ is the network approximation error and $U \subset \mathbb{R}^m$ (see, e.g., [20]-[23]). Now, let $\underline{\eta}_t$ be the error made in approximating $\underline{f}$ by the neural network implementing the mapping $\underline{\gamma}$, for the input $\underline{u}_t$. Such a network can be represented, for $t = 0, 1, \ldots, P-1$ as

$$\begin{cases} \underline{w}_{t+1} = \underline{w}_t \\ \underline{y}_t = \underline{\gamma}\left(\underline{w}_t, \underline{u}_t\right) + \underline{\eta}_t \end{cases} \tag{3}$$

where the network weights play the role of a constant state equal to the "ideal" network weights vector $\underline{w}^*$, i.e., $\underline{w}_0 = \underline{w}_1 = \ldots = \underline{w}_P \stackrel{\triangle}{=} \underline{w}^*$, and $\underline{\eta}_t \in K \subset \mathbb{R}^p$ is a noise. Equations (3) allows one to regard the supervised learning of feedforward neural networks according to a set of data as the problem of estimating the state of a nonlinear system. The measurement equation defines the nonlinear relationship among inputs, outputs, and weights according to the function $\underline{\gamma}$ implemented by the network.

The representation (3) will be used in the following as the departure point to derive a training algorithm as a recursive state estimator for the system representing the network. More precisely, the algorithm will be obtained as a sliding-window state estimator for system (3), with an estimation cost defined as

$$\begin{aligned} J_t &\stackrel{\triangle}{=} \mu \left\|\underline{\hat{w}}_t - \underline{\bar{w}}_t\right\|^2 + \sum_{i=t-N}^{t} \left\|\underline{y}_i - \underline{\gamma}\left(\underline{\hat{w}}_t, \underline{u}_i\right)\right\|^2\,, \\ & t = N, N+1, \ldots, P-1 \end{aligned} \tag{4}$$

where $\|.\|$ denotes the Euclidean norm, $\mu > 0$ is a scalar whose choice will be discussed later on, $\underline{\hat{w}}_t$ is the estimate of the "ideal" weights vector $\underline{w}^*$ at the time t, and $\underline{\bar{w}}_t = \underline{\hat{w}}_{t-1}$ is called *prediction*. The estimator has a *finite memory*, i.e., the estimation of the network weights relies on the last N input/output patterns. The minimization of J_t at each temporal stage is based on the *information vector*, expressed as

$$\begin{aligned} \underline{I}_t &\stackrel{\triangle}{=} \mathrm{col}\left(\underline{\bar{w}}_t^{\circ}, \underline{y}_{t-N}, \ldots, \underline{y}_t, \underline{u}_{t-N}, \ldots, \underline{u}_t\right) \\ &\in \mathbb{R}^{n+(N+1)(p+m)}\,,\ t = N, N+1, \ldots, P-1 \end{aligned} \tag{5}$$

where $\underline{\bar{w}}_t^{\circ} \stackrel{\triangle}{=} \underline{\hat{w}}_{t-1}^{\circ}$ denotes the prediction (see the formulation of the learning algorithm below); $\underline{\bar{w}}_N^{\circ}$ represents the "*a priori*" prediction, i.e., the weights values before training. The learning algorithm is obtained as a sequential state estimator for system (3) on the basis of the minimization of cost (4).

Algorithm OL (Optimized Learning). *Given the information vector* $\underline{I}_t$*, determine for each* $t = N, N+1, \ldots$*, the estimate* $\hat{\underline{w}}_t^{\circ}(\underline{I}_t)$ *of the network weights by minimizing cost (4).*

□

When a new input/output pattern $(\underline{u}_t, \underline{y}_t)$ becomes available, the information vector

$$\underline{I}_{t+1} \stackrel{\triangle}{=} \mathrm{col}\big(\bar{\underline{w}}_{t+1}^{\circ}, \underline{y}_{t-N+1}, \ldots, \underline{y}_{t+1}, \underline{u}_{t-N+1}, \ldots \underline{u}_{t+1}\big)$$

is used to generate the new estimate $\hat{\underline{w}}_{t+1}^{\circ}(\underline{I}_{t+1})$ of the "ideal" weights vector $\underline{w}^*$.

As regards the choice of the positive scalar μ, it should be noted that it expresses the relative importance of the prediction with respect to the last $N+1$ patterns. Therefore, if μ is increased, the first term of J_t attaches an increasing importance to the prediction $\bar{\underline{w}}_t$. This turns out to be useful in the case of large noises and accurate initial prediction. The minimization of a cost function with the structure of (4) has been first proposed in [10] to derive a sliding-window estimator for a class of nonlinear dynamic systems.

Note that the mapping $\underline{\gamma}$ is invariant to certain permutations of the weights [26]. Nevertheless, we only require the estimator to find one of the weights vectors that minimize cost (4): all the weights vectors obtained through permutations of a given vector can be regarded as being equivalent.

In Section III, we shall investigate the convergence of algorithm OL, supposing that the minimization of cost (4) is performed exactly. In Section IV, we shall show that the algorithm is robust to errors on the minimization of cost (4). The formulation of the algorithm within a batch–learning context will be considered in Section V.

III. Convergence analysis

Let

$$\underline{H}\left(\underline{w}, \underline{u}_{t-N}^{t}\right) \stackrel{\triangle}{=} \begin{bmatrix} \underline{\gamma}\left(\underline{w}, \underline{u}_{t-N}\right) \\ \underline{\gamma}\left(\underline{w}, \underline{u}_{t-N+1}\right) \\ \vdots \\ \underline{\gamma}\left(\underline{w}, \underline{u}_{t}\right) \end{bmatrix} \in \mathbb{R}^{p(N+1)} \tag{6}$$

where $\underline{u}_{t-N}^{t} \stackrel{\triangle}{=} \mathrm{col}\left(\underline{u}_{t-N}, \underline{u}_{t-N+1}, \ldots, \underline{u}_{t}\right) \in U^{N+1}$ (recall that $\underline{u}_t \in U \subset \mathbb{R}^m$). Similarly, let $\underline{y}_{t-N}^{t} \stackrel{\triangle}{=} \mathrm{col}\left(\underline{y}_{t-N}, \underline{y}_{t-N+1}, \ldots, \underline{y}_{t}\right)$. Given a square matrix A, $\lambda_{\min}(A)$ and $\lambda_{\max}(A)$ denote its minimum and maximum eigenvalues, respectively. For a generic matrix B, let $\|B\| \stackrel{\triangle}{=} \sqrt{\lambda_{\max}(B^T B)}$ and $\|B\|_{\min} \stackrel{\triangle}{=} \sqrt{\lambda_{\min}(B^T B)}$.

From now on we suppose that the network activation function in (1) belongs to class $C^1(\mathbb{R})$ and, using the notation of [10], we define the matrix

$$D(\underline{w}, \underline{u}_{t-N}^{t}) \stackrel{\triangle}{=} \frac{\partial \underline{H}}{\partial \underline{w}} \in \mathbb{R}^{p(N+1)\times n}$$

and the scalars

$$\delta \stackrel{\triangle}{=} \min_{\underline{w}\in W,\, \underline{u}_{t-N}^{t} \in U^{N+1}} \|D(\underline{w}, \underline{u}_{t-N}^{t})\|_{\min},$$

$$\Delta \stackrel{\triangle}{=} \max_{\underline{w}\in W,\, \underline{u}_{t-N}^{t} \in U^{N+1}} \|D(\underline{w}, \underline{u}_{t-N}^{t})\|.$$

Moreover, we let

$$r_\eta \stackrel{\triangle}{=} \max_{\underline{\eta}_{t-N}, \ldots, \underline{\eta}_t \in K} \|\mathrm{col}\left(\underline{\eta}_{t-N}, \ldots, \underline{\eta}_t\right)\|$$

and

$$\bar{k} \stackrel{\triangle}{=} k\sqrt{N+1}$$

where $k > 0$ is a scalar such that

$$\left\| \frac{\partial \underline{\gamma}(\underline{w}, \underline{u})}{\partial \underline{w}}(\underline{w}', \underline{u}) \;-\; \frac{\partial \underline{\gamma}(\underline{w}, \underline{u})}{\partial \underline{w}}(\underline{w}'', \underline{u}) \right\| \le k\, \|\underline{w}' - \underline{w}''\|$$
$$\forall\, \underline{w}', \underline{w}'' \in W\,,\ \forall \underline{u} \in U.$$

Finally, for $\xi \in \mathbb{R}$, let us consider the inequalities:

$$\left(\delta^4 - 8\bar{k}\Delta^2 r_\eta\right)\mu + \delta^6 > 0 \tag{7}$$

$$\mu^2 + 2\left(\delta^2 - \bar{k}\,\Delta\,\xi\right)\mu + \delta^4 - 2\bar{k}\,\Delta^2\, r_\eta > 0 \tag{8}$$

$$2\,\mu^2 + \left(4\,\delta^2 - 3\,\bar{k}\,\Delta\,\xi\right)\mu + 2\,\delta^4 - 2\,\bar{k}\,\Delta^2\, r_\eta - \delta^2\,\bar{k}\,\Delta\,\xi > 0 \tag{9}$$

and the second-order equation in ξ

$$2\Delta\bar{k}\mu^2\,\xi^2 + \left[4\bar{k}\Delta^2\mu r_\eta - \delta^2(\mu+\delta^2)^2\right]\xi + 2\bar{k}\Delta^3 r_\eta^2 + \Delta(\mu+\delta^2)^2 r_\eta = 0 \quad . \tag{10}$$

The paramters δ and Δ can be estimated on the basis of the sets W (representing a constraint on the size of the weights) and U (representing a constraint on the size of the inputs). The parameter k can be estimated in terms of the set W, the set U, and the number of computational units in the network.

r_η represents the maximum approximation error incurred by the network $\gamma(\underline{w}^*, \underline{u})$ in approximating the mapping $\underline{f}$ that generates the data (suitable hypotheses about the smoothness of such a mapping can be made on the basis of the process generating the data). As regards the possibility of estimating the value of r_η, there exist many results expressing the approximation error as a function of the number of computational units in the network. For the sake of simplicity and without loss of generality, let us consider the case of a one-hidden layer network with ν computational units in the hidden layer and a data set of input/output pairs $(\underline{u}_t, \underline{y}_t)$ generated by a mapping $\underline{f} : \mathbb{R}^m \longrightarrow \mathbb{R}$. For a large variety of smoothness properties of the mapping f generating the data, the existence of an upper bound of the order of $O\left(\frac{1}{\sqrt{\nu}}\right)$ on the approximation error has been proved (see, e.g., [32], [27], [28], [29], and [30]). Exploiting such results, once the number ν of computational units in the network has been fixed, one can obtain an estimate from above of the approximation error, hence of r_η.

The following proposition is obtained by applying [10, Theorem 1] to system (3).

Proposition 1 (Bounds on the weights estimation error by algorithm OL). *Suppose that there exists an integer N such that, for any $\underline{u}_{t-N}^t \in U^{N+1}$, $\operatorname{rank} D(\underline{w}, \underline{u}_{t-N}^t) = n$. Let $\underline{\hat{w}}_t^\circ$ be the estimate of the weights vector $\underline{w}^*$ at stage t obtained by applying algorithm OL, and let $\underline{\hat{e}}_t \stackrel{\triangle}{=} \underline{w}^* - \underline{\hat{w}}_t^\circ$ be the weights estimation error at stage t. Consider the largest closed ball $N(\hat{r}_e)$ of radius $\hat{r}_e$ and centered in the origin, such that $\underline{\hat{e}}_N \in N(\hat{r}_e)$. If there exists a choice of μ satisfying (7), then the second-order equation (10) has two real positive roots, $\xi^- < \xi^+$. If μ satisfies also (8) and (9) with $\zeta \in (\xi^-, \xi^+)$, then the following holds for the network's learning by algorithm OL:*

(i) $\hat{r}_e \leq \zeta \implies \|\underline{\hat{e}}_t\| \leq \xi_t \qquad t = N, N+1, \ldots, P-1$

(boundedness of the estimation error)

(ii) $\hat{r}_e \leq \zeta \implies \lim_{t\to+\infty} \|\underline{\hat{e}}_t\| \leq \xi^-$,

(asymptotic bound on the estimation error for $P \to \infty$)

where $\xi_N \stackrel{\triangle}{=} \hat{r}_e$ and ξ_t, $t = N, N+1, \ldots$, is a monotonically decreasing sequence given by

$$\xi_t = \frac{2\Delta\bar{k}\mu^2}{(\mu+\delta^2)^3}\xi_{t-1}^2 + \mu\frac{[(\mu+\delta^2)^2 + 4\bar{k}\Delta^2 r_\eta]}{(\mu+\delta^2)^3}\xi_{t-1} + \frac{2\bar{k}\Delta^3 r_\eta^2 + \Delta(\mu+\delta^2)^2 r_\eta}{(\mu+\delta^2)^3} \; .$$

□

Some remarks can be made about the way how the various quantities contained in equations (7) to (10) influence the bounds on the weights estimation error associated with algorithm OL. Towards this end, let $\bar{k}$ and Δ take on some fixed values, and let us study the influence of δ and r_η. If $\delta^4/r_\eta \geq 8\bar{k}\Delta^2$, then (7) is satisfied for any $\mu \geq 0$; if $\delta^4/r_\eta < 8\bar{k}\Delta^2$, then (7) is fulfilled for $\mu \in (\mu^+, +\infty)$, where $\mu^+ = \delta^6/(8\bar{k}\Delta^2 r_\eta - \delta^4)$. The analyses of (8) and (9) are less easy, for they depend on the roots of (10), i.e., ξ^- and ξ^+, which, in turn, depend on μ. However, in general, a higher value of the ratio δ^4/r_η determines a wider range for μ, in which (8) and (9) are satisfied. The situation can be summarized as follows. Larger values of r_η are allowed by larger values of δ, such that inequalities (7), (8), and (9) are satisfied for some range of values of μ. As ξ^- and ξ^+ depend on μ, this parameter has to be chosen such as to obtain the largest possible value of ξ^+ (recall that $\hat{r}_e \leq \zeta < \xi^+$) and the smallest possible value of ξ^- (recall that ξ^- is an asymptotic upper bound on the weights estimation error). In other words, the choice of μ should allow the smallest possible upper bound ξ^- on the asymptotic weights estimation error and the largest possible upper bound ξ^+ on the initial weights estimation error.

Let us discuss the requirement that, for any $\underline{u}_{t-N}^t \in U^{N+1}$, the matrix $D(\underline{w}, \underline{u}_{t-N}^t)$ be of rank n. As, by definition,

$$D\left(\underline{w}, \underline{u}_{t-N}^t\right) = \frac{\partial \underline{H}}{\partial \underline{w}}\left(\underline{w}, \underline{u}_{t-N}^t\right) = \begin{bmatrix} \frac{\partial \underline{\gamma}}{\partial \underline{w}}\left(\underline{w}, \underline{u}_{t-N}\right) \\ \frac{\partial \underline{\gamma}}{\partial \underline{w}}\left(\underline{w}, \underline{u}_{t-N+1}\right) \\ \vdots \\ \frac{\partial \underline{\gamma}}{\partial \underline{w}}\left(\underline{w}, \underline{u}_t\right) \end{bmatrix}$$

$$\in \mathbb{R}^{p(N+1)\times n} \, ,$$

the bigger $p(N+1)$, the "easier" for the above matrix to be of rank n. In this context, the condition $p(N+1) \gg n$ (i.e., choosing a large window size in algorithm OL, as compared with the size of the weights vector), used in the field of parameter estimation, can be related to the behavior of the error of a network with n parameters (the weights), trained to learn an unknown mapping on the basis of P input/output pairs. Recall that the overall error between the network and the target function, called *generalization error*, consists of two contributions (see, e.g., [24], [25]). The first one is the *approximation error*, which refers to the distance between the target function and the closest neural–network function of a given architecture. The second contribution is the *estimation error*, which measures the distance between the best network function (obtained by an ideally infinite data set) and the function estimated on the basis of the P training data (recall the $P \geq N+1$). In practice, the generalization error, i.e., the ability of the network to "generalize" to new data (not observed as not belonging to the training set), stems from a tradeoff between the accuracy of the approximation and the accuracy of the empirical fit to the theoretical approximation (see, e.g., [24], [25]).

Note that also the choice of N is important for the fulfilment of the condition on rank D.

Finally, we remark that the choice of μ is crucial for the behavior of the estimate. In particular, small values of μ imply a low dependence of the estimate on the prediction, whereas larger values of μ result in a higher dependence and in a smaller influence of new input/output patterns on the estimate. The problem of a suitable choice of μ for different noise levels will be addressed in detail in Section VI by means of extensive simulations.

IV. Robustness analysis

The minimization of cost (4) can be performed by using a nonlinear programming iterative algorithm. We are interested in evaluating the robustness of algorithm OL when such a minimization is not performed exactly. Let $\underline{\tilde{w}}_t^\circ$ be the estimate of the "ideal" network weights vector $\underline{w}^*$ (see (2)) obtained by algorithm OL when the minimization of cost (4) is not exactly accomplished. Setting

$$\underline{\varepsilon}_t \stackrel{\triangle}{=} \underline{\hat{w}}_t^\circ - \underline{\tilde{w}}_t^\circ,$$

the learning process can be considered to be perturbed by $\underline{\varepsilon}_t$, which causes a "perturbed estimation error" $\underline{\tilde{e}}_t \stackrel{\triangle}{=} \underline{w}^* - \underline{\tilde{w}}_t^\circ = \hat{e}_t + \underline{\varepsilon}_t$. Thus, the estimate $\underline{\tilde{w}}_t^\circ$ of the network's "ideal" weights $\underline{w}^*$ is obtained, rather than by using $\underline{I}_t^N$ (see (5)), by relying on the "modified information vector"

$$\begin{aligned}\underline{\tilde{I}}_t \stackrel{\triangle}{=} \mathrm{col}\big(&\underline{\bar{\tilde{w}}}_t^\circ, \underline{y}_{t-N}, \ldots, \underline{y}_t, \underline{u}_{t-N}, \ldots, \\ &\underline{u}_t\big) \in \mathbb{R}^{n+(N+1)(p+m)} \quad t = N, N+1, \ldots, P-1\end{aligned}$$

where $\underline{\bar{\tilde{w}}}_t^\circ \stackrel{\triangle}{=} \underline{\tilde{w}}_{t-1}^\circ$ denotes the prediction.

Let us consider the inequalities:

$$\mu + \delta^2 - \bar{k}\Delta\bar{\varepsilon} > 0 \tag{11}$$

$$-8\bar{k}\Delta\,\bar{\varepsilon}\,\mu^2 + (\delta^4 - 8\bar{k}\Delta^2 r_\eta - 16\bar{k}\Delta\delta^2\bar{\varepsilon})\,\mu + \delta^6 > 0 \tag{12}$$

$$\begin{aligned}\mu^2 + \Big(2\delta^2 - 2\bar{k}\Delta\,\tilde{\xi} - 4\bar{k}\Delta\bar{\varepsilon}\Big)\,\mu + \delta^4 - 2\bar{k}\Delta^2\,r_\eta \\ -4\bar{k}\Delta\delta^2\,\bar{\varepsilon} > 0\end{aligned} \tag{13}$$

$$\begin{aligned}2\,\mu^2 + \Big(4\,\delta^2 - 3\,\bar{k}\Delta\,\tilde{\xi} - 5\,\bar{k}\Delta\bar{\varepsilon}\Big)\,\mu + 2\,\delta^4 - 2\,\bar{k}\Delta^2\,r_\eta \\ -5\bar{k}\Delta\delta^2\,\bar{\varepsilon} - \delta^2\bar{k}\Delta\,\tilde{\xi} > 0\end{aligned} \tag{14}$$

where $\bar{\varepsilon} \stackrel{\triangle}{=} \max_{t\in\{N,N+1,\ldots,P-1\}} \|\underline{\varepsilon}_t\|$ and $\xi \in \mathbb{R}$, and the second-order equation

$$\begin{aligned}&(2\bar{k}\Delta\mu^2)\,\tilde{\xi}^2 + \big[\,4\bar{k}\Delta^2\mu r_\eta + 8\mu\bar{k}\Delta(\mu+\delta^2)\bar{\varepsilon} \\ &\quad - \delta(\mu+\delta^2)^2\,\big]\,\tilde{\xi} + 2\bar{k}\Delta^3 r_\eta^2 + \Delta(\mu+\delta^2)^2 r_\eta \\ &\quad + 8\bar{k}\Delta(\mu+\delta^2)^2\bar{\varepsilon}^2 + (\mu+\delta^2)^3\bar{\varepsilon} \\ &\quad + 8\bar{k}\Delta^2(3\mu+2\delta^2)r_\eta\bar{\varepsilon} = 0 \quad .\end{aligned} \tag{15}$$

Applying [10, Theorem 2] to system (3), the following proposition is obtained.

Proposition 2 (Robustness of the weights estimate by algorithm OL). *Suppose that there exists an integer N such that, for any $\underline{u}_{t-N}^t \in U^{N+1}$, $\mathrm{rank}\, D(\underline{w}, \underline{u}_{t-N}^t) = n$. Let $\underline{\tilde{w}}_t^\circ$ be the estimate of the weights vector $\underline{w}^*$ at stage t obtained by applying algorithm OL with an approximate minimization of cost (4), and let $\underline{\tilde{e}}_t \stackrel{\triangle}{=} \underline{w}^* - \underline{\tilde{w}}_t^\circ$ be the weights estimation error at stage t. Consider the largest closed ball $N(\tilde{r}_e)$ of radius $\tilde{r}_e$ and centered in the origin, such that*

$\underline{\tilde{e}}_N \in N(\tilde{r}_e)$. *If there exists a choice of* μ *satisfying (11) and (12), then the second-order equation (15) has two real positive roots,* $\tilde{\xi}^- < \tilde{\xi}^+$. *If the choice of* μ *satisfies also (13) and (14) with* $\tilde{\zeta} \in (\tilde{\xi}^-, \tilde{\xi}^+)$, *then the following holds for the network's learning by algorithm OL:*

(i) $\tilde{r}_e \leq \tilde{\zeta} \implies \|\underline{\tilde{e}}_t\| \leq \tilde{\xi}_t \quad , \quad t = N, N+1, \ldots$

(boundedness of the perturbed estimation error)

(ii) $\tilde{r}_e \leq \tilde{\zeta} \implies \lim_{t \to +\infty} \|\underline{\tilde{e}}_t\| \leq \tilde{\xi}^-$

(asymptotic bound on the perturbed estimation error for $P \to \infty$*)*

where $\tilde{\xi}_N = \tilde{r}_e$ *and* $\tilde{\xi}_t$, $t = N, N+1, \ldots$, *is a monotonically decreasing sequence given by*

$$\begin{aligned}\tilde{\xi}_t = {} & \frac{2\mu^2 \bar{k}\Delta}{(\mu+\delta^2)^3}\,\tilde{\xi}_{t-1}^2 + \frac{1}{(\mu+\delta^2)^3}\Big[\mu(\mu+\delta^2)^2 + 4\mu\bar{k}\Delta^2 r_\eta \\ & + 8\mu\bar{k}(\mu+\delta^2)\bar{\varepsilon}\Big]\,\tilde{\xi}_{t-1} + \Big[2\bar{k}\Delta^3 r_\eta^2 + \Delta r_\eta(\mu+\delta^2)^2 \\ & + 8\bar{k}\Delta(\mu+\delta^2)^2\bar{\varepsilon}^2 + (\mu+\delta^2)^3\bar{\varepsilon} \\ & + 8\bar{k}\Delta^2 r_\eta(\mu+\delta^2)\bar{\varepsilon}\Big]\frac{1}{(\mu+\delta^2)^3}\,.\end{aligned}$$

□

According to Proposition 2, when the minimization required by algorithm OL is not performed exactly, the boundedness of the estimation error during the network training is preserved.

As regards the influence of the various scalars on the bounds on the weights estimation error, considerations similar to those after Proposition 1 can be made. In particular, the presence of $\bar{\varepsilon}$ in inequalities (11) to (14) results in a narrower range of admissible values of μ. Further, $\bar{\varepsilon} > 0$ in (15) determines values of the roots $\tilde{\xi}^-$ and $\tilde{\xi}^+$ that are, respectively, larger and smaller than the values of ξ^- and ξ^+ in equation (10). This means that an approximate minimization of cost (4) results in a larger upper bound on the asymptotic weights estimation error and reduces the allowed initial weights estimation error.

V. Batch learning

So far, sufficient conditions guaranteeing the existence of an upper bound on the weights estimation error have been provided, assuming that the temporal window in cost (4) moves one stage at a time. Let us now consider a temporal window moving d stages at a time, where $1 \leq d \leq N$. The corresponding estimator is called *d-step estimator*. We consider the following estimation cost (recall the definition of $\underline{H}$ given by (6))

$$\begin{aligned}J_{d(t-N)+N} = {} & \mu \left\| \underline{\hat{w}}_{d(t-N)+N} - \underline{\bar{w}}_{d(t-N)+N} \right\|^2 + \left\| \underline{y}_{d(t-N)}^{d(t-N)+N} \right. \\ & \left. - \underline{H}\left(\underline{\hat{w}}_{d(t-N)+N}, \underline{u}_{d(t-N)}^{d(t-N)+N}\right) \right\|^2 , \; t = N, N+1, \ldots, \bar{t}\end{aligned} \tag{16}$$

and the information vector

$$\begin{aligned}\underline{I}_{d(t-N)+N}^{N} = {} & \mathrm{col}\Big(\underline{\bar{w}}^{\circ}_{d(t-N)+N}, \underline{y}_{d(t-N)}^{d(t-N)+N}, \\ & \underline{u}_{d(t-N)}^{d(t-N)+N}\Big) \in \mathbb{R}^{n+(N+1)(p+m)} \; , \; t = N, N+1, \ldots, \bar{t}\end{aligned}$$

where $\bar{t}$ is the largest integer such that $\bar{t} + N + 1 \leq P$, $\underline{\bar{w}}^{\circ}_{d(t-N)+N} \stackrel{\triangle}{=} \underline{\hat{w}}^{\circ}_{d(t-N-1)+N}$ is the optimal prediction (see the formulation of the batch training algorithm below) and $\underline{\bar{w}}^{\circ}_{N}$ denotes the "*a priori*" prediction, i.e., the weights values before training.

Algorithm OBL (Optimized Batch Learning). *Given the information vector* $\underline{I}^N_{d(t-N)+N}$, *determine, for each* $t = N, N+1, \ldots$, *the estimate* $\underline{\hat{w}}^{\circ}_{d(t-N)+N}\big(\underline{I}^N_{d(t-N)+N}\big)$ *of the network weights by minimizing cost (16).*

□

The estimator makes use of an optimization window of size $N+1$, which slides d temporal stages at a time (see Fig. 1.

Since the definitions of the constants $\bar{k}, \delta, \Delta$, and r_η are the same as for the one-step estimator, Propositions 1 and 2 hold true also for the $d-$step estimator. Nevertheless, the performance of algorithm OBL varies on the basis of the different choices of d, as will be clarified by the numerical results reported in Section VI.

Given a fixed number of iterations, say t, the one-step estimator explores $N + t$ patterns of the data set, whereas the N-step estimator uses $N\,t + 1$ patterns (generally, the d-step estimator explores $N + 1 + (t-1)d$ patterns), as depicted in Fig. 2.

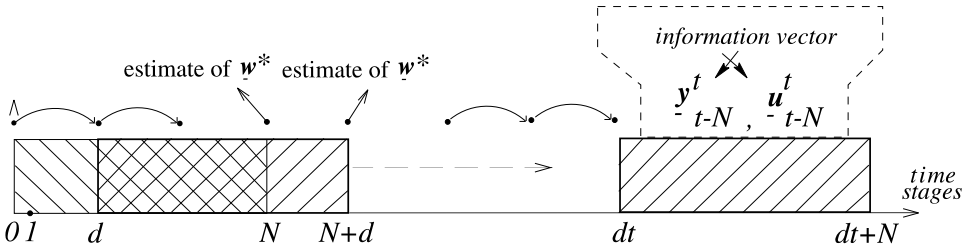


Fig. 1. Collection of the information vector for the d-step estimator.

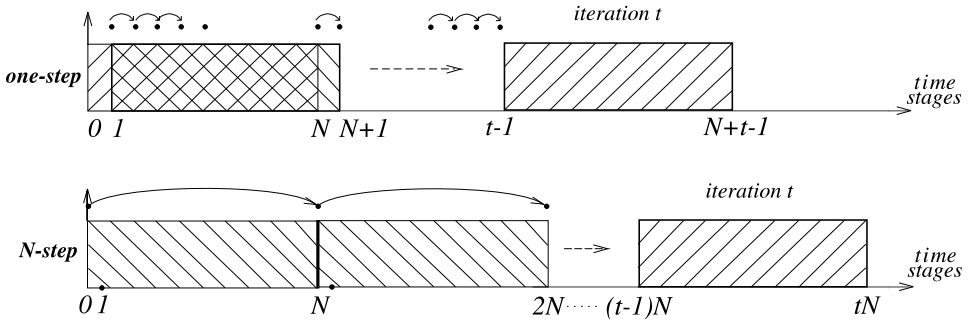


Fig. 2. Comparison between t iterations of the one-step and the N-step estimators ($t > N$).

Note that the N-step estimator maximizes the number of patterns explored with the same number of iterations. This turns out to be useful whenever the data set is very large and the application of gradient methods over an entire epoch becomes computationally unmanageable. Batch training overcomes the problem of a large data set by dividing the patterns into data batches of fixed length and by training the network with each data batch [31]. In this situation, the prediction term in cost (16) relates the current estimate of the weights vector to the last one and, consequently, the current batch to the previous one.

VI. Numerical results

In this section, the problem of identifying a nonlinear dynamic system on the basis of a set of input/output data is considered to evaluate the effectiveness of algorithm OBL in comparison with widely used training algorithms. Algorithm OBL is compared with classical backpropagation, with learning based on the extended Kalman filter, and with Levenberg-Marquardt training, all using data batches. More precisely, a sequence of input/output pairs is generated and all the above-mentioned algorithms are applied to fit the data according to a sliding-window strategy, for the same window size (i.e., the same batch length). In the following, the algorithms will be referred to as Back-Propagation Batch Learning (BPBL), Extended Kalman Filter Batch Learning (EKFBL), and Levenberg-Marquardt Batch Learning (LMBL). Both BPBL and LMBL are available in the Matlab Neural Networks Toolbox. EKFBL has been coded according to [18], using standard Matlab functions. OBL relies on the *fminu* routine of the Matlab Optimization Toolbox. The initial weights vector has been chosen by using the *initff* function of Matlab.

The input/output pairs of the dynamic system to be indentified represent a nonlinear dynamics describing the rotation of a mass in water with asymmetric drag coefficients. The system dynamics is expressed by:

$$\dot{\psi} = r \tag{17}$$

$$I_z \, \dot{r} = -b_l \, (r) \, r - b_s \, (r) \; r \; |r| + M \tag{18}$$

where ψ is the yaw angle, r is the yaw rate (i.e., the angular speed), M is the torque, and $I_z = 50.0 \, Kg \, m^2$ is the inertia. The linear and quadratic coefficients, $b_l \, (r)$ and $b_s \, (r)$, respectively, depend on the direction of rotation:

$$b_l \, (r) \triangleq \begin{cases} b_l^+ , & \text{for} \quad r \geq 0 \\ b_l^- , & \text{for} \quad r < 0 \end{cases} \tag{19}$$

and

$$b_s \, (r) \triangleq \begin{cases} b_s^+ , & \text{for} \quad r \geq 0 \\ b_s^- , & \text{for} \quad r < 0 \end{cases} \tag{20}$$

where $b_l^+ = 3.0 \, kg \, m^2 \, / \, sec$, $b_l^- = 10.0 \, kg \, m^2 \, / \, sec$, $b_s^+ = 5.0 \, kg \, m^2 / rad$, and $b_s^- = 10.0 \, kg \, m^2 / rad$. The measures of both ψ and r are corrupted by additive Gaussian random noises with zero mean and dispersions equal to σ_ψ

and σ_r. Moreover, an additive Gaussian random noise with zero mean and dispersion equal to σ_z acts on (18). If a simple Euler's discretization for a sample time equal to $T = 0.1$ sec is applied to (17) and (18), one obtains the pairs $[(\psi_i, r_i, M_i), (\psi_{i+1}, r_{i+i})]$, $i = 0, 1, \ldots$, where ψ_i, r_i, and M_i are the values of ψ, r, and M at the time iT, respectively. A pseudo-random binary sequence has been chosen for the torque, with maximum amplitude equal to $5\,N\,m$.

Three data sets (each made up of 5000 input/output pairs) have been generated for different levels of magnitude of the noise $\underline{\eta}_i \stackrel{\triangle}{=} \mathrm{col}\,(\eta_{1i}, \eta_{2i}, \eta_{3i})$, $i = 0, 1, \ldots$, where η_{1i}, η_{2i}, and η_{3i} are independently Gaussian distributed with dispersions σ_ψ, σ_r, and σ_z, respectively: a low noise level with $\sigma_\psi = 1\,\mathrm{deg}$, $\sigma_r = \sigma_z = 1\,\mathrm{deg/sec}$; a medium noise level with $\sigma_\psi = 2\,\mathrm{deg}$, $\sigma_r = \sigma_z = 2\,\mathrm{deg/sec}$; a high noise level with $\sigma_\psi = 5\,\mathrm{deg}$, $\sigma_r = \sigma_z = 5\,\mathrm{deg/sec}$. In the numerical simulations, all the algorithms have been applied to the same training set, given by the repetition of the 5000 input/output pairs for 100 times. This training set is well-suited to dealing with different operating conditions of algorithm OBL, i.e., different values of d and μ.

We have considered the following performance indices for the purpose of comparing the various algorithms. At each time instant $t = N, N+1, \ldots$, we have computed the summed squared approximation error (SSE) on the data. At the end of the training, we have computed both the final SSE and the total number of floating-point operations (expressed in MFOs, with 1 MFO = 10^6 floating-point operations). The termination criterion for the BPBL, LMBL, and OBL trainings is the decrease in the respective cost functions, i.e., the optimization routines end if the cost decreases less than 0.1% (this is not the case with the EKFBL algorithm, as it works without an optimization routine).

Two types of feedforward neural networks have been considered, with a linear output layer and one hidden layer having 2 and 5 hidden neurons, respectively, and the activation function $\tanh(\cdot)$. The diagonal covariance error matrix for the estimates of the optimal weights by the EKFBL algorithm has been initialized with the value 10^{-2} for all the weights and computed by using [18, p. 962, formulas (35) and (36), with $T_{\max}$ equal to 1000].

noise	$\mu = 1$		$\mu = 5$		$\mu = 10$		$\mu = 50$		$\mu = 100$	
level	**SSE**	**MFO**	**SSE**	**MFO**	**SSE**	**MFO**	**SSE**	**MFO**	**SSE**	**MFO**
low	4.89*	8975	5.62	8886	5.93	8873	6.24	8746	6.42	8645
medium	12.32	9044	11.86*	8920	12.13	8901	12.53	8808	12.59	8714
high	41.76	9259	41.61	9005	40.38	8951	39.99*	8855	40.39	8796

TABLE I

FINAL SSES AND MFOS OF THE OBL ALGORITHM WITH A 2–NEURON NETWORK FOR $d = 15$, $N = 15$, AND DIFFERENT CHOICES OF μ.

noise	$\mu = 1$		$\mu = 5$		$\mu = 10$		$\mu = 50$		$\mu = 100$	
level	**SSE**	**MFO**	**SSE**	**MFO**	**SSE**	**MFO**	**SSE**	**MFO**	**SSE**	**MFO**
low	3.94*	27906	4.76	27575	4.70	27463	4.78	26727	4.90	25973
medium	11.60	28371	10.89*	27817	10.90	27739	11.09	27196	11.29	26626
high	52.33	29826	41.12	28220	39.74	28094	39.14*	27485	39.83	27202

TABLE II

FINAL SSES AND MFOS OF THE OBL ALGORITHM WITH A 5–NEURON NETWORK FOR $d = 15$, $N = 15$, AND DIFFERENT CHOICES OF μ.

Tables I and II give the results obtained by the OBL algorithm for different choices of the parameter μ and for the two above-described networks. As can be noticed, a larger μ results in better performances in the presence of a high noise level (the superscript * denotes the lowest SSE value in each row of Tables I and II. Tables III and IV show the SSE performances and the computational loads for different choices of the number d of sliding steps. A lower SSE is provided by a smaller value of d but, in such a case, the training turns out to be more demanding in terms of computational load. On the other hand, the SSE decreases more slowly if d is taken smaller because the data batches are processed at a lower rate (see Figs. 3 and 4). Such a behavior may be ascribed to the degree of overlapping due to the parameter d: a smaller value of d involves a larger overlapping of two consecutive data batches. Thus, the data processing is slower but, for a fixed number of stages, the optimization is carried out on a larger amount of data. As a consequence, the learning is more effective, generally resulting in a smaller value of the final SSE at the end of the training. Of course, the computational effort increases. On the contrary, in general a larger value of d results in a larger final SSE, but is less demanding in terms of the computational load.

d	SSE	MFO
1	5.36	133130
2	5.52	66576
3	5.51	44333
4	5.79	33304
5	5.87	26599
6	5.79	22160
7	5.90	19003
8	5.79	16654
9	5.84	14760
10	5.98	13306
11	5.93	12096
12	5.90	11095
13	6.02	10238
14	6.01	9514
15	5.93	8873

TABLE III

FINAL SSES AND MFOS OF THE OBL ALGORITHM WITH A 2–NEURON NETWORK FOR $N = 15$, $\mu = 10$, A LOW NOISE LEVEL, AND DIFFERENT CHOICES OF d.

d	SSE	MFO
1	10.79	389824
2	10.78	196593
3	10.80	131486
4	10.82	99069
5	10.84	79278
6	10.87	66226
7	10.85	56839
8	10.88	49815
9	11.00	44252
10	11.05	39983
11	10.98	36412
12	11.12	33244
13	11.20	30765
14	11.19	28552
15	11.29	26626

TABLE IV

FINAL SSES AND MFOS OF THE OBL ALGORITHM WITH A 5–NEURON NETWORK FOR $N = 15$, $\mu = 100$, A MEDIUM NOISE LEVEL, AND DIFFERENT CHOICES OF d.

Comparison results among the various training algorithms are presented in Tables V and VI and in Figs. 5, 6, 7, and 8, for data batches corresponding to those processed by OBL with $d = 15$ and $N = 15$ (the results obtained by the OBL training for suitable choices of μ correspond to the entries with the superscript * in Tables I and II). OBL outperforms both LMBL and BPBL (note that LMBL shows a fast rate of decrease in the SSE but the final SSE value is the largest). As regards the comparison with EKFBL, OBL yields better or similar results (see Tables V and VI). However, it should be stressed that the performance of the EKFBL is obtained at the expense of a much higher computational load.

We now focus on Figs. 5, 6, 7, and 8. First of all, note that LMBL has a final SSE almost one order of magnitude larger than those of the other algorithms. Hence, let us neglect the results of LMBL and consider the final SSE for BPBL, which is the algorithm with the worst performance among EKFBL, OBL, and BPBL. The purpose is to compare the computational loads required by the three algorithms, to achieve the BPBL accuracy. From Tables V and VI with a medium noise level, it turns out that the final SSEs for BPBL are equal about to 17.71 and 11.46 for the 2–neuron and 5–neuron networks, respectively. After obtaining from Figs. 5 and 7 the iteration steps associated with these SSE values

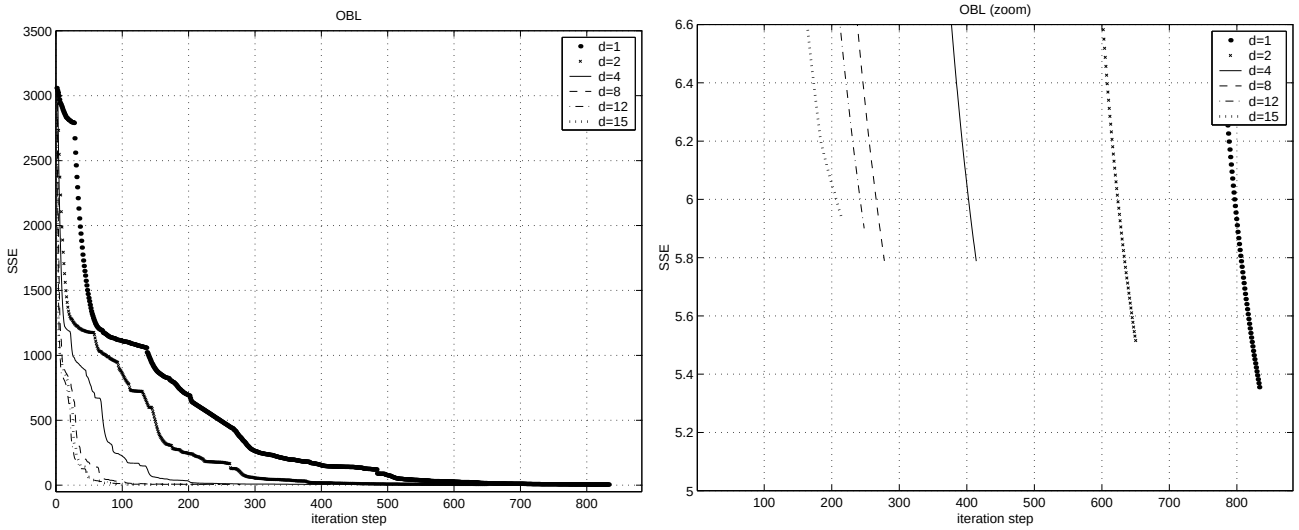


Fig. 3. SSEs for OBL with a 2–neuron network for different choices of d, $N = 15$, $\mu = 10$, and a low noise level.

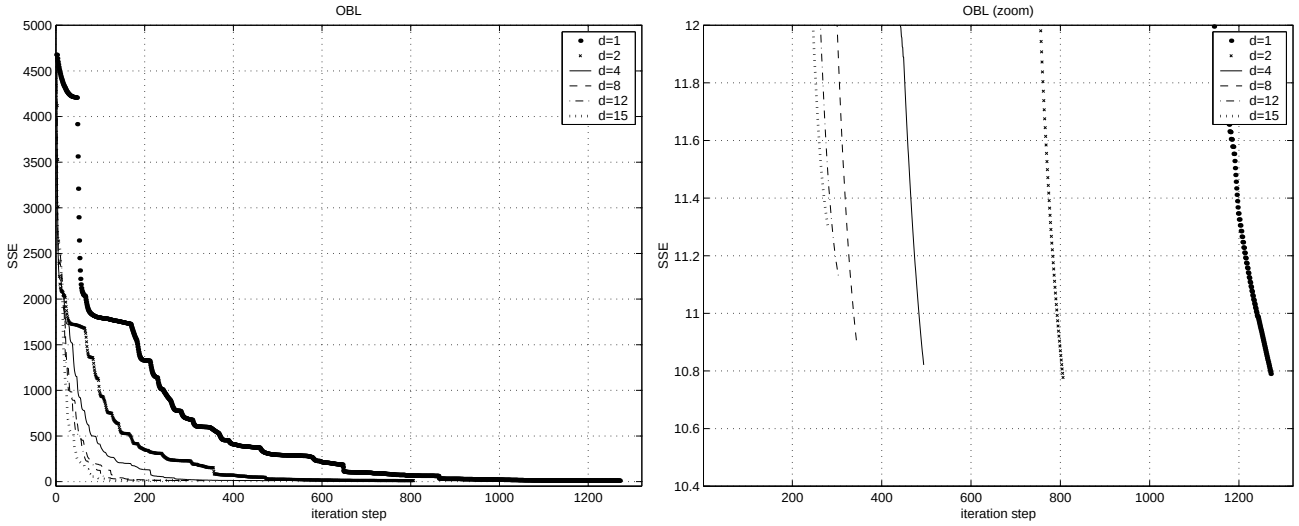


Fig. 4. SSEs for OBL with a 5–neuron network for different choices of d, $N = 15$, $\mu = 100$, and a medium noise level.

for the various algorithms, by Figs. 6 and 8 we can compare the computational loads required by BPBL, EKFBL, and OBL to achieve such errors. Algorithm OBL shows a better performance also according to this comparison criterion: its computational load is the lowest.

Finally, the prediction capabilities of the different training algorithms are presented in Figs. 9, 10, 11, and 12 for the angle prediction and in Figs. 13, 14, 15, and 16 for the angular speed prediction (with data batches corresponding to those processed by OBL with $d = 15$ and $N = 15$). Also this comparison shows that OBL outperforms both LMBL and BPBL. Its performance is comparable with that of EKFBL but is much less demanding in terms of computational load.

Acknowledgments

The authors are grateful to the anonymous Reviewers for their constructive comments.

References

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representation by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Vol. I: Foundations*, D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, Eds., pp. 318–362. MIT, Cambridge, MA, 1986.

noise level	BPBL		EKFBL		LMBL		OBL		
	SSE	**MFO**	**SSE**	**MFO**	**SSE**	**MFO**	**SSE**	**MFO**	μ
low	8.32	7438	5.48	$1.23 \cdot 10^4$	71.90	6829	4.89	8975	1
medium	17.71	7340	11.59	$1.23 \cdot 10^4$	88.65	6834	11.86	9044	5
high	51.49	7408	38.54	$1.23 \cdot 10^4$	953.70	6958	39.99	8855	50

TABLE V
FINAL SSES AND MFOS OF THE TRAINING ALGORITHMS WITH 2–NEURON NETWORKS.

noise level	BPBL		EKFBL		LMBL		OBL		
	SSE	**MFO**	**SSE**	**MFO**	**SSE**	**MFO**	**SSE**	**MFO**	μ
low	4.27	$1.61 \cdot 10^4$	2.70	$3.24 \cdot 10^4$	52.80	$1.48 \cdot 10^4$	3.94	$2.79 \cdot 10^4$	1
medium	11.46	$1.60 \cdot 10^4$	9.13	$3.24 \cdot 10^4$	60.61	$1.48 \cdot 10^4$	10.89	$2.78 \cdot 10^4$	5
high	41.60	$1.60 \cdot 10^4$	36.97	$3.24 \cdot 10^4$	395.34	$1.56 \cdot 10^4$	39.14	$2.75 \cdot 10^4$	50

TABLE VI
FINAL SSES AND MFOS OF THE TRAINING ALGORITHMS WITH 5–NEURON NETWORKS.

[2] T. Parisini, and R. Zoppoli, "Neural networks for feedback feedforward nonlinear control systems," *IEEE Trans. Neural Networks*, vol. 67, pp. 275–302, 1994.
[3] A. Alessandri, T. Parisini, and R. Zoppoli, "Neural approximators for finite-memory state estimation," *Int. J. Control*, vol. 70, pp. 275–302, 1997.
[4] T. Parisini, M. Sanguineti, and R. Zoppoli, "Nonlinear stabilization by receding-horizon neural regulators," *Int. J. Control*, vol. 70, pp. 341–362, 1998.
[5] S. R. Chu, R. Shoureshi, and M. Tenorio, "Neural networks for system identification," *IEEE Control Syst. Mag.*, vol. 10, pp. 31–35, 1990.
[6] R. Zoppoli, M. Sanguineti, and T. Parisini, "Approximating networks and extended Ritz method for the solution of functional optimization problems," *J. of Optimization Theory and Applications*, vol. 112, February 2002.
[7] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamic systems using neural networks," *IEEE Trans. Neural Networks*, vol. 1, pp. 4–27, 1990.
[8] T. J. Sejnowski and C. R. Rosenberg, "Parallel networks that learn to pronounce English text," *Complex Syst.*, vol. 1, pp. 145–168, 1987.
[9] D. J. Burr, "Experiments on neural net recognition of spoken and written text," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 30, pp. 1162–1168, 1988.
[10] A. Alessandri, M. Baglietto, T. Parisini, and R. Zoppoli, "A neural state estimator with bounded errors for nonlinear systems," *IEEE Trans. Automat. Contr.*, vol. 44, pp. 2028–2042, 1999.
[11] R. Fletcher, *Practical Methods of Optimization*, Wiley, Chichester, 1987.
[12] R. Battiti, "First- and second-order methods for learning: between steepest descent and Newton's method," *Neural Computation*, vol. 4, pp. 141–166, 1992.
[13] T. Tollenaere, "SuperSAB: fast adaptive backpropagation with good scaling properties," *Neural Networks*, vol. 3, pp. 561–573, 1990.

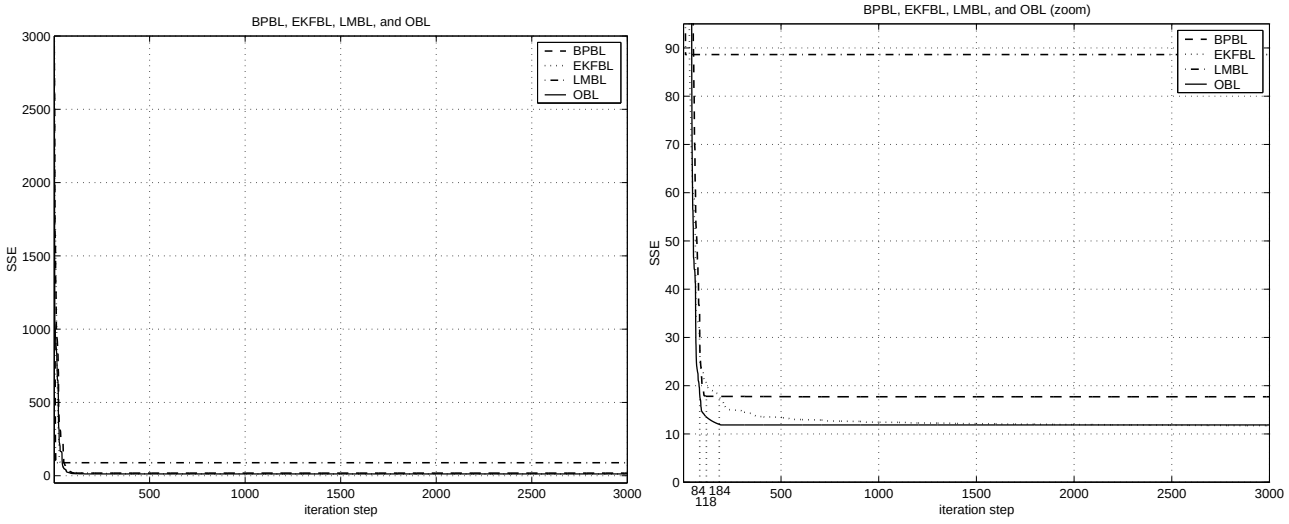


Fig. 5. SSEs for BPBL, EKFBL, LMBL, and OBL with 2–neuron networks and medium noise levels.

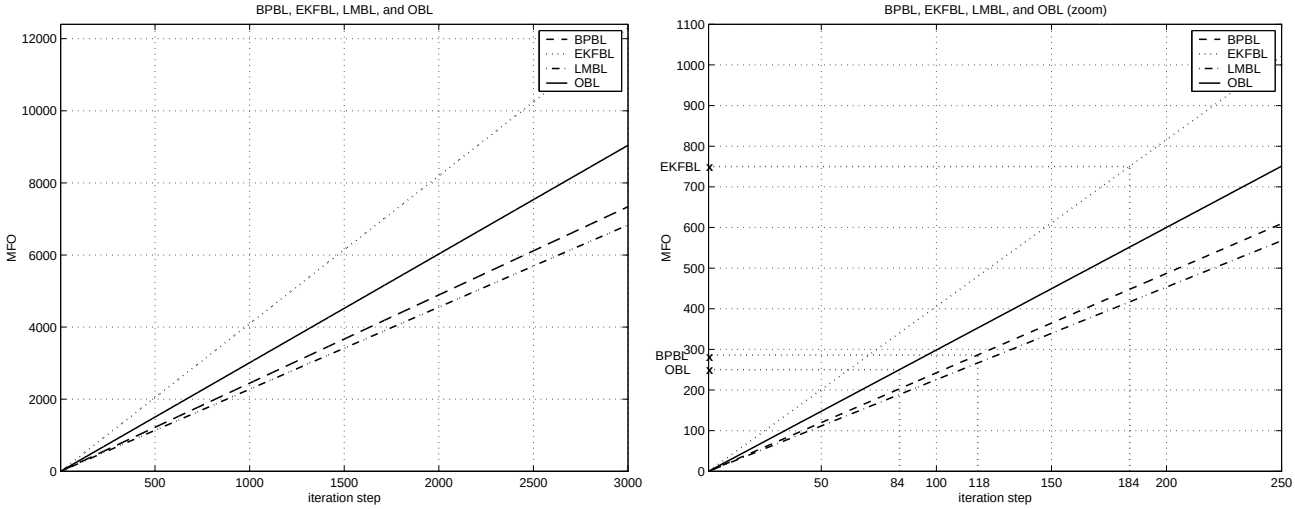


Fig. 6. MFOs for BPBL, EKFBL, LMBL, and OBL with 2–neuron networks and medium noise levels.

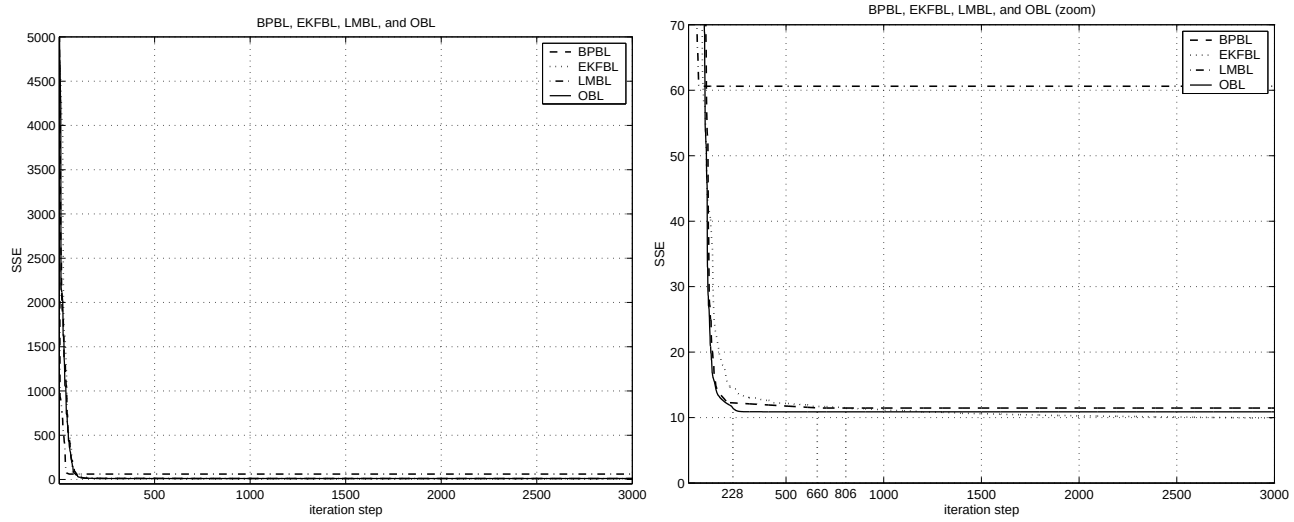


Fig. 7. SSEs for BPBL, EKFBL, LMBL, and OBL with 5–neuron networks and medium noise levels.

[14] R. A. Jacobs, "Increased rates of convergence through learning rate adaption," *Neural Networks*, vol. 1, pp. 295–307, 1988.

[15] J. W. Denton and M. S. Hung, "A comparison of nonlinear optimization methods for supervised learning in multilayer feedforward neural networks," *European Journal of Operational Research*, vol. 93, pp. 358–368, 1996.

[16] G. An, "The effects of adding noise during backpropagation training on a generalization performance," *Neural Computation*, vol. 8, pp. 643–674, 1996.

[17] S. Singhal and L. Wu, "Training multilayer perceptrons with the extended Kalman algorithm," in *Advances in Neural Information Processing Systems 1*, D. S. Touretzky, Ed., pp. 133–140. Morgan Kaufmann, San Mateo, CA, 1989.

[18] Y. Iiguni, H. Sakai, and H. Tokumaru, "A real-time learning algorithm for a multilayered neural network based on the extended Kalman filter," *IEEE Trans. Signal Processing*, vol. 40, pp. 959–966, 1992.

[19] D. W. Ruck, S. K. Rogers, M. Kabrisky, P. S. Maybeck, and M. E. Oxley, "Comparative analysis of backpropagation and the extended Kalman filter for training multilayer perceptrons," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 14, pp. 686–691, 1992.

[20] M. Stinchcombe and H. White, "Approximation and learning unknown mappings using multilayer feedforward networks with bounded weights," in *Proc. Int. Joint Conf. on Neural Networks*, 1990, pp. III7 – III16.

[21] K. Hornik, "Some new results on neural network approximation," *Neural Networks*, vol. 6, pp. 1069–1072, 1993.

[22] V. Kůrková, "Approximation of functions by perceptron networks with bounded number of hidden units," *Neural Networks*, vol. 8, pp. 745–750, 1995.

[23] V. Kůrková, "Neural networks as nonlinear approximators," in *Proc. ICSC Symposia on Neural Computation*, pp. 29–35, 2000.

[24] P. Niyogi and F. Girosi, "On the relationship between generalization error, hypothesis complexity and sample complexity for radial basis functions," *Neural Computation*, vol. 8, pp. 819–842, 1996.

[25] A. R. Barron, "Approximation and estimation bounds for artificial neural networks," *Machine Learning*, vol. 14, pp. 115–133, 1994.

[26] H. J. Sussmann, "Uniqueness of the weights for minimal feedforward nets with a given input–output map," *Neural Networks*, vol. 5, pp. 589–593, 1992.

[27] A. R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Trans. Inform. Theory*, vol. 39, pp. 930–945, 1993.

[28] V. Kůrková and M. Sanguineti, "Comparison of worst-case errors in linear and neural network approximation," *IEEE Trans. Inform. Theory*, vol. 48, 2002.

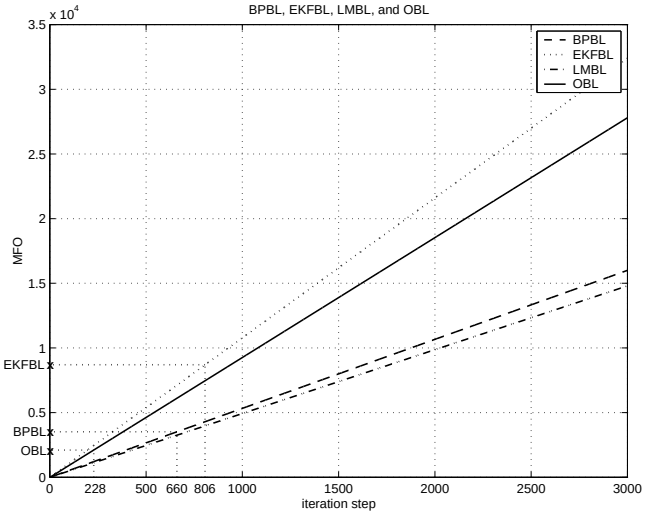


Fig. 8. MFOs for BPBL, EKFBL, LMBL, and OBL with 5–neuron networks and medium noise levels.

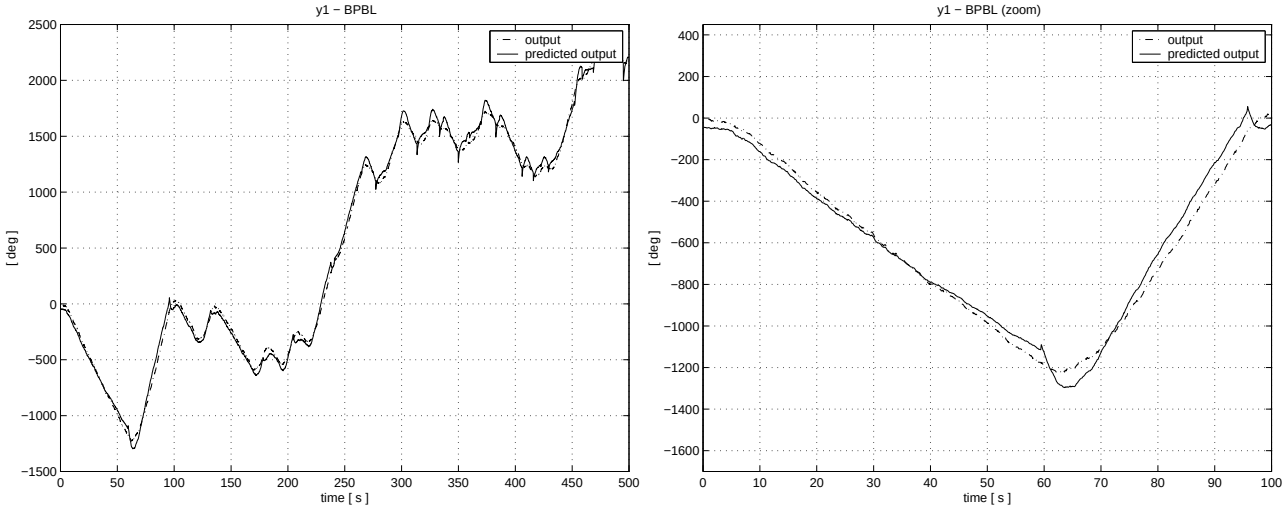


Fig. 9. y_1 prediction for BPBL (5–neuron network) with a medium noise level.

[29] V. Kůrková and M. Sanguineti, "Bounds on rates of variable-basis and neural-network approximation," *IEEE Trans. Inform. Theory*, vol. 47, pp. 2659-2665, 2001.

[30] F. Girosi and G. Anzellotti, "Rates of convergence for radial basis functions and neural networks," in *Artificial Neural Networks for Speech and Vision*, R. J. Mammone, Ed., pp. 97–113. Chapman & Hall, 1993.

[31] T. Heskes and W. Wiegerinck, "A theoretical comparison of batch-mode, on-line, cyclic, and almost-cyclic learning," *IEEE Trans. Neural Networks*, vol. 7, pp. 919–925, 1996.

[32] F. Girosi, M. Jones, and T. Poggio, "Regularization theory and neural networks architectures," *Neural Computation*, vol. 7, pp. 219–269, 1995.

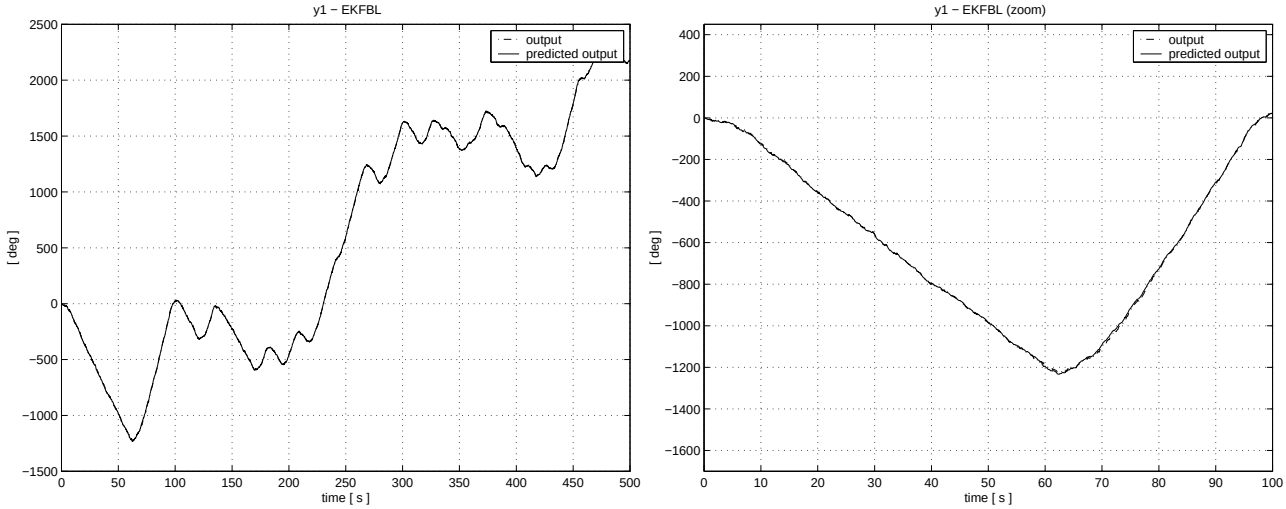


Fig. 10. y_1 prediction for EKFBL (5–neuron network) with a medium noise level.

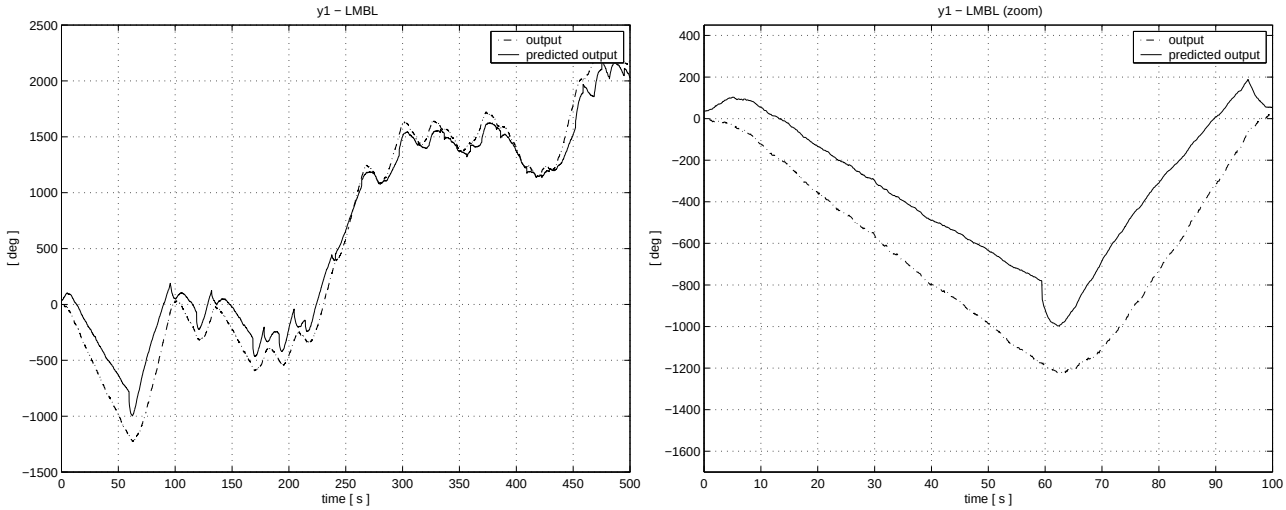
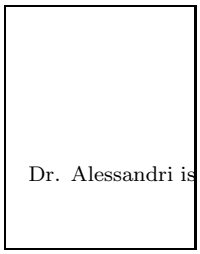


Fig. 11. y_1 prediction for LMBL (5–neuron network) with a medium noise level.



Angelo Alessandri was born in Genova, Italy, in 1967. He received the "Laurea" degree in electronic engineering in 1992 and the PhD degree in electronic engineering and computer science in 1996, both from the University of Genoa. Since 1997, he has been Adjunct Professor of System Analysis at DIST (Department of Communications, Computer and System Sciences), University of Genoa. Since 1996, he has been a Research Scientist at the Naval Automation Institute of the National Research Council of Italy, Genova (IAN-CNR). In 1998, he was a Visiting Scientist at the Naval Postgraduate School, Monterey, California. His research interests include neural networks, optimal control, estimation and fault diagnosis.

Dr. Alessandri is currently an Associate Editor for the IEEE Control Systems Society Conference Editorial Board.

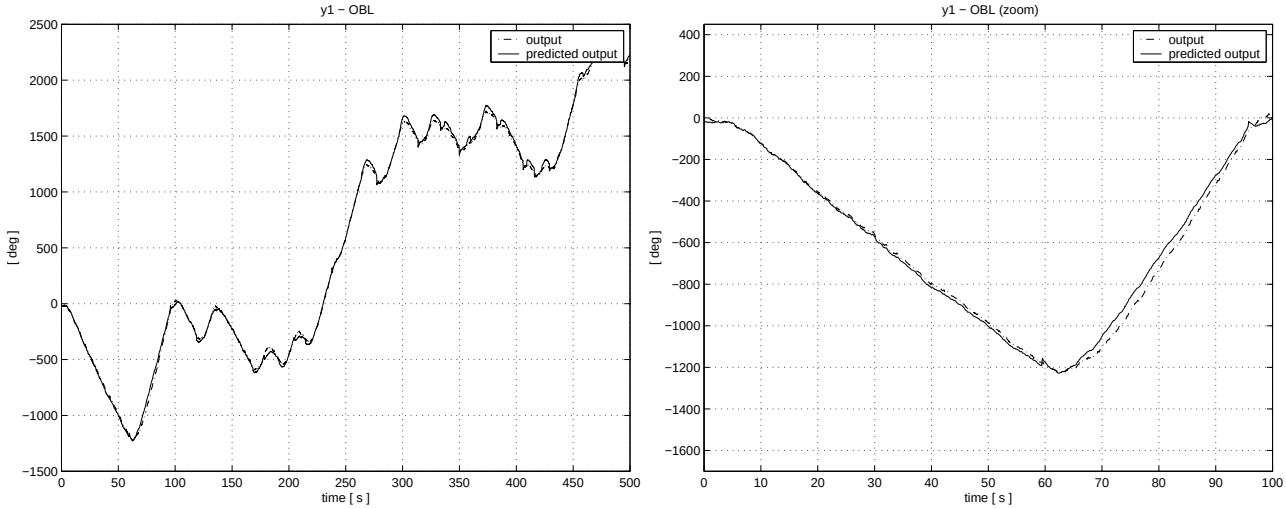


Fig. 12. y_1 prediction for OBL (5–neuron network) with a medium noise level.

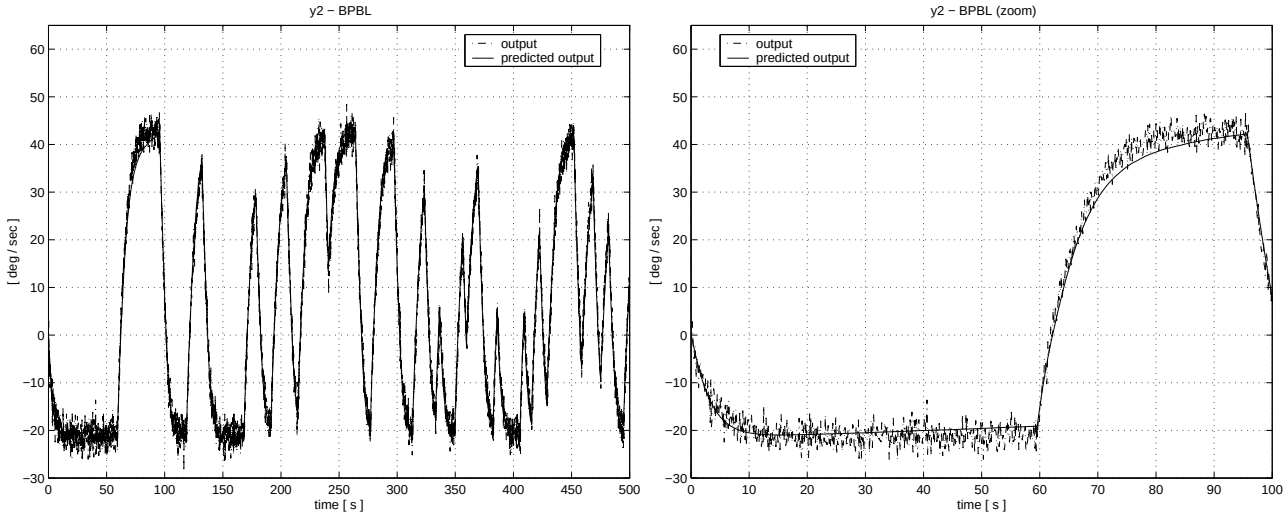


Fig. 13. y_2 prediction for BPBL (5–neuron network) with a medium noise level.

Marcello Sanguineti was born in Chiavari (Genova), Italy, in 1968. He received the "Laurea" degree in electronic engineering in 1992 and the Ph.D. degree in electronic engineering and computer science in 1996, both from the University of Genoa, Italy. Since 2000, he has been Adjunct Professor of System Analysis and of Functional Analysis for Optimization at DIST (Department of Communications, Computer and System Sciences), University of Genoa. Since 1997, he has spent several periods as Visiting Scientist at the Institute of Computer Science, Czech Academy of Sciences, Prague. He is currently a Research Associate at DIST.
His research interests include optimal control, neural networks, nonlinear approximation, and nonlinear dynamic systems.

Manfredi Maggiore received the "Laurea" degree in Electronic Engineering in 1996 from the University of Genoa, Italy, and the PhD degree in Electrical Engineering from the Ohio State University, USA, in 2000. He is currently Assistant Professor in the Edward S. Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto, Canada. His research interests include various aspects of nonlinear control, such as output feedaback control and output tracking, and nonlinear estimation.

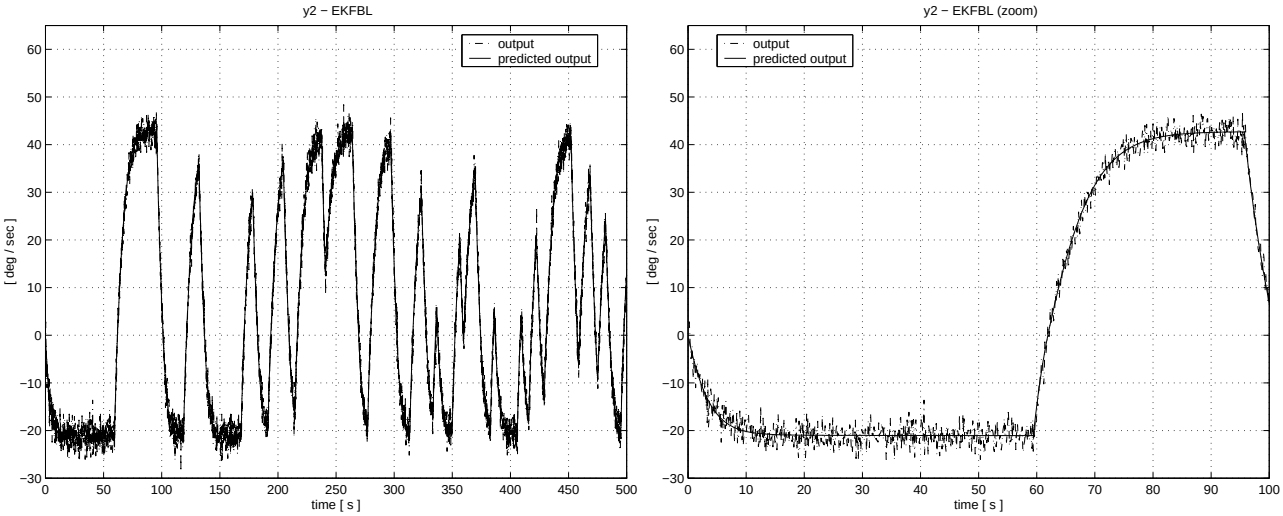


Fig. 14. y_2 prediction for EKFBL (5–neuron network) with medium noise level.

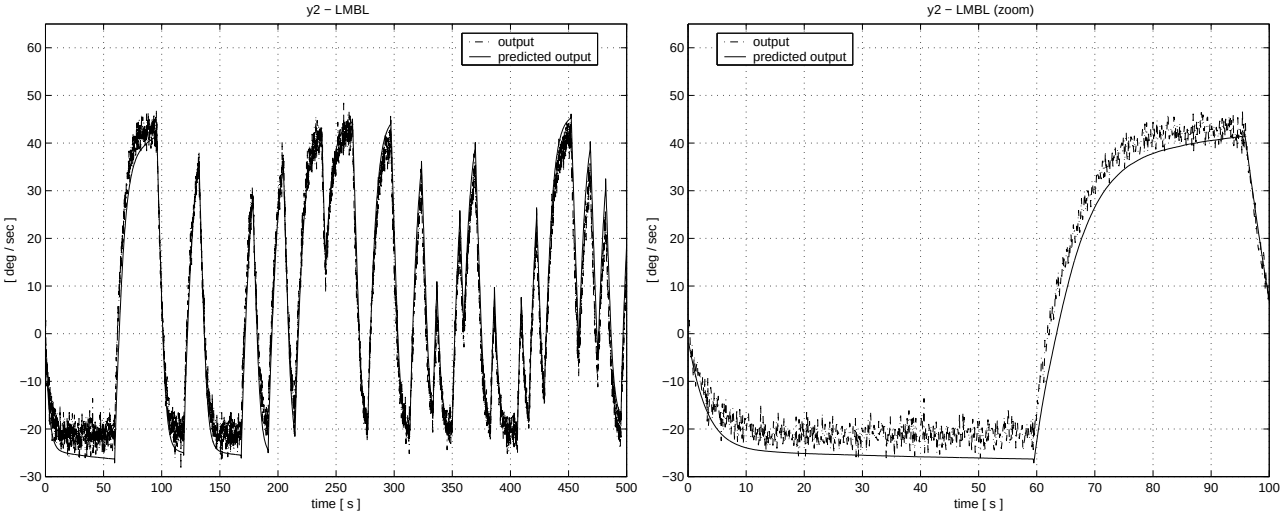


Fig. 15. y_2 prediction for LMBL (5–neuron network) with a medium noise level.

Fig. 16. y_2 prediction for OBL (5–neuron network) with a medium noise level.