EKF LEARNING FOR FEEDFORWARD NEURAL NETWORKS

A. Alessandri,[*] G. Cirimele,[**] M. Cuneo,[*] S. Pagnan,[*] M. Sanguineti[***]

[*] Institute of Intelligent Systems for Automation, ISSIA-CNR National Research Council of Italy,
Via De Marini 6, 16149 Genova, Italy
{angelo, marta, pagnan}@ge.issia.cnr.it

[**] Department of Mathematics (DIMA), University of Genoa
Via Dodecaneso 35, 16146 Genova, Italy
cir@ge.issia.cnr.it

[***] Department of Communications, Computer and System Sciences (DIST), University of Genoa
Via Opera Pia 13, 16145 Genova, Italy
marcello@dist.unige.it

Keywords: Feedforward neural networks, parameters optimization, extended Kalman filter, learning algorithms, nonlinear programming.

Abstract

Learning for feedforward neural networks can be regarded as a nonlinear parameter estimation problem with the objective of finding the optimal weights that provide the best fitting of a given training set. The extended Kalman filter is well-suited to accomplishing this task, as it is a recursive state estimation method for nonlinear systems. Such a training can be performed also in batch mode. In this paper the algorithm is coded in an efficient way and its performance is compared with a variety of widespread training methods. Simulation results show that the latter are outperformed by EKF-based parameters optimization.

1 Introduction

After the development of backpropagation (BP) [1], plenty of algorithms have been proposed to optimize the parameters in feedforward neural networks. Although BP has been successfully applied in a variety of areas, 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 [2], and to the fixed, arbitrarily chosen step length. For these reasons, algorithms using also the second derivatives have been developed (see, e.g., [3]) and modifications to BP have been proposed (see, e.g., the acceleration technique presented in [4] and the approach described in [5], 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 [6].

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) have been proposed (see, e.g., [7, 8, 9, 10, 11, 12]) that show faster convergence than BP. However, the advantages of EKF-based training are obtained at the expense of a notable computational burden (as matrix inversions are required) and a large amount of memory.

Recursive methods have been developed such that the data available at each step are used to optimize the weights. Clearly, such approaches are well-suited to dealing on line with a large amount of data on line but may suffer from poor performance. Computational efficiency can improved by using Lagrangian techniques [13]. In this context, the EKF provides a nice framework to perform optimization incrementally (i.e., one data block at a time), with advantages with respect to BP [14].

In this paper, optimization of parameters in feedforward neural networks is investigated following an EKF-based approach. The paper is organized as follows. Section 2 is focused on the approximation properties of neural networks. The basic algorithm to perform EKF learning is presented in Section 3, as well as a batch-mode EKF learning. Section 4 includes simulation results, showinh that the proposed approach outperforms backpropagation and other well-known training algorithms. The conclusions are drawn in Section 5.

2 On the approximation properties of neural networks

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

A. Alessandri and M. Sanguineti were partially supported by the MIUR 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."

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 $(\underline{u}_t,\underline{y}_t)$, $t=1,2,\ldots$, 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 a general neural network $\underline{\gamma}$, 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 $\underline{f}$ 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., [15, 16]). 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 as

$$\begin{cases} \underline{w}_{t+1} = \underline{w}_t \\ \underline{y}_t = \underline{\gamma}\,(\,\underline{w}_t,\underline{u}_t\,) + \underline{\eta}_t \end{cases} \qquad t=1,2,\ldots, \tag{3}$$

where the network weights play the role of a constant state equal to the "ideal" network weights vector $\underline{w}^*$, i.e., $\underline{w}_1 = \underline{w}_2 = \ldots = \underline{w}_P \stackrel{\triangle}{=} \underline{w}^*$, and $\underline{\eta}_t\in K\subset\mathbb{R}^p$ is a noise vector. The fictitious dynamic equations (3) allow one to regard supervised learning of feedforward neural networks 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. From now on we suppose that the network activation function in (1) belongs to the class $C^1(\mathbb{R})$.

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.

3 EKF-based parameters optimization for neural networks

A general EKF-based weights optimization algorithm can be described as follows.

EKF Algorithm. *The estimate $\underline{\hat{w}}_t$ of the network weights at time $t=1,2,\ldots$ is given by*

$$\underline{\hat{w}}_t = \underline{\hat{w}}_{t-1} + K_t\left[\,\underline{y}_t - \underline{\gamma}\,(\underline{\hat{w}}_{t-1},\underline{u}_t)\,\right] \tag{4}$$

where

$$H_t = \left.\frac{\partial\underline{\gamma}(\underline{w},\underline{u})}{\partial\underline{w}}\right|_{\underline{w}=\underline{\hat{w}}_{t-1},\underline{u}=\underline{u}_t} \tag{5}$$

$$K_t = P_t H_t^T\left(H_t P_t H_t^T + R_t\right)^{-1} \tag{6}$$

$$P_{t+1} = P_t - K_t H_t^T P_t^T + Q_t, \tag{7}$$

P_0 and R_t are symmetric positive definite matrices, Q_t is a symmetric positive semidefinite matrix, and (4) is initialized with a given $\underline{\hat{w}}_0$.

□

The extended Kalman filter is the Kalman filter of an approximate model of the nonlinear system linearized around the last estimate [17, 18, 19]. In the Kalman filter it is feasible to carry out the off-line computation of error covariance (i.e., P_t) and gain (i.e., K_t) matrices; on the contrary, for the extended Kalman filter H_t is a function of $\underline{\hat{w}}_{t-1}$ and $\underline{u}_t$, which requires the on-line computation of such matrices.

Remark. The matrix Q_t is usually taken equal either to the null matrix or to εI, with a small ε. The choice of R_t turns out to be more critical, as it represents the amount of the approximation error. Clearly, at first, it is difficult to figure it out since it depends on the structure of the networks with the initial choice of the weights, i.e., $\underline{\bar{w}}_0$. In order to overcome this issue, the matrix R_t is chosen quite large and tuned on-line (see, for example, [9], p. 962, formulas (34)–(36)).

EKF-based optimization can be used also in a batch mode. Batch training overcomes the issues arising with large data set by dividing the patterns into data batches of fixed length and by training the network with each batch at a time [20]. The proposed approach includes both standard EKF and batch-mode learning and is well-suited to dealing with large amount of data. According to recent results [21], we propose to shift the data batch of d steps, in general, with $d\le N$, as shown in Fig. 1. Full batch training may be obtained by choosing d equal to N.

Let us now consider a temporal window moving d stages at a time, where $1\le d\le N$. Given a fixed number of iterations, say t, $t=1,2,\ldots,\ N+t-1$ input/output patterns of the

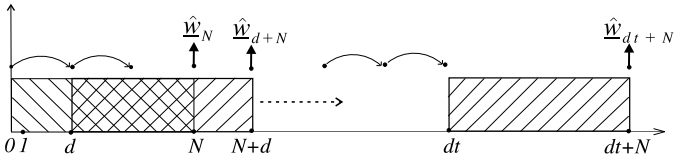

Figure 1: The d-step batch-mode training.

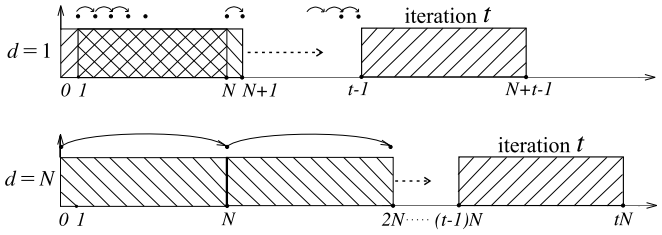

Figure 2: Comparison between the one-step and the N-step batch-mode training.

data set are explored if $d = 1$ and $N(t+1)-1$ if $d = N$. In a general case , the amount of patterns processed at step t is equal to $N + d\,t - 1$ (see Fig. 2).

Let

$$\underline{G}\left(\underline{w}, \underline{u}_{t-N+1}^{t}\right) \triangleq \begin{bmatrix} \underline{\gamma}\left(\underline{w}, \underline{u}_{t-N+1}\right) \\ \underline{\gamma}\left(\underline{w}, \underline{u}_{t-N+2}\right) \\ \vdots \\ \underline{\gamma}\left(\underline{w}, \underline{u}_{t}\right) \end{bmatrix} \in \mathbb{R}^{p\,N} \tag{8}$$

where $\underline{u}_{t-N+1}^{t} \triangleq \text{col}\left(\underline{u}_{t-N+1}, \underline{u}_{t-N+2}, \ldots, \underline{u}_{t}\right) \in U^N$ (recall that $\underline{u}_t \in U \subset \mathbb{R}^m$). Similarly, let $\underline{y}_{t-N+1}^{t} \triangleq \text{col}\left(\underline{y}_{t-N+1}, \underline{y}_{t-N+2}, \ldots, \underline{y}_{t}\right)$.

Thus, the extension of the EKF training to the general d-step case may be expressed as follows.

Batch-mode EKF (BEKF) Algorithm. *The estimate $\hat{\underline{w}}_t$, $t = N, N+1, \ldots$ of the network weights is given by*

$$\hat{\underline{w}}_t = \hat{\underline{w}}_{t-1} + \mathcal{K}_t\Big[\underline{y}_{d(t-N)}^{d(t-N)+N-1} - \underline{G}\left(\hat{\underline{w}}_{t-1}, \underline{u}_{d(t-N)}^{d(t-N)+N-1}\right)\Big] \tag{9}$$

where

$$\mathcal{H}_t = \left.\frac{\partial \underline{G}(\underline{w}, \underline{u})}{\partial \underline{w}}\right|_{\underline{w}=\hat{\underline{w}}_{t-1}, \underline{u}=\underline{u}_{d(t-N)}^{d(t-N)+N-1}} \tag{10}$$

$$\mathcal{K}_t = \mathcal{P}_t\mathcal{H}_t^T\left(\mathcal{H}_t\mathcal{P}_t\mathcal{H}_t^T + \mathcal{R}_t\right)^{-1} \tag{11}$$

$$\mathcal{P}_{t+1} = \mathcal{P}_t - \mathcal{K}_t\mathcal{H}_t^T\mathcal{P}_t^T + \mathcal{Q}_t, \tag{12}$$

$\mathcal{P}_0$ and $\mathcal{R}_t$ are symmetric positive definite matrices, $\mathcal{Q}_t$ is a symmetric positive semidefinite matrix, and (9) is initialized with a given $\hat{\underline{w}}_{N-1}$.

□

Note that algorithm BEKF exactly corresponds to the EKF training if d and N are both taken equal to 1. As, at each time t, N input/output pairs are processed instead of one, more computation is involved as the matrix to invert is of dimension $(pN)^2$ instead of p^2. This, however, makes efficient coding crucial as larger matrices are involved in the computation.

A generalization of the previous algorithm consists in repeating the weight and covariance updates by using the same batch of input/output patterns. Following [22] for the sake of comparison with alternative training methods, the repetitions are called *epochs*. Such a training method is called Iterated Batch-mode EKF (IBEKF) learning and corresponds to the procedure given below.

Iterated Batch-mode EKF (IBEKF) Algorithm. *The estimate $\hat{\underline{w}}_t$, $t = N, N+1, \ldots$ of the network weights is given by*

$\tilde{\underline{w}}_1 = \hat{\underline{w}}_{t-1}$

`for` $i = 1, 2, \ldots, N_E$

$$\mathcal{H}_j = \left.\frac{\partial \underline{G}(\underline{w}, \underline{u})}{\partial \underline{w}}\right|_{\underline{w}=\tilde{\underline{w}}_i, \underline{u}=\underline{u}_{d(t-N)}^{d(t-N)+N-1}}$$

$$\mathcal{K}_j = \mathcal{P}_j\mathcal{H}_j^T\left(\mathcal{H}_j\mathcal{P}_j\mathcal{H}_j^T + \mathcal{R}_j\right)^{-1}$$

$$\mathcal{P}_{j+1} = \mathcal{P}_j - \mathcal{K}_j\mathcal{H}_j^T\mathcal{P}_j^T + \mathcal{Q}_j$$

$$\tilde{\underline{w}}_{i+1} = \tilde{\underline{w}}_i + \mathcal{K}_j\Big[\underline{y}_{d(t-N)}^{d(t-N)+N-1} - \underline{G}\left(\tilde{\underline{w}}_i, \underline{u}_{d(t-N)}^{d(t-N)+N-1}\right)\Big]$$

$j = j + 1$

`end`

$\hat{\underline{w}}_t = \tilde{\underline{w}}_{N_E+1}$

where $\mathcal{P}_0$ and $\mathcal{R}_j$ are symmetric positive definite matrices, $\mathcal{Q}_j$ is a symmetric positive semidefinite matrix, N_E is the number of epochs, and the algorithm is initialized with a given $\hat{\underline{w}}_{N-1}$ and $j = 1$.

□

Algorithm	Mean Error	Mean Time (s)	Mean Err · Time (s)
trainb	$7.0494 \cdot 10^{-2}$	4.979	$3.51 \cdot 10^{-1}$
trainbfg	$9.7237 \cdot 10^{-3}$	6.143	$5.91 \cdot 10^{-2}$
traincgb	$1.8239 \cdot 10^{-2}$	5.138	$9.62 \cdot 10^{-2}$
traincgf	$1.3644 \cdot 10^{-2}$	5.035	$6.90 \cdot 10^{-2}$
trainekf	$2.5044 \cdot 10^{-4}$	1.384	$3.41 \cdot 10^{-4}$
traingda	$5.7891 \cdot 10^{-2}$	2.298	$1.33 \cdot 10^{-1}$
traingdx	$7.1620 \cdot 10^{-2}$	2.301	$1.64 \cdot 10^{-1}$
trainlm	$5.5580 \cdot 10^{-3}$	4.097	$2.27 \cdot 10^{-2}$
trainoss	$1.0452 \cdot 10^{-2}$	4.907	$5.17 \cdot 10^{-2}$
trainrp	$3.8550 \cdot 10^{-2}$	2.311	$8.92 \cdot 10^{-2}$
trainscg	$1.1021 \cdot 10^{-2}$	3.916	$4.31 \cdot 10^{-2}$

Table 1: Predictions with a 6–neuron one-hidden-layer neural network with $d = 10$, $N = 10$, and $N_E = 10$

Algorithm	Mean Error	Mean Time (s)	Mean Err · Time (s)
trainb	$5.6550 \cdot 10^{-2}$	4.997	$2.83 \cdot 10^{-1}$
trainbfg	$1.3295 \cdot 10^{-2}$	6.640	$8.86 \cdot 10^{-2}$
traincgb	$1.7554 \cdot 10^{-2}$	5.128	$9.01 \cdot 10^{-2}$
traincgf	$1.0989 \cdot 10^{-2}$	5.030	$5.68 \cdot 10^{-2}$
trainekf	$1.3101 \cdot 10^{-4}$	2.164	$2.91 \cdot 10^{-4}$
traingda	$4.3441 \cdot 10^{-2}$	2.330	$1.01 \cdot 10^{-1}$
traingdx	$4.9537 \cdot 10^{-2}$	2.270	$1.13 \cdot 10^{-1}$
trainlm	$8.4738 \cdot 10^{-3}$	6.755	$5.67 \cdot 10^{-2}$
trainoss	$8.9392 \cdot 10^{-3}$	4.841	$4.31 \cdot 10^{-2}$
trainrp	$3.6963 \cdot 10^{-2}$	2.350	$8.70 \cdot 10^{-2}$
trainscg	$1.2280 \cdot 10^{-2}$	3.991	$4.90 \cdot 10^{-2}$

Table 2: Predictions with a 9–neuron one-hidden-layer neural network with $d = 10$, $N = 10$, and $N_E = 10$

IBEKF reduces to BEKF algorithm if $N_E = 1$. It is worth to remark that IBEKF training method turns out to be the result of the application of the so-called Iterated Extended Kalman filter to the problem considered (see, e.g., [23]). The use of iterated Kalman filtering techniques to solve nonlinear programming problems is discussed in [14].

4 Numerical results

In this section, a problem of prediction for a chaotic time series is considered to evaluate the effectiveness of EKF learning (*trainekf*) in comparison with other widely used training algorithms (see [22]): batch training with weight and bias learning rules (*trainb*), BFGS quasi-Newton backpropagation (*trainbfg*), Powell-Beale conjugate gradient backpropagation (*traincgb*), Fletcher-Powell conjugate gradient backpropagation (*traincgf*), gradient descent with adaptive learning rate backpropagation (*traingda*), gradient descent with momentum and adaptive backpropagation (*traingdx*), Levenberg-Marquardt backpropagation (*trainlm*), one-step secant backpropagation (*trainoss*), resilient backpropagation (*trainrp*), scaled conjugate gradient backpropagation (*trainscg*). In the tests, we considered the Mackey-Glass series [24], which is a quite standard benchmark. The discrete-time Mackey-Glass series is given by the following delay-difference equation:

$$x_{t+1} = (1 - c_1)\, x_t + c_2 \, \frac{x_{t-\tau}}{1 + (x_{t-\tau})^{10}} \quad , \quad t = \tau, \tau + 1, \ldots \tag{13}$$

where $\tau \geq 1$ is an integer. The training data were generated using the parameters $c_1 = 0.1$, $c_2 = 0.2$, $\tau = 30$ and choosing the initial values of x_t uniformly distributed between 0 and 0.4. We arranged for data sets made of 100 series, each with 2000 samples. The first 1000 time steps of each series were omitted. The succeeding 500 time steps were used for training, the remaining 500 for testing. The prediction is constructed by assuming that the next value x_{t+1} depends on a vector of constant length given by the previous $l + 1$ samples, that is

$$x_{t+1} \leftarrow [x_t, x_{t-1}, x_{t-2}, \ldots, x_{t-l}], \tag{14}$$

where l was chosen equal to 5. Tables 1 and 2 summarize the simulation results. Each entry in the tables is averaged over 100 different tests, where time series with different initial conditions (uniformly distributed between 0 and 0.4) and random initial weights are used in each trial. All training algorithms used for comparison are available from the Matlab Neural Toolbox [24].

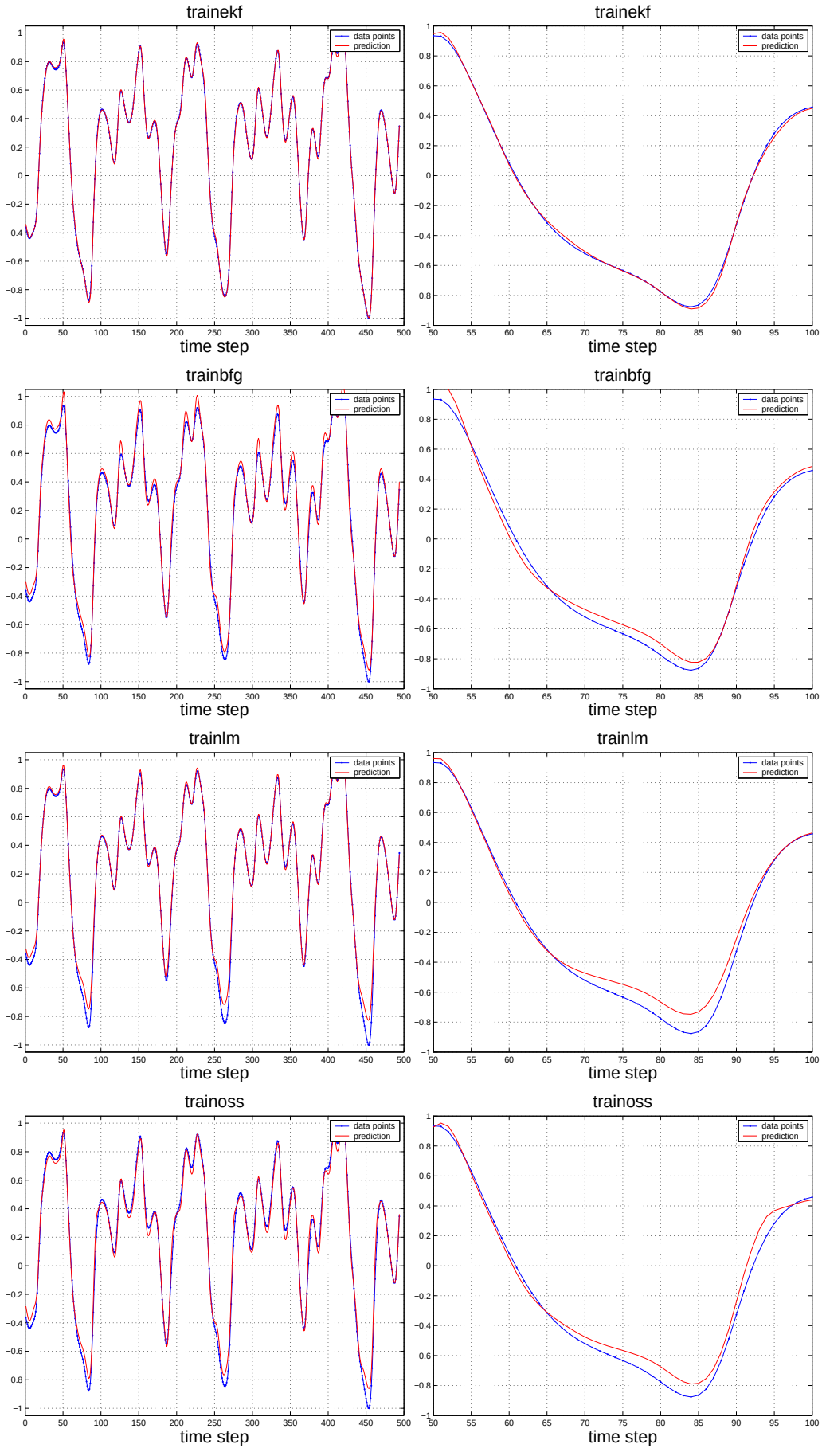

Figure 3: Mackey-Glass series predictions with a 9–neuron one-hidden-layer neural network for the four best training methods.

As can be seen in Tables 1 and 2, *trainekf* outperforms all the other algorithms in terms of both approximation precision (see the "Mean Error" column) and computational load (see the "Mean Time" column). Moreover, as a high precision may require too much computation, the efficiency may be more fairly evaluated by comparing the product of these performance indexes, which is given in the last column. The prediction capabilities obtained with the four best training methods (*trainekf*, *trainbfg*, *trainlm*, and *trainoss*) with a 9–neuron one-hidden-layer neural network are shown in Figure 3.

5 Conclusions

The problem of training neural networks via EKF has been considered and batch-mode EKF learning algorithms have been proposed to optimize the weights values. Simulation results show that the EKF training outperforms other well-known learning algorithms. However, in a fair evaluation of pros and cons, we have to point out that the EKF has to account for the storage requirements of the covariance matrix. This issue will be the objective of future investigations, as well as the convergence properties of EKF training.

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.

[2] R. Fletcher, *Practical Methods of Optimization*, Wiley, Chichester, 1987.

[3] R. Battiti, "First- and second-order methods for learning: between steepest descent and Newton's method", *Neural Computation*, **4**, pp. 141–166, 1992.

[4] T. Tollenaere, "SuperSAB: fast adaptive backpropagation with good scaling properties", *Neural Networks*, **3**, pp. 561–573, 1990.

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

[6] 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*, **93**, pp. 358–368, 1996.

[7] 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.

[8] 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. on Pattern Analysis and Machine Intelligence*, **14**, pp. 686–691, 1992.

[9] 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. on Signal Processing*, **40**, pp. 959–966, 1992.

[10] B. Schottky and D. Saad, "Statistical mechanics of EKF learning in neural networks", *Journal of Physics A: Mathematical and General*, **32**, pp. 1605–1621, 1999.

[11] K. Nishiyama and K. Suzuki, "H_∞–learning of layereed neural networks", *IEEE Trans. on Neural Networks*, **12**, pp. 1265–1277, 2001.

[12] C.-S. Leung, A.-C. Tsoi, and L. W. Chan, "Two regularizers for recursive least squared algorithms in feedforward multilayered neural networks", *IEEE Trans. on Neural Networks*, **12**, pp. 1314–1332, 2001.

[13] L. Grippo, "Convergent on-line algorithms for supervised learning in neural networks", *IEEE Trans. on Neural Networks*, **11**, pp. 1284–1299, 2002.

[14] D. P. Bertsekas, "Incremental least–squares methods and the extended Kalman filter", *SIAM Journal on Optimization*, **6**, pp. 807–822, 1996.

[15] M. Stinchcombe and H. White, "Approximation and learning unknown mappings using multilayer feedforward networks with bounded weights", *Proc. Int. Joint Conf. on Neural Networks IJCNN'90*, 1990, pp. III7 – III16.

[16] V. Kůrková, "Neural networks as nonlinear approximators", *Proc. ICSC Symposia on Neural Computation (NC'2000)* (H. Bothe and R. Rojas, Eds.), 2000, pp. 29–35.

[17] A. H. Jazwinski, *Stochastic Processes and Filtering Theory*, Academic Press, New York, 1970.

[18] A. Gelb, *Applied Optimal Estimation*, M.I.T. Press, Reading, MA, 1974.

[19] B. D. O. Anderson and J. B. Moore, *Optimal Filtering*, Prentice Hall, New York, 1979.

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

[21] A. Alessandri, M. Sanguineti, and M. Maggiore, "Optimization-based learning with bounded error for feedforward neural networks", *IEEE Trans. on Neural Networks*, **13**, pp. 261–273, 2002.

[22] H. Demuth and M. Beale, *Neural Network Toolbox – User's Guide*, The Math Works, Inc., Natick, MA, 2000.

[23] B. M. Bell and F. W. Cathey, "The iterated Kalman filter update as a Gauss-Newton method", *IEEE Trans. on Automatic Control*, **38**, pp. 294–297, 1993.

[24] M. C. Mackey and L. Glass, "Oscillation and chaos in physiological control systems", *Science*, **197**, pp. 287–289, 1977.