RBF Networks Training Using a Dual Extended Kalman Filter

Iulian B. Ciocoiu

Technical University of Iaşi, Romania
P.O. Box 877, Iaşi, 6600, Romania
Phone/Fax: + 40-32-213737; e-mail: iciocoiu@etc.tuiasi.ro

Abstract: **A new supervised learning procedure for training RBF networks is proposed. It uses a pair of parallel running Kalman filters to sequentially update both the output weights and the centers of the network. The method offers advantages over the joint parameters vector approach in terms of memory requirements and training time. Simulation results for chaotic time series prediction and the 2-spirals classification problem are reported, and the effect of using 2 different pruning techniques for improving the generalization capacity is addressed.**

Index Terms – **RBF networks, supervised training, Kalman filter, pruning**

I. INTRODUCTION

Radial Basis Function (RBF) networks have been traditionally used as a multidimensional interpolation technique of general mappings $f : R^N \rightarrow R$ according to [1]:

$$f(\boldsymbol{X}) = w_0 + \sum_{i=1}^{M} w_i \Phi(\left\| \boldsymbol{X} - \boldsymbol{C}^i \right\|) \tag{1}$$

where ϕ is a nonlinear function selected from a set of typical ones, $\| . \|$ denotes the Euclidean norm, w_i are the tap weights and $C^i \in R^N$ are called RBF centers. It is easy to see that the formula above is equivalent to a special form of a 2-layer perceptron, which is *linear in the parameters* by fixing all the centers and nonlinearities in the hidden layer. The output layer simply performs a linear combination of the (nonlinearly) transformed inputs and thus the tap weights w_i can be obtained by using the standard LMS algorithm or its momentum version. This leads to a dramatic reduction of the computing time with the supplementary benefit of avoiding the problem of local minima, usually encountered when simulating standard multilayer perceptrons. Anyway, in many applications the LMS algorithm could still be too slow, especially when the input correlation matrix is ill-conditioned [2]. Superior alternatives have been proposed, for example those relying on orthogonalization procedures [3], [4].
The approximation capabilities of RBF networks critically depend on the choice of the centres. Most existing approaches use a hybrid training strategy, using an unsupervised algorithm (*e.g.*, k-means clustering or Kohonen's self-organizing maps [2]) to pick the centres, followed by a supervised one to obtain the output weights. Theoretical justification of the suitability of such a strategy is presented in [5]. Anyway, in order to acquire optimal performance the centres training procedure should also include the target data [6], leading to a form of supervised learning which proved superior in several applications [7].
In this paper we propose the use of the Kalman filter as a framework for the supervised training of both weights and centres of the network, following the line of previous work related to considering the training procedure of a neural network as an estimation problem [8], [9]. More specifically, due to the intrinsic nonlinearity of the models, on-line linearization around the current state vector is required, leading to the Extended Kalman Filter (EKF) algorithm [2]. In the context of RBF networks the Kalman filter was used for output weights estimation only [10], and combined weights and centres estimation, by concatenating them into a joint state vector (this approach will be subsequently called

Global EKF (GEKF)) [11]. We propose a novel solution, based on using a *pair* of parallel running Kalman filters to sequentially update both the weights and the centres of the network. We call it a Dual Extended Kalman Filter (DEKF[1]) algorithm, by analogy with a similar concept introduced in [12], where a pair of Kalman filters was used for combined estimation of the states and the weights of a standard multilayer perceptron performing time series analysis. The proposed approach has the following advantages, to be further explicited in the next paragraph:

- much similar to GEKF, centers positions need not be computed before the estimation of the output weights and, more important, on-line training is possible. Anyway, the memory requirements and computational complexity are much smaller than in the joint parameters vector case.
- in the case of output weights estimation only, the required linearization of the state equations introduces no errors, since the model is *linear* in those parameters. This advantage would be lost if we used a joint (centres and weights) parameters vector, since the state-equations would still be nonlinear. Moreover, when compared to the classical gradient-descent procedure this solution offers improved performance, since it is a second-order parameter estimation method.
- the results provided by the Kalman filter training procedure for estimating the centers values may be directly used to obtain a measure of their relative importance for solving a given approximation task. This saliency information represents the basis for a pruning strategy aiming at improving the generalization capability of the network by eliminating the unimportant centers and associated output weights.

II. ALGORITHM DESCRIPTION

The Kalman filter is a well-known estimation procedure of a vector of parameters from available measured data. It is based on the formulation of the application within a state-space framework and was originally introduced for linear models, but linearization can be used to extend the method for the nonlinear case too. Given a controllable and observable system we may write [13]:

$$\begin{aligned} \boldsymbol{X}[k+1] &= \boldsymbol{\Psi}[k+1,k]\boldsymbol{X}[k] + \boldsymbol{v}[k] \\ \boldsymbol{y}[k] &= \boldsymbol{C}[k]\boldsymbol{X}[k] + \boldsymbol{q}[k] \end{aligned} \tag{2}$$

where $\boldsymbol{\Psi}$[k+1, k] is the state transition matrix (supposed to be known), **v**[k] is the input driving noise, and **q**[k] is the measurement noise, which are specified by the following correlation matrices (δ_{kn} denotes the Kronecker delta function):

$$\begin{aligned} E\{\boldsymbol{v}[k]\boldsymbol{v}^T[n]\} &= \delta_{kn}\boldsymbol{Q}[n] \\ E\{\boldsymbol{q}[k]\boldsymbol{q}^T[n]\} &= \delta_{kn}\sigma^2{}_q[n] = \delta_{kn}\boldsymbol{R}[n] \\ E\{\boldsymbol{v}[k]\boldsymbol{q}^T[n]\} &= \boldsymbol{0} \end{aligned} \tag{3}$$

Let us define by $\hat{\mathbf{X}}^-[k]$ the *predicted* value of the state vector at time k based on all information available *before* time instant k, and by $\mathbf{P}^-[k]$ its associated covariance error matrix. We may update this value using the currently measured data **y**[k] using a linear combination of the old value and the prediction error at time k according to:

$$\hat{\mathbf{X}}[k] = \hat{\mathbf{X}}^-[k] + \mathbf{G}[k](y[k] - \mathbf{C}[k]\hat{\mathbf{X}}^-[k]) \tag{4}$$

where **G**[k] represents the current value of the Kalman gain, whose value is obtained by:

[1] The DEKF notation should not be confused with the one in [9], which stands for Decoupled Extended Kalman Filter.

$$\mathbf{G}[k] = \mathbf{P}[k]^{-}\mathbf{C}[k]^{T}(\mathbf{C}[k]\mathbf{P}[k]^{-}\mathbf{C}[k]^{T} + R[k])^{-1} \tag{5}$$

The predicted error covariance matrix $\mathbf{P}^{-}[k]$ is recursively obtained via a Ricatti-type equation of the form:

$$\mathbf{P}^{-}[k+1] = \mathbf{\Psi}[k+1,k]\mathbf{P}[k]\mathbf{\Psi}^{T}[k+1,k] + \mathbf{Q}[k] \tag{6}$$

where the current matrix **P**[k] is given by:

$$\mathbf{P}[k] = \mathbf{P}^{-}[k] - \mathbf{G}[k]\mathbf{C}[k]\mathbf{P}^{-}[k] \tag{7}$$

When nonlinear models are used the linearization of the equations around the current operating point is needed, thus approximating a nonlinear system by a time-varying linear one. The matrices $\mathbf{\Psi}$[k+1,k] and **C**[k] must be replaced accordingly with the Jacobian of the (nonlinear) function appearing in the state transition equation, and the measurement equation, respectively, leading to the formulation of the so-called Extended Kalman filter algorithm [13].

A short discussion about the memory requirements for storing the (symmetric) error covariance matrix **P** and the computational complexity of the algorithm is now in place. Consider an RBF network with a single output neuron using a gaussian-type activation function for the neurons in the hidden layer:

$$\Phi(\mathbf{X}) = e^{-\frac{\mathbf{X}^2}{2\sigma^2}} \tag{8}$$

Suppose there are M centers of dimensionality N, and associated (M+1) output weights, along with a common σ value. When concatenating all those parameters into a single state vector, this amounts to a total length of T = [M*N+M+2] components. The covariance matrix **P** has T^2 elements, which is prohibitively large for most practical applications. Moreover, the computational requirements are also of the order $O(T^2)$ [14]. Several methods have been proposed in order to reduce these requirements, among which we may cite:

- partition of the global estimation problem into a set of separate, local subproblems, whose degree of granularity vary from the layer level towards a single neuron level (MEKA) [15];
- split the global state vector into groups of components that are considered independent (DEKF) [16]. As a consequence, the original error covariance matrix may be block-diagonalized by ignoring the off-diagonal terms.

The proposed approach is related to the second choice. We use a pair of distinct Kalman filters for estimating the optimum values of the centers, respectively the output weights of an RBF network. The state vector of the first filter has T_C = (M*N) components, while the second one has T_W = (M+1) components. The error covariance matrices will have a total number of $(T^2_C+T^2_W)$ elements, which is much less than for the global case when the number of centers is large.

Standard use of the Kalman filtering framework for designing feedforward neural networks is based on the formulation of the state transition equation simply as [14], [15]:

$$\mathbf{X}[k+1] = \mathbf{X}[k] \tag{9}$$

where the state vector **X** includes all parameters of the network. Comparing the relation above with equation (2) it is important to note that the absence of a process noise term may yield computational difficulties in estimating the error covariance matrix **P** (it may become singular or lose the property of nonnegative definiteness). In order to avoid this we may add *artificial* process noise, which has additionally proven to increase the rate of convergence and the quality of solutions in a number of applications [14]. As will be pointed out in the next paragraph, this idea is also useful for assessing the relative importance (saliency) of the network parameters, which may be pruned accordingly [17].

In the following, we present the actual equations used for estimating the values of the centres and the output weights of the RBF network.

A. Estimation of the output weights

The learning task is formulated as follows:

$$\begin{aligned} \boldsymbol{W}[k] &= \boldsymbol{W}[k-1] \\ y[k] &= \Phi(\boldsymbol{C}[k-1])\boldsymbol{W}[k] + q[k] \end{aligned} \tag{10}$$

where: $\boldsymbol{W}[k] = \begin{bmatrix} w_0 & w_1 \cdots w_M \end{bmatrix}^T$, $\Phi(k) = \begin{bmatrix} 1\Phi(\left\| \boldsymbol{X}[k] - \boldsymbol{C}^1 \right\|)\Phi(\left\| \boldsymbol{X}[k] - \boldsymbol{C}^2 \right\|)\ldots\Phi(\left\| \boldsymbol{X}[k] - \boldsymbol{C}^M \right\|) \end{bmatrix}$,

and q[k] is the measurement noise, assumed white with variance σ^2_q. It is important to observe that in the case of RBF networks with *fixed* centres the estimation (learning) problem is a *linear* one, as opposed to the case of standard MLP networks. According to equations (4-7), the (least square) estimate of the output weights vector $\hat{\boldsymbol{W}}[k]$ and its prediction $\hat{\boldsymbol{W}}^-[k+1]$, along with their respective error covariance matrices $P_{\boldsymbol{W}}[k]$, and $P_{\boldsymbol{W}}^-[k+1]$ become:

$$\boldsymbol{G}_W[k] = \boldsymbol{P}_{\boldsymbol{W}}^-[k]\Phi^T[k]^* \{\Phi[k]\boldsymbol{P}_{\boldsymbol{W}}^-[k]\Phi^T[k] + \sigma^2{}_q\}^{-1} \tag{11}$$

$$\hat{\boldsymbol{W}}[k] = \hat{\boldsymbol{W}}^-[k] + \mathbf{G}_W[k]\{y[k] - \Phi[k]\hat{\boldsymbol{W}}^-[k]\} \tag{12}$$

$$\boldsymbol{P}_{\boldsymbol{W}}[k] = \boldsymbol{P}_{\boldsymbol{W}}^-[k] - \mathbf{G}_W[k]\Phi[k]\boldsymbol{P}_{\boldsymbol{W}}^-[k] \tag{13}$$

$$\hat{\boldsymbol{W}}^-[k+1] = \hat{\boldsymbol{W}}[k] \tag{14}$$

$$\boldsymbol{P}_{\boldsymbol{W}}^-[k+1] = \boldsymbol{P}_{\boldsymbol{W}}[k] \tag{15}$$

where $\mathbf{G}_W$[k] designates the current value of the *Kalman gain*.

B. Estimation of the centres

A second Kalman filter is used to estimate the centres, which are described by the state equations:

$$\begin{aligned} \boldsymbol{C}[k] &= \boldsymbol{C}[k-1] \\ y[k] &= f\{\boldsymbol{X}[k], \Phi(\boldsymbol{C}[k]), \boldsymbol{W}[k-1]\} + q[k] \end{aligned} \tag{16}$$

where f(.) is given in equation (1). The adaptation algorithm requires the linearization of the relation above and is formulated as [13]:

$$\boldsymbol{G}_C[k] = \boldsymbol{P}_{\boldsymbol{C}}^-[k]J^T[k]\square\ \{\boldsymbol{J}[k]\boldsymbol{P}_{\boldsymbol{C}}^-[k]\boldsymbol{J}^T[k] + \sigma^2{}_q\}^{-1} \tag{17}$$

$$\hat{\boldsymbol{C}}[k] = \hat{\boldsymbol{C}}^-[k] + \boldsymbol{G}_C[k]\{y[k] - \Phi(\hat{\boldsymbol{C}}^-[k])\hat{\boldsymbol{W}}^-[k]\} \tag{18}$$

$$\boldsymbol{P}_{\boldsymbol{C}}[k] = \boldsymbol{P}_{\boldsymbol{C}}^-[k] - \mathbf{G}_C[k]\boldsymbol{J}[k]\boldsymbol{P}_{\boldsymbol{C}}^-[k] \tag{19}$$

$$\boldsymbol{J}[k] = \frac{\partial f[\hat{\boldsymbol{C}}[k], \hat{\boldsymbol{W}}]}{\partial \hat{\boldsymbol{C}}[k]} \tag{20}$$

$$\hat{\boldsymbol{C}}^{-}[k+1] = \hat{\boldsymbol{C}}[k] \tag{21}$$

$$\boldsymbol{P_C}^{-}[k+1] = \boldsymbol{P_C}[k] \tag{22}$$

where the elements of the Jacobian are:

$$\mathrm{J}_{ij}[k] = \{y[k] - d[k]\} w_{kj} e^{-\frac{\left\|\boldsymbol{X}[k] - \boldsymbol{C}^{j}\right\|^{2}}{2\sigma_j^{2}}} \frac{X_i[k] - \boldsymbol{C}_i{}^{j}}{\sigma^2{}_j} \tag{23}$$

C_i^j denotes component i of center vector $\mathbf{C}^j$, d[k] is the current desired output, and y[k] is the output of the RBF network.

The learning algorithm works on a pattern-by-pattern basis: the adaptation procedure consists in consecutively performing the modification of the centres (eq.(17)-(22)) and the weights (eq. (11)-(15)) on the arrival of each training input-output pair. Typically, the initial values of the centres should be obtained after an unsupervised training phase, while the weights are initialized to small random values. The initial values of the symmetric error covariance matrices reflect the uncertainty in locating the weights and the centres, and are typically chosen proportional to an identity matrix of proper order [13]. As pointed out in [14], those matrices and the noise covariance matrices **Q** and **R** represent the only parameters one has for tuning the Kalman filter, so their proper setting is critical in obtaining accurate results, especially for nonlinear models. Making noise covariances larger decreases the Kalman gain, much similar as decreasing the value of the learning rate in standard gradient descent. A method for estimating noise covariances values from measured data is given in [18] based on a general recursive procedure presented in [19]. Specifically, matrix **R** is assumed to be diagonal **R** = λ**I**, where the parameter λ is estimated recursively by:

$$\hat{\lambda}[k] = \hat{\lambda}[k-1] + \mu[k]\left\{(d[k] - y[k])^2 - \hat{\lambda}[k-1]\right\} \tag{24}$$

where $\mu[k] = \frac{1}{k}$. A specific approach related to speech processing applications is reported in [12], considering periodic re-estimation based on a linear autoregressive (AR) model describing the measured data.

III. SIMULATION RESULTS

We have tested the efficiency of our algorithm on two different applications, namely one-step ahead prediction of the chaotic time series generated by the quadratic map, and the celebrated 2-spirals classification problem.

A. Chaotic time series prediction

The database was formed of 500 points generated by the quadratic map: y[k] = ax[k](1-x[k]), which is known to exhibit chaotic behavior for a = 4. We retained a separate test set of 100 points to assess the quality of the solution. The measurement noise was considered as Gaussian zero mean with $\sigma^2_q = 10^{-6}$. Initial values for the weights were selected random. Gaussian activation functions were used, and the centres were initially chosen using the k-means clustering algorithm. The variances σ^2_j were set by the k-nearest-neighbour algorithm. Computer simulations were performed using different number of centers and the following algorithms: a) global EKF (GEKF) as proposed in [x]; b) Dual EKF (DEKF) based on a pair of Kalman filters as described above; c) Kalman filter training of the output weights only (KF_weights); d) standard LMS for training all parameters of the RBF network.

In Fig. 1 the phase plots of the true and predicted series on the test set are shown in the case of DEKF algorithm using 5 centres. Typical learning trajectories are given in Fig. 2, showing comparable convergence speed for the DEKF and GEKF algorithms, while all Kalman-based approaches outperformed the standard LMS algorithm (typically less than 20 training epochs compared to several hundreds of epochs for the LMS case). It is clear from these figures that the proposed solution yields high performance for this particular application even for a small number of centres, which is further proven by the values of the normalized mean-square error (NMSE) and cross-correlation coefficient R given in Table 1 (averages for 10 separate runs). R is calculated according to:

$$R = \sqrt{1 - \frac{\sum_i \{y[i] - y_p[i]\}^2}{\sum_i \{y[i] - \overline{y}\}^2}} \tag{24}$$

B. Two-spirals classification problem

The task is to correctly classify two sets of 194 training points which lie on two distinct spirals in the x-y plane. The spirals twist three times around the origin and around each other. This is a benchmark problem considered to be extremely difficult for standard multilayer networks trained with classical back-propagation class algorithms, although successful results were reported using other architectures or learning strategies [20], [21].

We performed intensive computer simulations using gaussian activation functions, variable number of centres and initialization procedures. We tested the same algorithms indicated in the previous section, namely GEKF, DEKF, KF_weights, and the gradient-descent procedure described in [6]. Simulation results in terms of NMSE values and classification accuracy are indicated in Table 2. They show much similar performances of DEKF and GEKF while the former needs significantly less computation time (tests were performed with MATLAB 5.3 running on a Pentium II/400 MHz processor). The classification performance was tested as follows: a) on a separate set of 41x41 points, uniformly distributed on the surface covered by the training data. In Fig. 3 we present the results for M = 96 centres, evenly selected initially from the training database; b) on a set of 194 points placed in between the training data (not shown in the figures). The results are given in Table 2.

Convergence was typically reached in about 150 training epochs with DEKF, while the gradient-descent required several thousands of epochs and a careful tuning of the learning parameters. Rigorously speaking, the σ_j parameters of the gaussian functions should also be estimated during the training phase, but considering an extra Kalman filter could render the computational cost excessive. Since the approximation capabilities of RBF networks are still preserved using a common value for those parameters [22], they were all taken equal to 0.3. We have also tested the possibility of heuristically adapting their values according to the distance between the centres, but no improvement was obtained.

In order to improve the generalization performances of neural networks we could typically choose between two alternatives [2]: a) including regularization terms in the definition of the error function; b) constructive methodologies leading to proper architecture synthesis. The approaches belonging to the later case fall into two categories, namely "learn-and-grow" techniques, starting from small nets and successively introducing neurons and weights, respectively pruning methods, starting with large networks and progressively eliminating insignificant weights/neurons. In the context of RBF networks both approaches have been tested, *e.g.* Platt's resource allocating network [23], and $\Im$-Projection principle [10]. We have compared the performances of 2 distinct pruning strategies, which are described next:

a) *Principal Components Pruning* (PCP) [24] is ideally suited for linear networks (such as RBF networks with fixed centres). Basically, the idea relies on projecting the (successive layers of) weights of a trained network on the subspace spanned by the significant eigenvectors of the corresponding input correlation matrix. It is easy to apply even for large networks and it needs no retraining after pruning is performed. The relation between PCP and other pruning techniques as Optimal Brain Damage (OBD) and Optimal Brain Surgeon (OBS) is presented in [25]. Eliminating eigennodes should improve generalization performances by reducing the number of *effective* number of parameters of the network [25].
b) recently, a number of papers have addressed pruning within a Kalman filtering framework [17], [26]. The relative importance (saliency) of the parameters of a feedforward neural network is assessed using specific information available after the training period, namely the final values of the error covariance matrix **P** from equation (7). Denoting $P_\infty = \lim_{k \to \infty} P[k]$ and the covariance matrix of the process noise **Q** = q**I,** the incremental change of the approximation error due to removing the k-th element of the (parameters) state-vector **W** is given by [17]:

$$\Delta E_k = q\left(\mathbf{P}_\infty{}^{-2}\right)_{kk} \mathbf{W}_k{}^2 \tag{25}$$

Remark: The formula above is valid under the assumption that the eigenvalues of P_∞ are much larger than q (which was true in our experiments). Moreover, while the standard approach when using the Kalman filter in neural network training assumes no process noise to be present, experimental results show that its inclusion has benefits in terms of numerical stability and optimality of the final solution [14].

We applied both techniques for the classification problem described above on a network previously trained with the DEKF algorithm, and the results are presented in Fig. 4. It is obvious that only 75 centres out of 96 could be used without significantly degrading the classification performances. In Fig. 5 we present the modification of the NMSE values and classification accuracy on the training set as a function of remaining centers, showing smoother degradation for the Kalman-based pruning strategy.

IV. CONCLUSIONS

We analyzed the efficiency of a new fully supervised training algorithm for RBF networks relying on a pair of parallel running Kalman filters. It offers advantages when compared to the global approach in terms of memory requirements and computational cost whereas the approximation capability is comparable. Decoupling the training procedure of the weights and the centres of the RBF network could sometimes result in convergence problems, although in the simulations reported herein this phenomenon was not observed. In the case of large networks, the memory requirements for storing the (symmetric) error covariance matrices could still become prohibitive, and pruning techniques need to be used. A recently introduced Kalman-based pruning strategy was proven efficient for a difficult classification problem.

REFERENCES:

[1] D.S. Broomhead, D. Lowe, Multivariable functional interpolation and adaptive networks, Complex Systems 2 (1988) 321-355.
[2] S. Haykin, Neural Networks - A Comprehensive Foundation, IEEE Press, New York, 1994.
[3] S. Chen, S. , C.F.N. Cowan, P.M. Grant, Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks", IEEE Trans. Neural Networks 2 (2) (1991) 302-309.
[4] W. Kaminski, P. Strumillo, Kernel Orthonormalization in Radial Basis Function Neural Networks, IEEE Trans. Neural Networks 8 (5) (1997) 1177-1183.
[5] T. Chen, R. Chen, Approximation Capability to Functions of Several Variables, Nonlinear Functionals and Operators by Radial Basis Function Neural Networks, IEEE Trans. Neural Networks (1995)
[6] C. M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, New York, 1995.
[7] D. Wettschereck, T. Dietterich, Improving the performance of radial basis function networks by learning centre locations, in: J.E. Moody, S.J. Hanson, R.P. Lippmann (Eds.), Advances in Neural Information Processing Systems 4, Morgan Kaufmann, San Mateo, CA, 1992, pp. 1133-1140.
[8] J. Connor, R. Martin, L. Atlas, Recurrent neural networks and robust time series prediction, IEEE Trans. Neural Networks 5 (2) (1994) 240-254.
[9] G. Puskorious, L. Feldkamp, Neural Control of Nonlinear Dynamic Systems with Kalman Filter Trained Recurrent Networks, IEEE Trans. Neural Networks 5 (2) (1994) 279-297.
[10] V. Kadirkamanathan, M. Niranjan, F. Fallside, Models of dynamic complexity for time-series prediction, ICASSP, 1992, pp. 1-4.
[11] I.T. Nabney, Practical methods of tracking of non-stationary time series applied to real world problems, in: S.K. Rogers, and D.W. Ruck (Eds.), AeroSense '96: Applications and Science of Artificial Neural Networks II (SPIE Proc. No. 2760), 1996, pp. 152-163.
[12] A.T. Nelson, E.A. Wan, Neural Speech Enhancement Using Dual Extended Kalman Filtering, ICNN, 1997, pp. 2171-2175.
[13] R.G. Brown, Random Signal Analysis and Kalman Filtering, Wiley, New York, 1983.
[14] R.J. Williams, Some Observations on the Use of the Extended Kalman Filter as a Recurrent Network Learning Algorithm, NU-CCS-92-1, Northeastearn University, 1992
[15] S. Shah, F. Palmieri, M. Datum, Optimal Filtering Algorithms for Fast Learning in Feedforward Neural Networks, Neural Networks 5 (1992) 779-787.
[16] G.V. Puskorious, L.A. Feldkamp, Decoupled Extended Kalman Filter Training of Feedforward Layered Networks, IJCNN, 1991, pp. 771-777.
[17] Sum, J., *et al.*, On the Kalman Filtering Method in Neural-Network Training and Pruning, IEEE Trans. Neural Networks 10 (1999) 161-166.
[18] Y. Iiguni, *et al.*, A Real-Time Learning Algorithm for a Multilayered Neural Network Based on the Extended Kalman Filter, IEEE Trans. Neural Networks 40 (1992) 959-966.
[19] L. Ljung, T. Soderstrom, Theory and Practice of Recursive Identification, MIT, Cambridge, 1983.
[20] S.E. Fahlman, C. Lebiere, The cascade-correlation learning architecture, NIPS2, Morgan Kaufmann, San Mateo, CA, 1990, pp. 524-532.
[21] R. Lengelle, T. Den∩ux, Training MLPs Layer by Layer Using an Objective Function for Internal Representations, Neural Networks 9 (1996) 83-97.
[22] J. Park, I.W. Sandberg, Approximation and radial basis function networks, Neural Computation 5 (1993) 305-316.
[23] J. Platt, A resource allocating network for function interpolation, Neural Computation 3 (1991) 213-225.
[24] A.U. Levin, T.K. Leen, J.E. Moody, Fast Pruning Using Principal Components, in: J. Cowan, G. Tesauro, J. Alspector, (Eds.), Advances in Neural Information Processing Systems 6, Morgan Kaufmann, San Mateo, CA, 1994, pp.
[25] J. Moody, The effective number of parameters: An analysis of generalisation and regularisation in nonlinear learning systems, in: J.E. Moody, S.J. Hanson, R.P. Lippmann, (Eds.), Advances in Neural Information Processing Systems 4, Morgan Kaufmann , San Mateo, CA, 1992, pp. 847-854.
[26] J. Sum, *et al.*, Extended Kalman Filter-Based Pruning Method for Recurrent Neural Networks, Neural Computation 10 (1998) 1481-1505.

Table 1:

	Training set							
No. centres	DEKF		GEKF		KF_weights		LMS	
	NMSE ($*10^{-3}$)	R	NMSE ($*10^{-3}$)	R	NMSE ($*10^{-3}$)	R	NMSE ($*10^{-3}$)	R
5	0.12	0.995	0.13	0.99	0.14	0.99	2.9	0.98
7	0.1	0.995	0.5	0.99	0.1	0.99	2	0.98
10	0.04	0.995	0.05	0.99	0.07	0.99	0.4	0.98
	Test set							
No. centres	DEKF		GEKF		KF_weights		LMS	
	NMSE ($*10^{-3}$)	R	NMSE ($*10^{-3}$)	R	NMSE ($*10^{-3}$)	R	NMSE ($*10^{-3}$)	R
5	0.14	0.994	0.15	0.98	0.16	0.98	3.1	0.97
7	0.12	0.994	0.6	0.98	0.11	0.984	2.5	0.983
10	0.06	0.994	0.07	0.98	0.08	0.986	0.8	0.98

Table 2:

		DEKF	GEKF	KF_weights	LMS
NMSE_train ($*10^{-3}$)		0.41	1.28	3.59	8.05
No. of training epochs		< 100	< 100	< 100	> 5000
CPU time (s/training epoch)		70	200	6	-
Classification accuracy	Train set	97%	98%	95%	98%
	Test set	97%	98%	96%	98%

Table and Figure Caption

Table 1: Simulation results for chaotic time series prediction

Table2: Simulation results for the 2-spirals classification problem

Fig. 1: Quadratic map chaotic time series prediction using DEKF algorithm (5 centres): phase plot for the test set

Fig. 2: Typical evolution of NMSE error on the training set for the prediction application:
a) DEKF algorithm (circles - 5 centres, dotted line - 7 centres, solid line - 10 centres)
b) 5 centres: GEKF (solid line), DEKF (dotted line), KF_weights (circles)

Fig. 3: Simulation results for the 2-spirals classification problem (96 centres):
a) training data; b) DEKF algorithm; c) GEKF algorithm; d) gradient descent

Fig. 4: Simulation results for pruned networks (75 centres, DEKF) on the test set:
a) PCP; b) Kalman-based pruning

Fig. 5: Performance evolution on the training set during the pruning process (solid line – Kalman-based pruning; dotted line – PCP):
a) NMSE ; b) classification accuracy

Footnote:

[1]The DEKF notation should not be confused with the one in [9], which stands for Decoupled Extended Kalman Filter.

Figure 1:

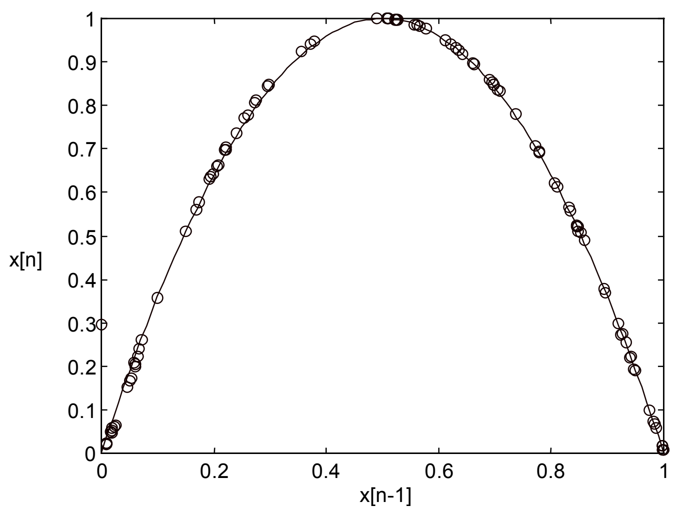


Figure 2a:

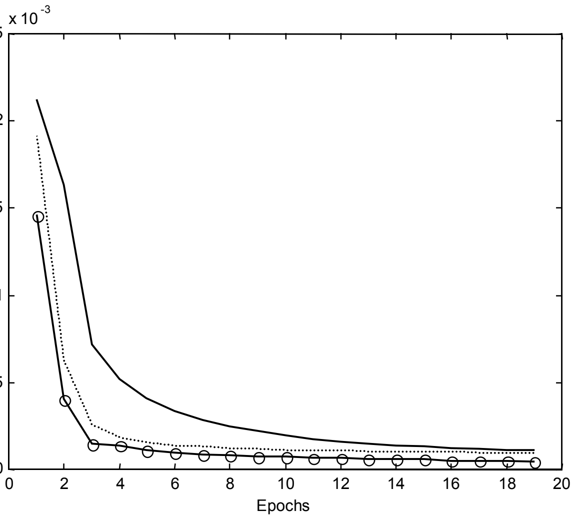


Figure 2b:

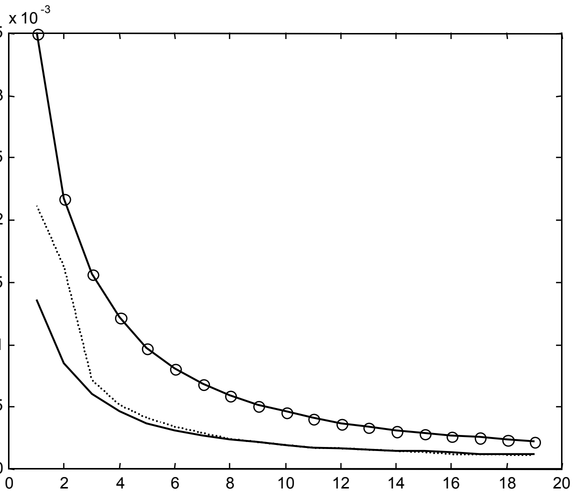
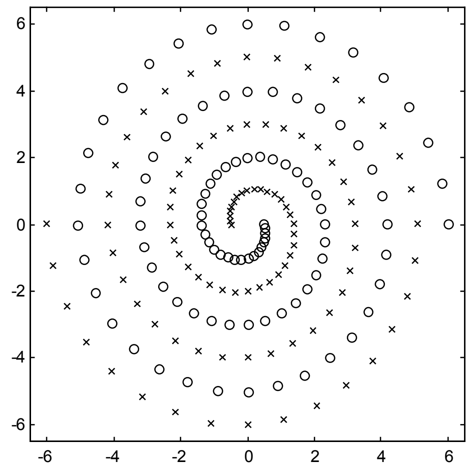
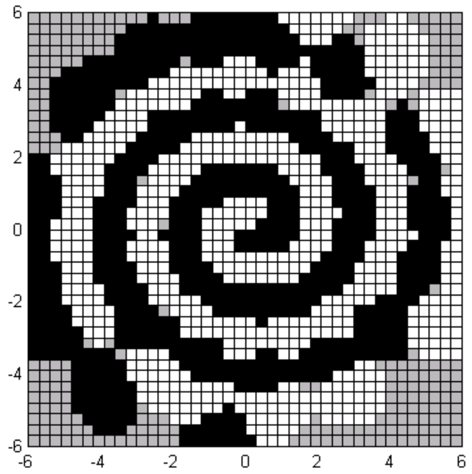


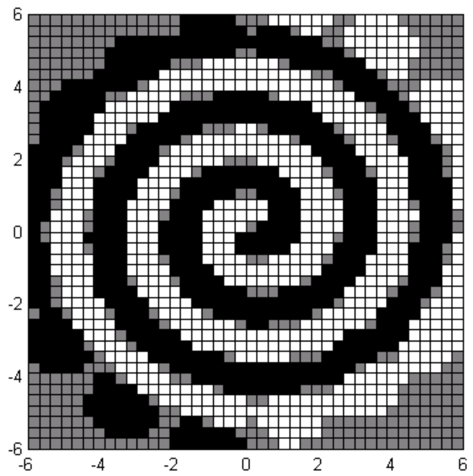
Figure 3:



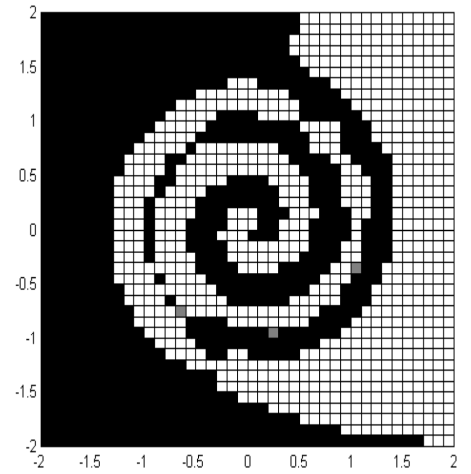
a)



b)

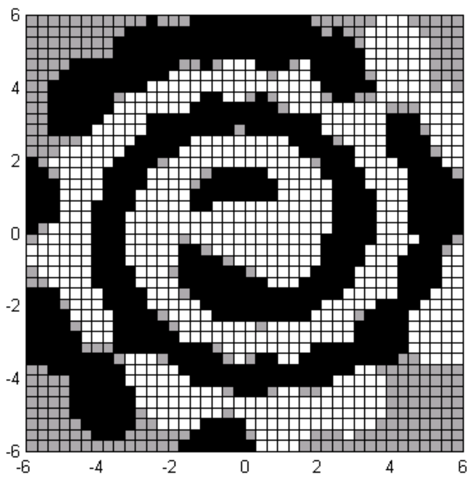


c)

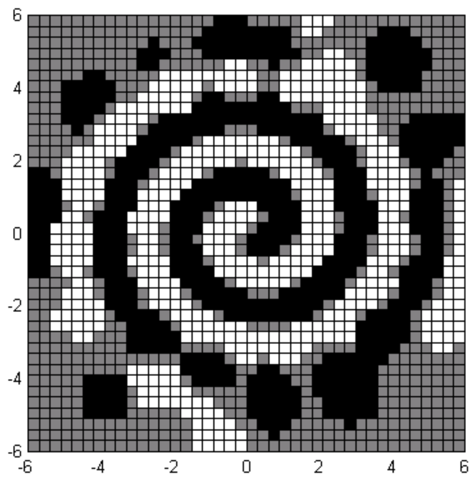


d)

Figure 4:

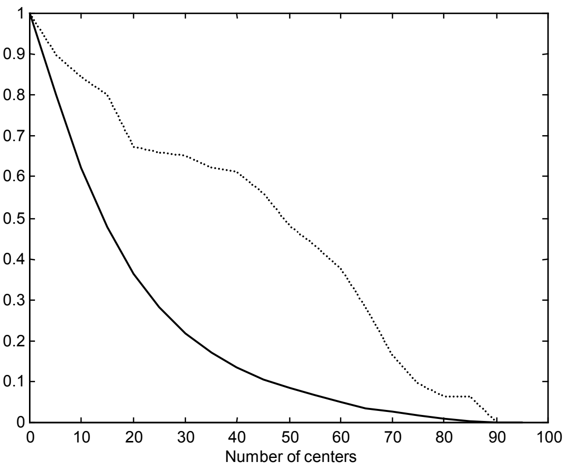


a)

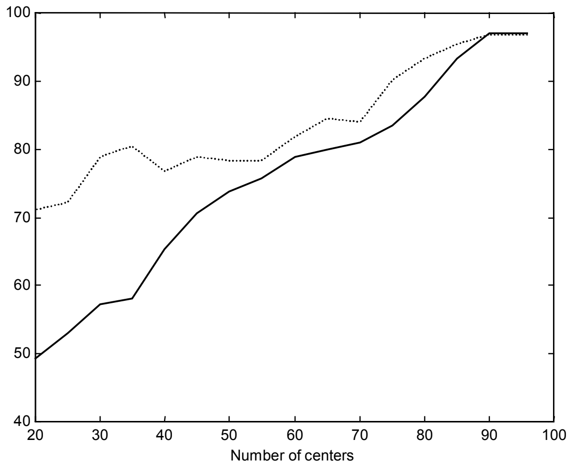


b)

Figure 5:



a)



b)