Efficient training of RBF neural networks for pattern recognition.

Efficient Training of RBF Neural Networks for Pattern Recognition

Francesco Lampariello and Marco Sciandrone

***Abstract*—The problem of training a radial basis function (RBF) neural network for distinguishing two disjoint sets in R^n is considered. The network parameters can be determined by minimizing an error function that measures the degree of success in the recognition of a given number of training patterns. In this paper, taking into account the specific feature of classification problems, where the goal is to obtain that the network outputs take values above or below a fixed threshold, we propose an approach alternative to the classical one that makes use of the least-squares error function. In particular, the problem is formulated in terms of a system of nonlinear inequalities, and a suitable error function, which depends only on the violated inequalities, is defined. Then, a training algorithm based on this formulation is presented. Finally, the results obtained by applying the algorithm to two test problems are compared with those derived by adopting the commonly used least-squares error function. The results show the effectiveness of the proposed approach in RBF network training for pattern recognition, mainly in terms of computational time saving.**

***Index Terms*—Error functions, neural-network training, pattern recognition.**

I. Introduction

FEEDFORWARD neural networks have increasingly been used in many areas for the solution of difficult real-world problems. This is due to the approximation capability of these devices, i.e., to the property that any continuous function can be approximated within an arbitrary accuracy by means of a neural network, provided that its topology includes a sufficient number of hidden nodes (see, e.g., [1]–[4]). Once the architecture has been defined, the network is determined by performing a training process, which can be viewed as a nonlinear optimization problem where the goal is to find the network parameters that minimize a suitable error function. This is done by using a given number of pattern-target pairs, that are samples of the input–output mapping to be approximated.

Our attention in this paper is focused on the problem of training radial basis function (RBF) networks in the field of pattern recognition, and more specifically for classification problems where the task is to assign a label to one of a number of discrete classes or categories. In principle, even multilayer perceptron (MLP) neural networks could be considered instead of RBF networks. In this connection, we observe that in MLP the sigmoidal function is used as activation function, so that saturation of a number of node outputs may occur, i.e., a number of hidden nodes may become insensitive to the training process. Indeed, prior to training MLP, it is necessary to choose a suitable initialization range for the network parameters in order to ensure that the network activations lie within the normal operating region of the sigmoidal function. However, saturation cannot be easily avoided during the entire minimization, and hence it may cause the training process to slow down. For this reason we limit ourselves to address classification problems via RBF neural networks, where saturation does not occur by a suitable choice of the activation function.

The error function commonly used for training a neural network is the *least-squares* function. Different error functions can be adopted, such as the *cross-entropy* [5] or the *exponential* [6] function, in order to avoid some undesirable effects related to the use of the sum of squares. In the minimization of an error function of these kinds, the attempt is to reduce as much as possible the differences between the network outputs corresponding to the given inputs and the label values associated with the training patterns. In classification problems, however, it is sufficient to obtain that the network outputs take values above or below a fixed threshold. During the training process based on the foregoing error functions, even the patterns that are already correctly classified with respect to the threshold value give a contribution to the overall network error. This may imply a poor convergence rate of the optimization algorithm applied, since these patterns will unnecessarily influence both the search direction and the steplength used by the algorithm.

On the basis of these observations, we formulate the training problem simply in terms of a system of nonlinear inequalities and, for solving it, we consider a *threshold* error function to which only the patterns corresponding to the violated inequalities give a contribution. Then, making use of a standard routine for minimizing an error function of this kind, a specific algorithm is designed by introducing two scalar parameters, which represent the upper and lower reference levels for the network outputs with respect to the fixed threshold. The values of these parameters are automatically updated during the training process, according to the progress made in the recognition of the training patterns. In this way, it is possible to avoid that the algorithm will converge to points of the parameter space which do not provide an acceptable solution of the training problem.

Finally, the results obtained by applying the algorithm to real world data are compared with those obtained by adopting the commonly used least-squares error function.

II. RBF Networks

A feedforward neural network is a computing device whose processing units (the *nodes*) are distributed in adjacent layers connected through unidirectional links (the *weights*). In particular, a RBF network is a fully connected network with one “hidden” layer, whose nodes have some radially symmetric

Manuscript received July 30, 1999; revised July 31, 2000.

The authors are with the Istituto di Analisi dei Sistemi ed Informatica, CNR, 00185 Rome, Italy (e-mail: lampariello@iasi.rm.cnr.it; sciandro@iasi.rm.cnr.it).

Publisher Item Identifier S 1045-9227(01)07568-3.

function as activation function. Such a network implements an input–output mapping $f\colon R^n \rightarrow R$ according to

$$f(x) = \sum_{i=1}^{n_r} \lambda_i \phi(\|x - c^i\|)$$

where

$x \in R^n$	input vector;
$\phi(\cdot)\colon R^+ \rightarrow R$	radially symmetric function;
$\|\cdot\|$	Euclidean norm;
$\lambda_i \in R, i = 1, \ldots, n_r$	weights;
$c^i \in R^n, i = 1, \ldots, n_r$	RBF *centers*;
n_r	number of hidden nodes.

Typical choices for the function $\phi(\|x - c^i\|)$ are

$\exp(-\|x - c^i\|^2/2\sigma^2)$	(Gaussian function)
$(\|x - c^i\|^2 + \sigma^2)^{1/2}$	(Direct multiquadric function)
$(\|x - c^i\|^2 + \sigma^2)^{-1/2}$	(Inverse multiquadric function)

where $\sigma > 0$ is the so-called "shift parameter."

The method of RBFs has been used in the theory of *multivariable interpolation* in high-dimensional space [7], [8]. It has been shown (see, e.g., [2] and [3]) that the RBF network is a universal approximator for continuous functions, provided that the number n_r of the hidden nodes is sufficiently large. This property makes the network a powerful tool for dealing with many real world problems. From both theoretical and practical investigations it appears that the performance of the RBF network is not greatly influenced by the choice of the activation function ϕ. However, as observed in the introduction, the use of the direct multiquadric function will avoid automatically saturation of the node outputs.

In order to approximate a given nonlinear mapping, the network parameters λ_i, c^i, $i = 1, \ldots, n_r$ have to be determined by using a finite set of input–output data (training process).

As regards the centers c^i, they may be chosen randomly among the input data or fixed at specific locations using some "clustering" technique (see, e.g., [9]), i.e., placed in the regions where the input data are more meaningful. Then, the weights λ_i are computed by applying an optimization algorithm for minimizing a suitable cost function.

A different approach [4] is based on a supervised learning process involving both the weights and the centers of the network. Although this approach is more complex and computationally more expensive, it usually leads to an improvement of the network performance, and hence it will be adopted here.

III. Pattern Recognition via Neural Networks

Let us consider two disjoint point sets A and B in R^n. A reference value, say 1, is associated to the points in A and a different value, say 0, to those in B. The classification problem consists in distinguishing the two point sets, i.e., given an arbitrary point $x \in A \cup B$, in recognizing whether x belongs to A or to B. This problem can be dealt with by using a feedforward neural network. In particular, we consider a RBF network that implements the input–output mapping $f(w)\colon R^n \rightarrow R$, where the parameter vector $w \in R^{n_r(n+1)}$ is composed of the weights and the centers λ_i, c^i, $i = 1, \ldots, n_r$. By using a given number N_p of pattern-target pairs (the training set)

$$\text{TS} = \{(x_p, d_p) \in A \cup B \times \{0, 1\},\ p = 1, \ldots, N_p\}$$

w is to be determined (network training) in order to obtain that

$$\begin{aligned} &\forall x \in A \Rightarrow f(x; w) = 1 \\ &\forall x \in B \Rightarrow f(x; w) = 0. \end{aligned}$$

The network is trained by minimizing an error function $E(w)$ that measures the degree of success in the recognition of the training patterns in the set TS

$$E(w) = \sum_{(x_p, d_p) \in \text{TS}} E_p(w)$$

where $E_p(w)$ is the contribution of pattern x_p to the overall network error. Then, starting from a prefixed vector $w(0)$, new values of the parameters are iteratively determined to reduce the network error, according to a scheme of the form

$$w(k+1) = w(k) + \alpha(k)\, d(k)$$

where $\alpha(k)$ is the stepsize (*learning rate*) along the search direction $d(k)$. Let $w^\star$ be a vector such that $E(w^\star)$ is sufficiently small. Then, a given point x is recognized to belong to A or to B by computing $z = f(x, w^\star) - 0.5$, and using the step function that maps negative numbers z into $\{0\}$ and nonnegative numbers z into $\{1\}$.

The commonly used error function is the *least-squares* function

$$E(w) = 1/2 \sum_{(x_p, d_p) \in \text{TS}} (d_p - f(x_p, w))^2 \qquad (1)$$

or a normalized version of it.

It has been observed that the use of an error function of this kind may have undesirable effects. For instance, it is possible that the training algorithm will converge toward regions of the parameter space where the wide majority of patterns are correctly classified, while a small number of these are severely misclassified. This may imply poor subsequent improvements in the training process, since the contribution of the few misclassified patterns to the overall error function is overcomed by that of the numerous correctly classified ones.

In order to avoid this drawback, some alternative error functions have been proposed. In particular, Solla *et al.* [5] have considered the *cross-entropy* (or *logarithmic*) function

$$E(w) = -\sum_{(x_p, d_p) \in \text{TS}} \ln\left[(f(x_p, w))^{d_p}(1 - f(x_p, w))^{1-d_p}\right]. \qquad (2)$$

Using this function, the contribution to the error gradient due to the misclassified patterns is significantly higher than using the least-squares function, making it easier for the algorithm to escape from bad regions of the parameter space. Note that it is necessary to ensure $0 < f(x_p, w) < 1$, by suitably rescaling the network outputs.

A different approach, proposed by Møller [6], is based on the use of the *exponential* error function

$$E(w) = 1/2 \sum_{(x_p,d_p)\in \mathrm{TS}} \cdot \exp\{-\alpha(f(x_p, w) - d_p + \beta)(d_p - f(x_p, w) + \beta)\} \quad (3)$$

where α and β are positive scalars. The function (3) incorporates the constraint $|d_p - f(x_p, w)| \leq \beta, \forall (x_p, d_p) \in$ TS, that establishes an "acceptable" error level β for the network output values with respect to the reference values. During the training process, the value of β, initially set to a relatively high level, is progressively reduced in size. The contribution of a severely misclassified pattern to this error function is significantly higher (depending on the choice of the parameter α) than those of the patterns for which the constraint is satisfied. In this way, it is unlikely that the training algorithm will converge toward bad regions of the parameter space.

We observe that, in the minimization of (1)–(3), the goal is to reduce as much as possible the differences between the network output values corresponding to the given inputs and the reference values associated with the training patterns, i.e., the algorithm searches for a vector w such that $f(x_p, w)$ is as close as possible to 1 or to 0.

In pattern recognition problems, however, it is sufficient to obtain that the network output takes values above or below a fixed *threshold value* τ, i.e., we have to find w such that $f(x_p, w) \geq \tau$ or $f(x_p, w) < \tau$. Therefore, at a given stage of the training process based on the foregoing error functions (in this case, for reference values 0 and 1, $\tau = 0.5$), even the patterns $(x_p, 1)$ for which $f(x_p, w)$ is greater than or equal to 0.5, and those $(x_p, 0)$ for which $f(x_p, w)$ is lower than 0.5, although already correctly classified, give a contribution to the overall error function. This may imply a poor convergence rate of the algorithm used for minimizing $E(w)$, since these contributions will unnecessarily influence both the search direction and the steplength, which are calculated from the objective function.

IV. An Alternative Formulation of the Training Problem for Pattern Recognition

We propose here an alternative approach that takes into account the specific feature of pattern recognition, by defining a suitable error function. In particular, the problem is formulated in terms of a system of nonlinear inequalities as follows.

Let $\varepsilon > 0$ be an arbitrarily small real number which represents the tolerance in distinguishing whether a network output value is above or below a fixed threshold τ. As discussed in the preceding section, the RBF network training problem consists of determining a vector $w \in R^{n_r(n+1)}$ such that the nonlinear inequalities (for simplicity and without loss of generality, we assume $\tau = 0$)

$$\begin{cases} f(x_p, w) \geq \varepsilon, & \forall (x_p, 1) \in \mathrm{TS}; \\ f(x_p, w) \leq -\varepsilon, & \forall (x_p, 0) \in \mathrm{TS} \end{cases} \quad (4)$$

are satisfied. Then, for solving system (4), we consider the following *threshold* error function

$$E(w) = \sum_{(x_p,\,1)\in \mathrm{TS}} (\max\{0, \varepsilon - f(x_p, w)\})^q + \sum_{(x_p,\,0)\in \mathrm{TS}} (\max\{0, \varepsilon + f(x_p, w)\})^q \quad (5)$$

where $q \geq 2$ is a given integer, which has the following properties:

1) $E(w)$ is nonnegative;
2) $E(w) = 0$ iff w is a solution of system (4);
3) $E(w)$ is continuously differentiable $q - 1$ times.

Therefore, by 1) and 2), the problem becomes that of finding global minima of (5).

We remark that an error function similar to (5) has been considered by Sontag and Sussmann in [10], where, with reference to multilayer perceptron networks, the properties of its local minima have been analyzed. However, as far as we are aware, the use of such an error function for addressing the problem of network training has not yet been proposed.

It is evident that function (5) is only dependent on the violated inequalities, so that, during its minimization, only the misclassified training patterns contribute to the overall error.

We observe that, since ε must be small, due to the structure of function (5), there are regions of the parameter space where the overall error is close to zero and, at the same time, the number of the misclassified patterns may be relatively high. Therefore, the minimization algorithm could be trapped in one of these regions, reaching a point where $E(w) \simeq 0$ with a very low gradient norm, which may not provide an acceptable solution of the training problem, although representing a "good" solution of the optimization problem. A similar situation cannot occur when the least-squares function is adopted as error function, since in this case an overall error close to zero implies a high percentage of correctly classified patterns. In other words, by using least-squares based error functions, a good approximation of a global minimum provides in any case a good solution of the training problem, whereas this correspondence may not hold when function (5) is adopted. In order to overcome this drawback, we substitute in the threshold error function the fixed tolerance ε with two adjustable scalar parameters $\delta_u > 0$ and $\delta_\ell > 0$, which represent the upper and lower reference levels for the network output values with respect to the fixed threshold. The values of these parameters are systematically reduced, during the minimization process, according to the progress made in the recognition of the training patterns.

In particular, after a certain number of iterations, the percentage of the patterns which have been correctly classified (classification accuracy), i.e., the percentage of the satisfied inequalities in system (4), is compared with that corresponding to the initial point. Then, the minimization process is continued with the same parameter values δ_u and δ_ℓ or with suitably reduced ones, depending on whether a sufficient increase in the classification accuracy has been obtained or not. This procedure is repeated until either all the training patterns have been correctly classified (or their number is sufficiently large), or both the parameters δ_u and δ_ℓ have been reduced at the prefixed tolerance.

We note that the use of two parameters instead of a single one, is advantageous since it is possible to better handle the contribution to the overall error of the patterns in each of the two training subsets $\{x_p, 1\}$ and $\{x_p, 0\}$.

Formally, we define the following threshold error training algorithm (TETA). Let δ_u and δ_ℓ be the adjustable scalar parameters, N_u the number of patterns $(x_p, 1) \in \text{TS}$ and n_u the number of these for which $f(x_p, w) \geq \varepsilon$, N_ℓ the number of patterns $(x_p, 0) \in \text{TS}$ and n_ℓ the number of these for which $f(x_p, w) \leq -\varepsilon$, so that $p_u = n_u/N_u$ and $p_\ell = n_\ell/N_\ell$ are the fractions of the correctly classified patterns in the two subsets, θ the minimum relative increase in the classification fraction required for maintaining the value of δ_u or δ_ℓ unaltered and N the maximum allowed number of iterations performed by a given algorithm used for minimizing the overall error.

The Training Algorithm TETA

Data: Let $w^{(i)} \in R^{n_r(n+1)}$ be the starting parameter vector, $\varepsilon > 0$, $\delta_u > \varepsilon$, $\delta_\ell > \varepsilon$, $\theta > 0$ real numbers and N a positive integer.

Step 0: Compute the fractions $p_u^{(i)}$ and $p_\ell^{(i)}$ corresponding to $w^{(i)}$ of the correctly classified patterns in the two training subsets of TS and the overall classification accuracy $p^{(i)} = (p_u^{(i)} N_u + p_\ell^{(i)} N_\ell)/N_p$.

Step 1: If $p^{(i)} = 1$, or if $\delta_u \leq \varepsilon$ and $\delta_\ell \leq \varepsilon$, stop.

Step 2: Starting from the point $w^{(i)}$, perform N iterations of a given algorithm for minimizing

$$E(w) = 1/2 \sum_{(x_p,\,1) \in \text{TS}} (\max\{0,\, \delta_u - f(x_p, w)\})^2 + 1/2 \sum_{(x_p,\,0) \in \text{TS}} (\max\{0,\, \delta_\ell + f(x_p, w)\})^2 \quad (6)$$

and let $w^\star$ be the reached point. Compute the corresponding fractions $p_u^\star, p_\ell^\star, p^\star = (p_u^\star N_u + p_\ell^\star N_\ell)/N_p$, and set $w^{(i)} := w^\star$.

Step 3: If $(p^\star - p^{(i)})/(1 - p^{(i)}) \geq \theta$, go to Step 5.

Step 4: Compute $\Delta_u = (p_u^\star - p_u^{(i)})/(1 - p_u^{(i)})$ and $\Delta_\ell = (p_\ell^\star - p_\ell^{(i)})/(1 - p_\ell^{(i)})$.

If $\Delta_u < \theta$ and $\Delta_\ell < \theta$, then Update δ_u and Update δ_ℓ.

If $\Delta_u \geq \theta$, then Update δ_ℓ if $0 \leq \Delta_\ell < \theta$, or Update δ_u if $\Delta_\ell < 0$.

If $\Delta_\ell \geq \theta$, then Update δ_u if $0 \leq \Delta_u < \theta$, or Update δ_ℓ if $\Delta_u < 0$.

Step 5: Set $p_u^{(i)} := p_u^\star, p_\ell^{(i)} := p_\ell^\star, p^{(i)} := p^\star$, and go to Step 1.

Update δ_u: Let S_u be the set: $S_u = \{(x_p, 1) \in \text{TS}: \varepsilon < f(x_p, w^{(i)}) < \delta_u\}$ and n_{S_u} the number of patterns in S_u. If $S_u \neq \emptyset$, set

$$\delta_u = 1/n_{S_u} \sum_{(x_p,\,1) \in S_u} f(x_p, w^{(i)}).$$

Update δ_ℓ: Let S_ℓ be the set: $S_\ell = \{(x_p, 0) \in \text{TS}: -\delta_\ell < f(x_p, w^{(i)}) < -\varepsilon\}$ and n_{S_ℓ} the number of patterns in S_ℓ. If $S_\ell \neq \emptyset$, set

$$\delta_\ell = 1/n_{S_\ell} \sum_{(x_p,\,0) \in S_\ell} f(x_p, w^{(i)}).$$

We observe first that the initial value of the parameters δ_u and δ_ℓ should be chosen sufficiently large (with respect to the tolerance ε), in order to obtain that the network output values corresponding to the patterns $(x_p, 1)$ become well separated from those corresponding to the patterns $(x_p, 0)$. On the other hand, the initial values should be of the same order of magnitude as the network outputs. Therefore, it appears reasonable to take, for δ_u and δ_ℓ, the absolute mean values of the outputs corresponding to the patterns that are initially already correctly classified in the two training subsets.

Starting from the initial point $w^{(i)}$, the optimization algorithm performs at most N iterations. The number N should be reasonably large in order to ensure a significant progress in the minimization of $E(w)$, so that the choice of N should be made taking into account the convergence properties of the algorithm used. After N iterations, if the overall classification accuracy has not been sufficiently increased or, possibly, it has been at once reduced, it is likely that the contribution to the overall error of the correctly classified patterns for which $\varepsilon < f(x_p, w) < \delta_u$ and of those for which $-\delta_\ell < f(x_p, w) < -\varepsilon$ has been prevailing over that of the misclassified patterns. In this case a reduction of the parameter δ_u, or δ_ℓ, or both, is beneficial, since some correctly classified patterns will no more give a contribution and hence, that of the misclassified patterns will be reinforced. In particular, we distinguish whether the classification accuracy has not been sufficiently increased in both or in one only of the two training subsets. In the first case, both the parameters δ_u and δ_ℓ are reduced. In the second case, if in both the training subsets the classification accuracy has been increased, only the parameter is reduced corresponding to the subset in which the increase was insufficient. If in one subset the classification accuracy is decreased, the parameter is reduced corresponding to the other subset. As regards the updating rule of δ_u and δ_ℓ, it is important that their values be reduced gradually. Then, we take as new values, the means of the outputs corresponding to the patterns in the sets S_u and S_ℓ, provided that the latter are not empty. Otherwise, the values δ_u and δ_ℓ remain unchanged.

Finally, it is possible that both the parameters δ_u and δ_ℓ become lower than or equal to ε, and the overall classification accuracy obtained is still unsatisfactory for the problem considered. In this case, since it is unlikely that a further progress could be obtained, the training algorithm should be restarted by performing a greater number N of iterations, or with a suitably reduced value of the parameter θ. Alternatively, a network having a greater number of hidden nodes could be considered.

As an example, we compare in the Figs. 1 and 2 the behavior of Algorithm TETA with that of a training algorithm (LSTA) that minimizes the least-squares function. Both algorithms use the same minimization routine (E04DGF of the NAG library). The implementative details are described in the next section.

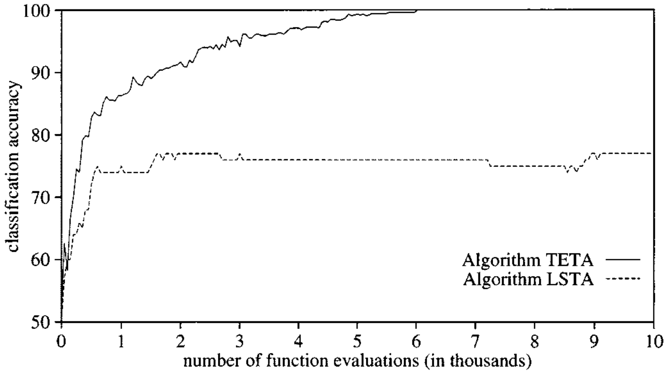

Fig. 1. Performance of the training algorithms for the two spirals problem.

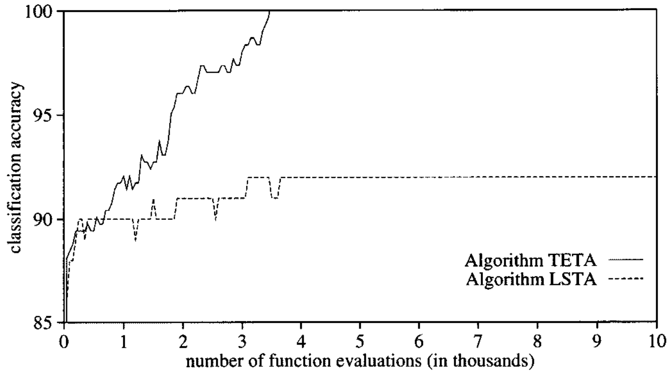

Fig. 2. Performance of the training algorithms for the heart problem.

Fig. 1 refers to the two spirals problem, a classical test problem, where the task is to discriminate between two sets of points that are arranged in two interlocking spirals in the plane. The training set consists of $N_p = 720$ patterns. The input space dimension is $n = 2$ and the network is composed of $n_r = 15$ hidden nodes.

Fig. 2 refers to the heart problem [11], where the task is to predict heart disease, i.e., to decide whether at least one of four major vessels is reduced in diameter by more than 50%. The decision is made based on personal data and results of various medical examination. The training set consists of $N_p = 303$ patterns. The input space dimension is $n = 35$ and the network is composed of $n_r = 25$ hidden nodes.

In order to properly compare the behavior of the two algorithms, the number n_r is taken sufficiently large so as to allow the correct classification of all patterns by the TETA algorithm. Indeed, to obtain perfect performance it is necessary that the set of points $w \in R^{n_r(n+1)}$ for which the system of nonlinear inequalities (4) is satisfied [and hence the global minimum of the error function (5) is zero] be not empty. This can be ensured provided that the number of network parameters is large enough. It is evident that, with the same number of hidden nodes, even by minimizing the least-squares error function it will be possible, in principle, to obtain perfect performance. However, the behavior of the two algorithms in terms of classification accuracy obtained during the minimization process shows that the training algorithm that uses the least-squares error function, after the initial progress, does not obtain further significant improvements in spite of an extended minimization process, while the TETA algorithm continues to obtain a progress. This confirms that the contribution to the overall error of the patterns that have been already correctly classified may have a significant influence on the behavior of a training algorithm (note that this outcome may result of specific importance in the case where the network training is to be continued using new available data). In particular, after 30 000 function evaluations, the LSTA algorithm obtains the classification accuracy of 75.6% for the two spirals problem and of 95.7% for the heart problem. Moreover, for instance with reference to the first problem, by increasing the number of nodes n_r at 20 and at 25, after 30 000 function evaluations, the percentage of correctly classified patterns becomes 95.5% and 98.9%, respectively. On the basis of these results, it is likely that the LSTA algorithm requires both more hidden nodes and more training iterations than the TETA algorithm to achieve perfect performance on a given set of patterns. It could be of some interest to establish how many more iterations or nodes are needed, but we observe that perfect performance is not the goal of network training, since, as well known, the overfitting effect will arise at the final stage of the training process.

A more interesting aspect is that the TETA algorithm obtains a high percentage of correctly classified patterns by performing a relatively small number of function evaluations, and hence requiring a limited computation time. Therefore, it is to be expected a significant time saving by using the TETA algorithm in conjunction with a strategy for ensuring a satisfactory generalization property of the trained network, i.e., for obtaining that even patterns not used in the training process will be correctly classified, which is the ultimate goal of learning. This aspect will be considered in the next section, where some experimental results are reported.

Finally, we observe that the proposed approach refers to 2-class classification problems. However, the same approach can be easily extended for classifying multiple subsets, for instance, as follows.

Let $A_1, A_2, \ldots, A_m$ be m disjoint point sets in R^n, and let us consider a RBF network having n_r hidden nodes and m outputs f_k: $R^n \rightarrow R$, $k = 1, \ldots, m$ (note that, for $m = 2$, this problem formulation is alternative to that adopted in this paper, where networks having a single output are considered). Then, we want to determine the network parameter vector $w \in R^{n_r(n+m)}$ in such a way that, for any pattern $x_p \in A_k$, the output f_k is above a fixed threshold τ and all the other network outputs are below it. This is equivalent, by assuming again $\tau = 0$, to solve the following system of nonlinear inequalities: for $k = 1, \ldots, m$

$$\begin{cases} f_k(x_p, w) \geq \varepsilon & \forall x_p \in A_k \\ f_k(x_p, w) \leq -\varepsilon, & \forall x_p \in A_j, \, j \neq k \end{cases}$$

where $\varepsilon > 0$ is the arbitrarily small tolerance.

The search for such a vector w can be performed by minimizing the following threshold error function:

$$E(w) = \sum_{k=1}^{m} \left[\sum_{x_p \in A_k} (\max\{0, \varepsilon - f_k(x_p, w)\})^q \right.$$
$$\left. + \sum_{x_p \in A_j, \, j \neq k} (\max\{0, \varepsilon + f_k(x_p, w)\})^q \right].$$

We can again substitute in $E(w)$ the fixed tolerance ε with two adjustable scalar parameters δ_u and δ_ℓ, or it is even possible to use a different pair δ_u, δ_ℓ for each of the m subsets. This last choice, i.e., the use of multiple "adjustable thresholds," may give rise to a more flexible behavior of the algorithm that minimizes $E(w)$, although the procedure for updating them may become more complex. An in depth investigation of this aspect could be the topic of further work.

V. Experimental Results

Among the various strategies proposed in the literature for achieving good generalization, we adopt here the easy method of performing "early stopping." In particular, the available data are split into three disjoint sets, training, validation and test. Then, during the learning process performed by using the patterns in the training set, the network is periodically tested on the validation set, every time Step 2 of the TETA algorithm is completed, i.e., after every N steps of the minimization algorithm. The optimization is stopped when the classification accuracy on the validation set is lower than the best value achieved five times consecutively. Finally, the generalization capability of the trained network is evaluated by computing the classification accuracy on the test set.

We report here the results obtained by applying the TETA algorithm in conjunction with the described strategy to two classification problems which consist of real world data taken from the PROBEN1 benchmark site [11]. The aim of the experiments is to show the efficiency of the proposed approach in terms of computational time saving, by comparing the numerical results with those obtained by adopting the commonly used least-squares error function. Obviously, even in this case the early stopping is performed by testing the network every N iterations.

The TETA algorithm has been implemented by applying, for minimizing the overall error function (6), a preconditioned limited memory quasi-Newton conjugate gradient method (E04DGF routine, NAG library). Taking into account the effectiveness of this standard routine, we have set the maximum number of iterations at the value $N = 100$. The values chosen for the other parameters are $\varepsilon = 10^{-6}$ and $\theta = 0.01$.

In the starting parameter vector, the weights λ_i, $i = 1, \ldots, n_r$, are randomly chosen in the interval $[-0.5, 0.5]$, whereas the centers c^i, $i = 1, \ldots, n_r$ are randomly chosen among the input vectors x_p.

As regards the LSTA algorithm, we used again the E04DGF routine, starting from the same initial points as our algorithm.

The RBF networks are composed of hidden nodes having the direct multiquadric function as activation function and the shift parameter is set to $\sigma = 0.1$. The choice of the number n_r of hidden nodes should be made in accordance with the problem complexity. This is related on one hand to the input space dimension and the number of the training patterns, and on the other, to the nonlinearity degree of the decision surface (note, in particular, that with linearly separable subsets, one hidden node only is sufficient to achieve perfect performance independent of the problem dimension). Since we have no control over the latter factor, we report here for both the problems considered the results obtained with different values of n_r. As suggested in [11], the sizes of the training, validation and test sets are 50%, 25%, and 25% of the available data, respectively.

Cancer *Problem [11]*

This problem deals with the diagnosis of breast cancer. The task is to classify a tumor as either benign or malignant based on cell descriptions obtained by microscopic examination. This data set was created for the "Breast Cancer Wisconsin" of the University of Wisconsin Hospitals, Madison [12].

The available data consist of 699 patterns. The training set is composed of $N_p = 350$ patterns, while 175 and 174 are used for validation and test, respectively. The input space dimension is $n = 9$ and we consider networks with $n_r = 5, 10, 15, 20$ and 25.

Credit Card *Problem [11]*

The task is to predict the approval or nonapproval of a credit card for a customer.

The available data consist of 690 patterns. The training set is composed of $N_p = 345$ patterns, while 173 and 172 are used for validation and test, respectively. The input space dimension is $n = 51$ and we consider networks with $n_r = 10, 15, 20, 25$ and 30.

The results obtained for these classification problems, starting from three different initial points (a, b, c), are shown in Tables I and II, where the performance of the TETA and LSTA algorithms is given in terms of classification accuracy for each pattern subset, and of CPU time (in seconds).

We observe first that the behavior of the two algorithms in terms of classification accuracy is substantially similar, independent of the number of hidden nodes, although, as regards the generalization property, the LSTA algorithm obtains in most cases a slightly better percentage. This fact may be explained by observing that, using the TETA algorithm, less and less training patterns are qualified as misclassified during the training process, and hence, the drastic reduction of the number of patterns involved may give rise to the overfitting effect, which, however, is limited by the early stopping strategy. As regards the computational cost, the CPU time employed by the TETA algorithm is lower than that employed by the LSTA algorithm in 24 runs over the 30 performed. The time saving is appreciable for both problems in most runs, particularly for the Cancer problem. In summary, the TETA algorithm performs the network training more quickly than the LSTA algorithm does, with less accurate results, but the worsening is very limited.

It is worth noting that, for the cases corresponding to $n_r = 5$ in Table I, the TETA algorithm stops because both the adjustable thresholds δ_u and δ_ℓ have been reduced below the fixed tolerance ε (Step 1), and not by the effect of the early stopping criterion. Nevertheless, the generalization property of the trained network is good, although, as observed, slightly worse than that obtained by the LSTA algorithm, with computation times clearly lower. On the other hand, even taking into account the results corresponding to $n_r = 10$ in Table II, it does seem that the use of networks with a low number of hidden nodes is not advisable when the TETA algorithm is applied for training them. In fact, the system of nonlinear inequalities (4) could not

TABLE I
COMPARISON OF THE CLASSIFICATION ACCURACY OBTAINED AND THE CPU TIME EMPLOYED BY ALGORITHMS TETA AND LSTA (FIRST AND SECOND NUMBER, RESPECTIVELY, IN EACH COLUMN) FOR THE CANCER PROBLEM, STARTING FROM THREE DIFFERENT INITIAL POINTS. n_r IS THE NUMBER OF HIDDEN NODES

	i.p.	training (%)	validation (%)	test (%)	*time* (sec.)
$n_r = 5$	a	96.30 - 96.87	98.28 - 98.28	97.15 - 99.43	215 - 4234
	b	95.44 - 96.30	97.70 - 98.28	97.15 - 98.28	71 - 231
	c	95.73 - 96.30	98.28 - 98.28	97.70 - 98.85	247 - 3205
$n_r = 10$	a	96.87 - 99.14	98.28 - 98.85	97.70 - 98.28	221 - 1436
	b	95.73 - 97.15	98.28 - 98.28	97.70 - 98.85	49 - 610
	c	98.28 - 97.15	98.28 - 98.28	98.28 - 98.85	230 - 612
$n_r = 15$	a	95.73 - 98.58	98.28 - 98.85	97.70 - 98.85	185 - 1131
	b	96.59 - 98.87	98.85 - 98.85	97.15 - 98.85	108 - 2065
	c	98.85 - 96.59	98.28 - 98.85	98.28 - 98.28	240 - 225
$n_r = 20$	a	99.72 - 96.59	98.85 - 98.28	98.28 - 98.28	271 - 49
	b	97.15 - 98.28	98.85 - 98.85	98.85 - 99.43	130 - 1802
	c	98.58 - 99.14	98.28 - 98.85	96.55 - 98.28	323 - 3307
$n_r = 25$	a	99.43 - 98.58	98.85 - 98.28	98.28 - 98.85	394 - 2917
	b	99.43 - 98.85	98.85 - 98.85	98.85 - 99.43	342 - 4788
	c	99.43 - 98.58	98.28 - 98.85	97.70 - 99.43	345 - 3780

TABLE II
COMPARISON OF THE CLASSIFICATION ACCURACY OBTAINED AND THE CPU TIME EMPLOYED BY ALGORITHMS TETA AND LSTA (FIRST AND SECOND NUMBER, RESPECTIVELY, IN EACH COLUMN) FOR THE CREDIT CARD PROBLEM, STARTING FROM THREE DIFFERENT INITIAL POINTS. n_r IS THE NUMBER OF HIDDEN NODES

	i.p.	training (%)	validation (%)	test (%)	*time* (sec.)
$n_r = 10$	a	88.73 - 89.02	88.96 - 90.12	86.05 - 87.80	1214 - 1532
	b	88.44 - 87.28	88.96 - 89.54	85.46 - 86.63	810 - 571
	c	88.16 - 87.57	90.12 - 89.54	86.63 - 86.63	1229 - 234
$n_r = 15$	a	87.87 - 88.44	88.38 - 89.54	86.05 - 86.63	458 - 525
	b	89.60 - 87.87	88.38 - 89.54	87.20 - 86.63	183 - 353
	c	88.44 - 90.18	89.54 - 88.96	86.05 - 88.96	456 - 2614
$n_r = 20$	a	90.18 - 88.73	88.38 - 89.54	87.20 - 87.20	489 - 248
	b	91.33 - 89.30	88.38 - 89.54	88.96 - 87.80	487 - 1175
	c	88.73 - 89.02	89.54 - 89.54	87.80 - 87.20	490 - 718
$n_r = 25$	a	90.45 - 89.60	88.38 - 89.54	87.80 - 87.80	1817 - 901
	b	91.05 - 91.33	87.80 - 89.96	88.38 - 89.54	1517 - 3466
	c	92.49 - 89.02	87.20 - 89.54	86.63 - 88.38	906 - 1215
$n_r = 30$	a	91.33 - 89.30	87.80 - 89.54	88.96 - 88.38	390 - 722
	b	90.75 - 91.33	88.38 - 88.96	87.80 - 88.38	185 - 4858
	c	88.73 - 90.18	89.54 - 88.96	88.38 - 88.96	777 - 1767

admit a solution when the number of network parameters is relatively small with respect to that of the training patterns. In this case, the TETA algorithm, which concentrates on the misclassified patterns only, will not easily obtain further improvements during the minimization process, since, unavoidably, some patterns that were correctly classified will become misclassified.

In conclusion, although the numerical experience reported here is not particularly extensive, on the basis of the results obtained it appears that the use of a threshold-type error function and the definition of a suitable algorithmic scheme for handling its minimization represent an attractive approach in RBF network training for pattern recognition.

REFERENCES

[1] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, pp. 359–366, 1989.

[2] E. J. Hartman, J. D. Keeler, and J. M. Kowalsky, “Layered neural networks with Gaussian hidden units as universal approximators,” *Neural Comput.*, vol. 2, pp. 210–215, 1990.

[3] F. Girosi and T. Poggio, “Networks and the best approximation property,” *Biol. Cybern.*, vol. 63, pp. 169–176, 1990.

[4] T. Poggio and F. Girosi, “Networks for approximation and learning,” *Proc. IEEE*, vol. 78, no. 9, pp. 1481–1497, 1990.

[5] S. A. Solla, E. Levin, and M. Fleisher, “Accelerated learning in layered neural networks,” *Complex Syst.*, vol. 2, pp. 625–640, 1989.

[6] M. Møller, “Efficient training of feedforward neural networks,” Ph.D. dissertation, Daimi PB-464, Computer Science Department, Aarhus University, 1993.

[7] C. A. Micchelli, “Interpolation of scattered data: Distance matrices and conditionally positive definite function,” *Construct. Approx.*, vol. 2, pp. 11–22, 1986.

[8] M. J. D. Powell, “Radial basis function approximations to polynomials,” in *Proc. 12th Biennial Numerical Anal. Conf.*, Dundee, 1987, pp. 223–241.

[9] M. T. Musavi, W. Ahmed, K. h. Chan, K. B. Faris, and D. M. Hummels, “On the training of radial basis function classifiers,” *Neural Networks*, vol. 5, pp. 595–603, 1992.

[10] E. D. Sontag and H. J. Sussmann, “Backpropagation separates where perceptrons do,” *Neural Networks*, vol. 4, pp. 243–249, 1991.

[11] L. Prechelt, “PROBEN1-A set of neural network benchmark problems and benchmarking rules,” Fakultät für Informatik, Universität Karlsruhe, Karlsruhe, Germany, Tech. Rep. 21/94, 1994.

[12] O. L. Mangasarian and W. H. Wolberg, “Cancer diagnosis via linear programming,” *SIAM News*, vol. 23, pp. 1–18, 1990.