Training Radial Basis Function Networks Using the Differential Evolution Based Approaches

Junhong Liu Saku Kukkonen Jouni Lampinen
Department of Information Technology, Lappeenranta University of Technology
P.O.Box 20, FIN–53851 Lappeenranta
email: liu@lut.fi, Saku.Kukkonen@lut.fi, jlampine@lut.fi

***Abstract*— The Differential Evolution algorithm (DE) is a floating-point encoded evolutionary algorithm for global optimization. It has been demonstrated to be an efficient, effective, and robust optimization method, especially for problems containing continuous variables. This paper concerns applying DE and two DE-based methods to training Radial Basis Function (RBF) networks with variable widths. The Euclidean distance and the Mean Square Error from the desired outputs to the actual network outputs are applied as the objective functions to be minimized. Training the networks is demonstrated by approximating a set of functions using three approaches related to DE and two of them are also applied to reconstructing the spectra of oil samples and classification. In the experiments, each approach works effectively, while the DE-based growing approach is the most efficient one and a comparison of the net performances with another approach reported in the literature is performed and shows the resulting network generally performs well. The results show that the DE-based methods are potential ways to train Gaussian RBF networks.**

I. INSTRUCTION

Radial Basis Functions (RBFs) emerged as a variant of artificial neural networks in the late 80's. RBFs are embedded in a three layer neural network, i.e., the input layer, the hidden layer, and the output layer, where each hidden unit implements a radial activation function. The output units implement a weighted sum of hidden unit outputs. Approximation capabilities of RBFs have been studied in [1], [2]. Due to their nonlinear approximation properties, RBF networks are able to model complex mappings [3], [4]. RBFs have been used to build a class of nonlinear models, i.e., RBF models for multivariate approximation partially because the RBF models have the properties of localization, boundedness, stability, good interpolation, and smoothness.

The performance of a trained RBF neural network depends on the number of the radial basis functions as well as their locations, shapes, widths, and the method used for learning the input-output mapping. Finding the variable factors of RBFs is called network training. If a set of input-output pairs, called the training set, is at hand, the network parameters are optimized in order to fit the network outputs to the given inputs. The fit is evaluated by means of a cost function. After training, the RBF network can be used to respond to data whose underlying statistics is similar to that of the training set. Different approaches for training radial basis function networks have been developed and can be divided into three categories [5], [6], [7], [8], [9], [10], [11], [12], [13]: (i) Learning the centres and widths in the hidden layer; (ii) Learning the connection weights from the hidden layer to the output layer; (iii) Learning the network structure; and (iv) hybrid learning: learning the centres, widths, weights, or the network structure together. On-line training algorithms adapt the network parameters to the changing data statistics [6], [14], [15], [16]. RBF networks have been successfully applied to a large diversity of applications including interpolation [17], [18], signature recognition [10], heart disease classification [7], production and process control [19], handwritten digit recognition [20], radar target recognition [21], image restoration [22], 3-D object representation [23], motion estimation and moving object segmentation [24].

Artificial neural networks are widely recognized for their ability to approximate complicated non-linear relationships and to estimate underlying trends, even when substantial noise is present in the data. But it is often difficult to design appropriate neural network models since the basic principles governing the processing of information in neural networks are not well understood. As a result, conventional design techniques usually become inapplicable when applied to mapping the complex interactions among network units. Methods, which are more efficient, are required for the development of neural processing systems [13], such as Evolutionary techniques that fit this purpose because of the following reasons:

1) Evolutionary algorithms have been successfully applied to finding the global optima of various multidimensional functions where local optima in the space of possible solution are common.
2) Evolutionary algorithms are able to handle the optimization of parameters for which no gradient information or other auxiliary information is available.
3) Evolutionary algorithms can optimize a broader range of network parameters; even adaptively change the type of the transfer functions of nodes.

The Differential Evolution algorithm (DE) introduced by Price and Storn [25], a floating-point encoded Evolutionary Algorithm (EA) for global optimization, has been found to be an efficient, effective, and robust optimization method, especially for problems containing continuous problem variables [25], [26], [27]. The DE algorithm has become popular and has been used in many practical cases mainly because it has demonstrated good convergence properties and its basic

principles are easy to understand. DE is also particularly easy to work with, having only a few control variables. This paper applies DE to finding the set of parameters of RBFs with variable widths that provides the best possible function approximation.

The rest of the paper is structured as follows: the formulation of an optimization problem is explained briefly in Sect. II, the optimization systems of RBF networks, using DE, are described in Sect. III, and experimental results are shown in Sect. IV. The conclusion is given in Sect. V.

II. FORMULATION OF AN OPTIMIZATION PROBLEM

A. The Nonlinear Regression Model

A nonlinear regression model can be written as [28]

$$y_n = g(\mathbf{v}_n, \mathbf{x}) + z_n, \tag{1}$$

where $\mathbf{v}_n$ is a vector of associated regressor variables or independent variables for the nth case, g is the expectation function and at least one of the derivatives of the expectation function with respect to the parameters depends on at least one of the parameters. $\mathbf{x}$ is a vector of the parameters in the nonlinear model. y_n is a random variable representing the response for case $n, n = 1, 2, ..., N$ and has a deterministic part and a stochastic part. The deterministic part, $g(\mathbf{v}_n, \mathbf{x})$, depends upon the parameters $\mathbf{x}$ and the predictor $\mathbf{v}_n \in \Re^d$, where d is the dimensionality of the function g. The stochastic part, represented by the random variable, z_n, is a disturbance that perturbs the response for that case.

When analyzing a particular set of data, the vectors $\mathbf{v}_n$, $n = 1, 2, ..., N$, are considered as fixed and the intention is on the dependence of the expected response on $\mathbf{x}$. An N-dimensional vector $\boldsymbol{\eta}(\mathbf{x})$, having the nth element, is created,

$$\eta_n(\mathbf{x}) = g(\mathbf{v}_n, \mathbf{x}), \qquad n = 1, 2, ..., N, \tag{2}$$

and write the nonlinear regression model for N cases as

$$\mathbf{y} = \boldsymbol{\eta}(\mathbf{x}) + \mathbf{z}, \tag{3}$$

where $\mathbf{y}$ is the vector of random variables representing the data that may be given and $\mathbf{z}$ is the vector of random variables representing the disturbances, assumed to have a spherical normal distribution. That is, $E[\mathbf{z}] = \mathbf{0}, Var(\mathbf{z}) = E[\mathbf{z}\mathbf{z}^T] = \sigma^2\mathbf{I}$, where $\mathbf{I}$ is an $N \times N$ identity matrix. The deterministic part, $\boldsymbol{\eta}(\mathbf{x})$, a function of the parameters and regressor variables, gives the mathematical model for the responses.

B. Radial Basis Functions Network

RBFs have their origin in the solution of the multivariate interpolation problem [17]. Arbitrary function $g(\mathbf{v})$: $\Re^d \rightarrow \Re$ can be approximated by mapping, using a RBF network with a single hidden layer of p units:

$$\begin{aligned} \widehat{g}(\mathbf{v}, \mathbf{x}) &= w_0 + \sum_{j=1}^{p} w_j r_j(\mathbf{v}, \boldsymbol{\sigma}_j, \mathbf{c}_j) \\ &= w_0 + \sum_{j=1}^{p} w_j \phi_j(\boldsymbol{\sigma}_j, \|\mathbf{v} - \mathbf{c}_j\|), \end{aligned} \tag{4}$$

where $\mathbf{v} \in \Re^d$; $\mathbf{x}$ is the vector of variable factors including w_0, w_j, $\boldsymbol{\sigma}_j$, and $\mathbf{c}_j$; p denotes the number of basis functions; $\mathbf{w} = (w_1, w_2, ..., w_p)^T$ contains the weight coefficients; w_0 is the bias; $r_j(\cdot)$ represents the d-dimensional activation function (also known as the radial basis function) from $\Re^d$ to $\Re$; $\|\cdot\|$ is the Euclidean norm; $\mathbf{c}_j = (c_{j1}, c_{j2}, ..., c_{jd})^T$, $j = 1, 2, ..., p$, are the centres of the basis functions; $\boldsymbol{\sigma}_j = (\sigma_{j1}, \sigma_{j2}, ..., \sigma_{jd})^T$, $j = 1, 2, ..., p$, are the widths, which are called scaling factors for the radii $\|\mathbf{v} - \mathbf{c}_j\|$, $j = 1, 2, ..., p$, of the basis functions, respectively; and $\phi(\cdot)$ is a non-linear function that monotonically decreases (or increases) as $\mathbf{v}$ moves away from $\mathbf{c}_j$ and can be one of common RBFs:

$$\begin{aligned} \phi(r) &= r \quad \text{(Linear)}, \\ \phi(r) &= r^3 \quad \text{(Cubic)}, \\ \phi(r) &= r^2 log(r) \quad \text{(Thin plate spline)}, \\ \phi(r) &= e^{-(r/c)^2} \quad \text{(Gaussian)}, \\ \phi(r) &= (r^2 + c^2)^{-\alpha} \quad \text{(Inverse multiquadric)}, \\ \phi(r) &= (r^2 + c^2)^{\beta} \quad \text{(Multiquadric)}, \end{aligned}$$

where $r \geq 0$, c is a positive constant, $\alpha > 0$, and $0 < \beta < 1$. The Gaussian function was chosen from the above listed radial basis functions. In order to simplify the notation, coordinate axes-aligned Gaussian RBF functions are used:

$$r_j(\mathbf{v}) = e^{-\frac{\|\mathbf{v} - \mathbf{c}_j\|^2}{2\boldsymbol{\sigma}_j^2}}. \tag{5}$$

A multidimensional Gaussian RBF function can be represented as the product of lower dimensional Gaussian RBF functions. When a 2-dimensional Gaussian-type RBF is centred at the centroids $\mathbf{c}_j$, it follows from (4) that

$$\begin{aligned} \widehat{g}(\mathbf{v}, \mathbf{x}) &= w_0 + \sum_{j=1}^{p} w_j e^{-\frac{\|\mathbf{v} - \mathbf{c}_j\|^2}{2\boldsymbol{\sigma}_j^2}} \\ &= w_0 + \sum_{j=1}^{p} w_j e^{-\frac{(v_1 - c_{j1})^2}{2\sigma_{j1}^2}} e^{-\frac{(v_2 - c_{j2})^2}{2\sigma_{j2}^2}}, \end{aligned} \tag{6}$$

where $\mathbf{x}$ is the vector of variable factors and can be written as

$$\begin{aligned} \mathbf{x}^T &= (w_0, \mathbf{w}^T, \boldsymbol{\sigma}_1^T, \mathbf{c}_1{}^T, ..., \boldsymbol{\sigma}_j^T, \mathbf{c}_j{}^T, ..., \boldsymbol{\sigma}_p^T, \mathbf{c}_p{}^T)^T \\ &= (w_0, w_1, ..., w_p, \sigma_{11}, \sigma_{12}, c_{11}, c_{12}, ..., \\ &\quad \sigma_{j1}, \sigma_{j2}, c_{j1}, c_{j2}, ..., \sigma_{p1}, \sigma_{p2}, c_{p1}, c_{p2})^T. \end{aligned} \tag{7}$$

The network can be trained to approximate function $g(\mathbf{v})$ by finding the optimal vector $\mathbf{x}$ given a (possibly noisy) training set, $V = \{(\mathbf{v}_n, y_n) | n = \{1, 2, ..., N\}, \mathbf{v}_n \in \Re^d, y_n \in \Re\}$.

C. Cost Function

Given the number of radial basis functions and a training set, the network parameters are found such that they minimize the Euclidean distance between the desired and actual outputs, i.e., the objective function:

$$f(\mathbf{x}) = \sqrt{\sum_{n=1}^{N} \Big(y_n - \widehat{g}(\mathbf{v}_n, \mathbf{x}) \Big)^2}. \tag{8}$$

In order to make comparisons with the former algorithm by Esposito et al. [4], the Mean Square Error (MSE) will also be considered:

$$f(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N}\left(y_n - \widehat{g}(\mathbf{v}_n, \mathbf{x})\right)^2. \quad (9)$$

D. Brief Description of the Differential Evolution Algorithm

The optimization target function is of the form

$$f(\mathbf{x}) : \Re^D \to \Re. \quad (10)$$

The optimization objective is to minimize the value of the target function by finding the optimal values of its parameters, $\mathbf{x} = (x_1, x_2, ..., x_D)^T$, where $\mathbf{x}$ denotes a vector composed of D objective function parameters. Usually the parameters of the target function are also subject to the lower and upper boundary constraints $\mathbf{x}^{(L)}$ and $\mathbf{x}^{(U)}$:

$$x_k^{(L)} \le x_k \le x_k^{(U)}, k = 1, 2, ..., D. \quad (11)$$

DE is a parallel direct search method. It utilizes N_{pop} D-dimensional parameter vectors, $\mathbf{x}_{i,G}, i = 1, 2, ..., N_{pop}$, as a population for generation G in order to minimize the target function by finding an optimal objective function parameter vector, where N_{pop} is the number of population members.

The initial population is chosen randomly to cover the entire parameter space uniformly (otherwise stated). The crucial idea behind DE is a scheme for generating trial parameter vectors. DE generates a new noisy parameter vector by adding the difference weighted by mutation amplification F between two (or more) population vectors to the other vector subject to perturbation during the mutation operation. In crossover operation, each parameter of a trial vector may be chosen as that of the former generated noisy vector or that of the target vector (i.e., the predetermined population member), based on the comparison result between crossover operator C_r and a uniformly distributed random number in $[0, 1]$ generated anew for each parameter, except that one parameter will always come from the noisy vector in order to ensure that the trial vector differs from the target vector by at least one parameter which index can be D (the last parameter) for all the trial vectors [27] or chosen anew from $[1, D]$ for each trial vector. After crossover, if the resulting trial vector yields a lower or equal objective function value than the predetermined population vector, the newly generated trial vector will replace the predetermined population vector in the following generation's population; otherwise, the predetermined population vector is retained [29].

III. OPTIMIZATION OF RADIAL BASIS FUNCTION NETWORKS BY DIFFERENTIAL EVOLUTION

A. Representations of the Parameters of the Objective Function

1) Approach 1: When function $g(\mathbf{v})$ is computed using a set of d-dimensional Gaussian RBF functions as in (6) and the optimization objective function is stated as in (8), the optimization process using DE will minimize $f(\mathbf{x})$ by finding an optimal parameter vector $\mathbf{x}$ [8]:

$$\begin{aligned}\mathbf{x}^T = \; & (w_0, w_1, ..., w_j, ..., w_p \\ & \sigma_{11}, \sigma_{12}, ..., \sigma_{1d}, c_{11}, c_{12}, ..., c_{1d}, ..., \\ & \sigma_{j1}, \sigma_{j2}, ..., \sigma_{jd}, c_{j1}, c_{j2}, ..., c_{jd}, ..., \\ & \sigma_{p1}, \sigma_{p2}, ..., \sigma_{pd}, c_{p1}, c_{p2}, ..., c_{pd})^T. \end{aligned} \quad (12)$$

The dimensionality of the parameter vector of the objective function, D, is $(d \cdot 2 + 1)p + 1$. When $d = 1$, $D = 3p + 1$, it follows that

$$\begin{aligned}\mathbf{x}^T = \; & (w_0, w_1, ..., w_j, ..., w_p, \\ & \sigma_{11}, c_{11}, ..., \sigma_{j1}, c_{j1}, ..., \sigma_{p1}, c_{p1})^T. \end{aligned} \quad (13)$$

Especially when $d = 1$, $p = 1$, $w_0 = 0$, $w_1 = 1$, $c_1 = 0.5$, $\sigma_1 = 0.1$, (6) becomes $\widehat{g}(v, \mathbf{x}) = e^{-\frac{(v-0.5)^2}{2\times 0.1^2}}$, where $\mathbf{x^T} = (0 \quad 1 \quad 0.5 \quad 0.1)^T$.

2) Approach 2: When the number of units, p, is not known properly before hand, the minimum sized RBF networks can be built to approximate functions with a dimension changeable parameter vector $\mathbf{x}$:

$$\begin{aligned}\mathbf{x}^T = \; & (w_0, w_1, ..., w_j, ..., w_p, \\ & \sigma_1, ..., \sigma_j, ..., \sigma_p)^T, \end{aligned} \quad (14)$$

using a DE-based heuristic explained as the following:

a) The p initial centres, called set C, are selected randomly from training set V. Those in V that have not been sampled constitute the set V_e.
b) Dimensionality $D = 2 \cdot p + q$, where q is the number of new centre candidates added in each adjusting operation.
c) Evolution: adapt the widths and the weights of RBFs, using DE.
d) After a certain number of generations, say G_e, then adjust by:
   i) Deletion: delete w_is, if w_i is small, i.e., if $|w_i/\bar{w}| < \epsilon_w$, where ϵ_w is a threshold; set C is renewed by deleting the related centres $c_{ij}, j = 1, 2, ..., d$. p is renewed by subtracting the number of deleted weights.
   ii) Addition: if V_e is not empty, take randomly q centre candidates from V_e, add to C, renew p by adding q, go to Step 2c.
e) Terminate the process if none of the basis functions has been removed during Step 2(d)i and the change

TABLE I
CONTROL PARAMETERS' SETTING

Variable	Approach 1	Approach 2	Approach 3
N_{pop}	$p \cdot 3 \cdot 10$	100	100
C_r	0.9	0.9	0.9
F	0.9	0.9	0.9

Note: p denotes the number of Gaussian RBF functions.

of weights between two deletion operations is small, in addition to that of no element being left in V_e. Otherwise go to Step 2c.

3) Approach 3: Combine Approach 1 and Approach 2 stated above and get a DE-based heuristic to approximate functions with growing RBF networks. The objective function is defined as in (9) to minimize the Mean Square Error. The entire samples in training set V are used for training and $\mathbf{v}_n \in \Re^d, n = 1, 2, ..., N$, are possible centre candidates.
   a) The sample pair $(\mathbf{v}_i, y_i)$ from set V, for which the function approximation is the worst, having the biggest difference between the desired and the actual outputs of the existing RBF network, is chosen to be taken as a new node centre, i.e., $|y_i - \hat{g}_{p-1}(\mathbf{v}_i)| = \max\{|y_n - \hat{g}_{p-1}(\mathbf{v}_n)|, (\mathbf{v}_n, y_n) \in V, n = 1, 2, ..., N\}$, where $\hat{g}_{p-1}$ is the existing Gaussian RBF network.
   b) Growing: (i) p is increased by 1; (ii) the chosen node $\mathbf{v}_i$ is added to centres' set $C = \{c_{jk} \mid j = 1, 2, ..., p, k = 1, 2, ..., d\}$, where p is the number of hidden nodes present in the RBF network; and (iii) the weight of this node is set as $w_p = y_i - \hat{g}_{p-1}(\mathbf{v}_i)$, i.e., the error between the desired output and the existing RBF network output (i.e., before adding the new node).
   c) Local tuning: find the width of the newly added node after G_{local} generations, using DE.
   d) Global tuning: the widths of the present nodes of the entire RBF network are learned after G_{global} generations, using DE with dimensionality D as p, as the parameter vector $\mathbf{x}$ of the objective function can be written as
   $$\mathbf{x}^T = (\boldsymbol{\sigma}_1^T, ..., \boldsymbol{\sigma}_j^T, ..., \boldsymbol{\sigma}_p^T)^T. \quad (15)$$
   where in $\boldsymbol{\sigma}_j$, $\sigma_{j1} = \sigma_{j2} = ... = \sigma_{jd}$, $j = 1, 2, ..., p$, i.e., the RBFs have circular cross sections.
   e) Criterion: if $f(x) \geq \epsilon_f$ and $p < p_{max}$, where ϵ_f is a threshold and p_{max} is the maximum number of nodes of the RBF network, go to Step 3a. Otherwise, stop the process.
   f) Testing the performance of the DE-based approach: for each function, testing set V_t, which is different from training set V, contains the same number of samples as in V and the samples are uniformly distributed random values in the function domain.

B. Control Parameters' Setting of DE

The strategy used is DE/rand/1/bin.

Realizing the above proposed approaches, the parameters of the objective function, stated in (8) or (9), usually correspond to the network architecture. The evolutionary process starts with a population of the above mentioned parameters, randomly generated, based on the defined region and the boundaries of the discussed function, $g(\mathbf{v})$, i.e., $\mathbf{c}_j \in [-0.25 + \mathbf{v}^{(L)}, 0.25 + \mathbf{v}^{(U)}]$, $\boldsymbol{\sigma}_j \in (0, (\mathbf{v}^{(U)} - \mathbf{v}^{(L)})/2]$, and $w_j \in [-2 \cdot |g|_{max}, +2 \cdot |g|_{max}]$, where $\mathbf{v}^{(L)}$ and $\mathbf{v}^{(U)}$ are the vectors of the lower limits and the upper limits of $\mathbf{v}$ respectively, and $|g|_{max}$ is the maximum of the absolute function values of the computed function $g(\mathbf{v})$. For every member of this initial population, the objective function value, calculated according to (8) or (9), is evaluated, based on the given data set.

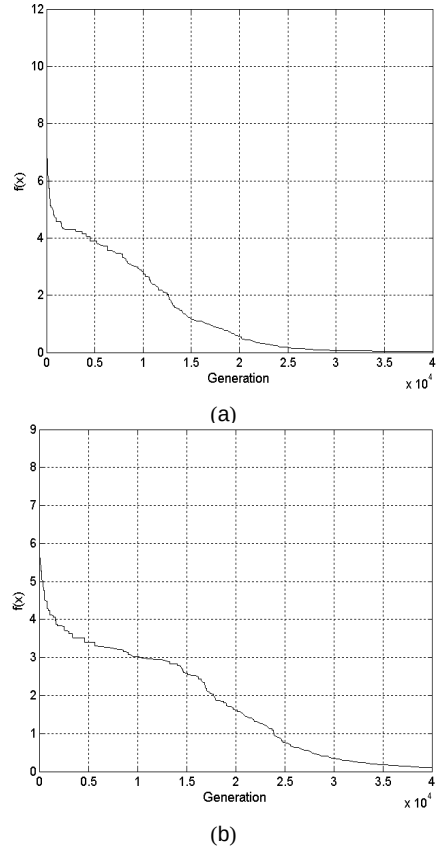

Fig. 1. The objective function value history during the optimization process ($p = 5$): the average of 10 independent runs for functions. (a) A Hermite polynomial; (b) 1D sine wave

The control parameters' setting affects the performance of the proposed approaches, and its values, in the case of Approach 1, were chosen based on discussions in [29], [30], [31] and are given in column 2 of Table I. In the case of Approach 2 and Approach 3, the control parameters were different and are shown in columns 3 and 4 of Table I.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

A. Experiments with Functions

In this section, the proposed approaches are applied to single-input-single-output and two-input-single-output test functions without any disturbance [8]:

1) The first benchmark problem is a Hermite polynomial approximation problem, given by
$$g_1(v) = 1.1 \cdot (1 - v - 2v^2) \cdot e^{-\frac{v^2}{2}}, v \in [-4, +4], \quad (16)$$
as shown in Fig. 2a.

TABLE II

EXPERIMENTAL RESULTS WITH TEST FUNCTIONS IN (16) AND (17)

Function	Approach 1				Approach 2					Approach 3				
	$f(\mathbf{x})$ in (8)	N	Generations	NF (10^4)	$f(\mathbf{x})$ in (8)	N	p	Generations	NF (10^4)	$f(\mathbf{x})$ in (8)*	$f(\mathbf{x})$ in (9) (10^{-5})	N	p	NF (10^4)
g_1	0.0384	201	$4 \cdot 10^4$	600	0.3337	41	4	5,200	52	0.0973	9.4636	100	11	7.7
g_2	0.0969	201	$4 \cdot 10^4$	600	1.1437	41	3	5,200	52	0.1262	15.935	100	12	8.4

Note: NF stands for the number of the objective function evaluations. (*): the Euclidean distance transformed from the MSE.

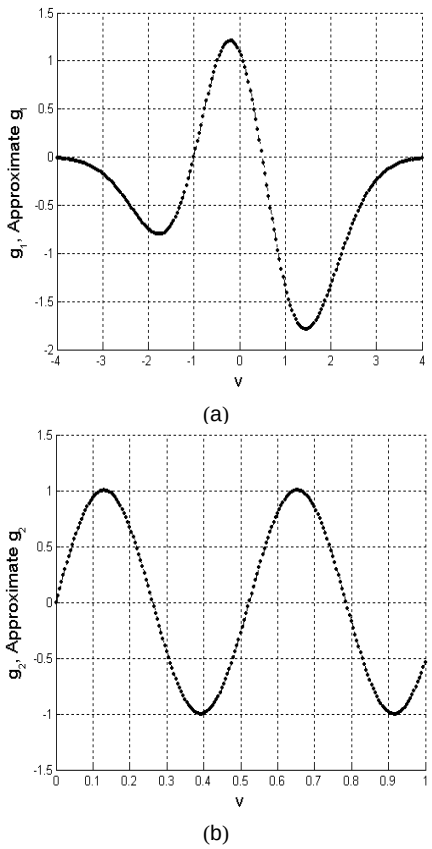

Fig. 2. Net functions and their approximations by 5-centroid RBF networks. (a) A Hermite polynomial; (b) 1D sine wave (figure legends: '-'—net function; "·"—approximation)

2) The second problem is a 1-D sine wave function:

$$g_2(v) = \sin(12v), v \in [0, 1], \tag{17}$$

as shown in Fig. 2b.

3) The third function is given by:

$$g_3(v) = \sum_{n=1}^{4} \left(n \cdot e^{-n^2 \cdot (v-4+2n)^2}\right), v \in [-4, +4], \tag{18}$$

4) The fourth function is:

$$g_4(v) = v, v \in [0, +1], \tag{19}$$

5) The fifth function is:

$$g_5(\mathbf{v}) = v_1^2 + v_2^2, (v_1, v_2) \in [-1, +1], \tag{20}$$

6) The sixth function is:

$$g_6(\mathbf{v}) = 2 - (v_1^2 + v_2^2), (v_1, v_2) \in [-1, +1], \tag{21}$$

For continuous function $g(\mathbf{v})$, the input is considered as data set V, containing N data points in the function domain:

$$V = \left\{(\mathbf{v}_n, y_n) \in \Re^d \times \Re, 1 \leq n \leq N : y_n = g(\mathbf{v}_n)\right\},$$

where d denotes the dimension of the discussed function $g(\mathbf{v})$, (i.e., $d = 1$ for single-input functions and so on) and $\mathbf{v}_n$, $n = 1, 2, ..., N$, are N uniformly spaced noiseless points in the defined region of the computed function except that they are uniformly distributed random values in Approach 3.

The functions in (16) – (21) were used, realizing the approaches stated in Sect. III-A. (i) Approach 1: $p = 5$, DE ran with the control parameter' setting from column 1 of Table I and the experiment for each function was repeated 10 times; (ii) Approach 2: p is variable and initially set as 5 while $q = 1$; and (iii) Approach 3: p is initially set as zero. Key experimental results are:

1) a) Fig. 1 shows the optimization processes realizing Approach 1, i.e., the average of objective function values stated in (8) based on 10 independent runs. After $40,000$ generations, the objective function value decreases from 10.0053 to 0.0384 for g_1 and from 8.2830 to 0.0969 for g_2, the approximation error is under $1.5 \cdot 10^{-3}$ and $4.5 \cdot 10^{-3}$ respectively, and the standard deviations are $3.3666 \cdot 10^{-4}$ and $7.5326 \cdot 10^{-4}$ individually. Fig. 2 shows the net functions and their RBF network outputs.

   b) Figs. 1 and 2 reveal that the optimization process converges effectively after a certain number of generations for each of the tested functions. It also shows that the less the complexity of the function, the faster the optimization process (i.e., g_2 is slightly more complex than g_1, concerning the number of turns in the function domain).

2) a) Fig. 3 shows the implementation using Approach 2 with training set V of the size of 41. Fig. 3a-b shows how the objective function value changes when the number of the centres of RBF networks changes during the optimization process, and Fig. 3c shows the net function, the final network output as well as its composing centres, Gaussian RBFs, and the bias when approximating g_1; and Fig. 3d-f

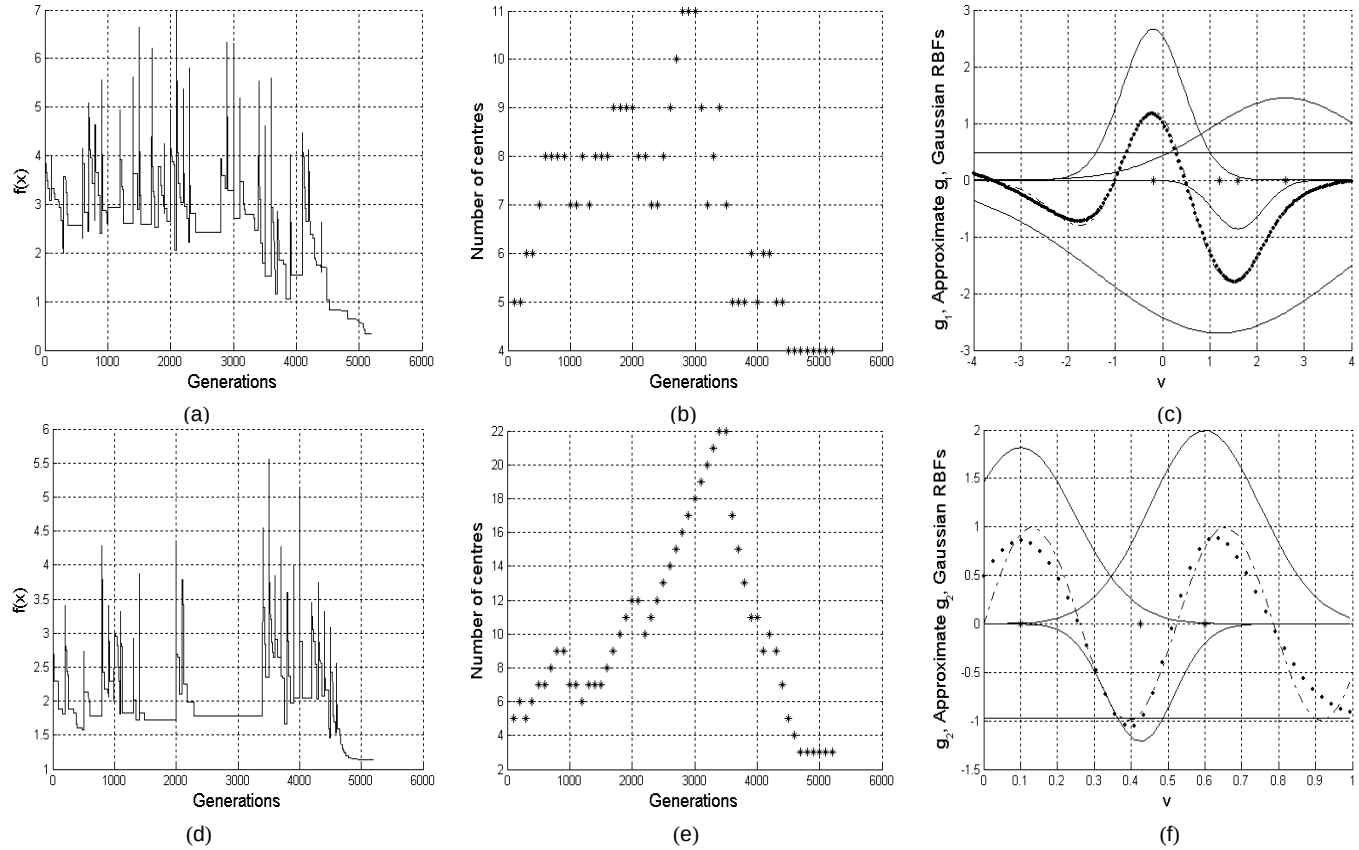

Fig. 3. Building minimum-sized RBF networks when p is variable. (a) The objective function value history when approximation g; (b) The number of centres; (c) A Hermite polynomial and its approximation; (d) The objective function value history when approximation g; (e) The number of centres; (f) 1D sine wave and its approximation (figure legends in c and f: "*"—centres; "-"— Gaussian RBFs and the bias; "·"—approximation; "– -"—net function)

gives the similar information when using Gaussian RBF networks to approximate g_2.

b) From Fig. 3, it can be seen that 4-centroid and 3-centroid RBF networks are finally chosen for optimally approximating g_1 and g_2, respectively. The method can solve the problem of how many hidden units should be used when building RBF networks to approximate functions.

3) a) The experimental results of realizing Approach 1 and Approach 2 reveal that after 5,200 generations, the average distance $f(\mathbf{x})/N$, which is obtained from the objective function value (the Euclidean distance) divided by the size of the training set in order to get rid of the influence from the size of the training set, is 0.0190 and 0.0081 for g_1 respectively, and 0.0170 and 0.0279 for g_2 as well.

b) The experimental results of realizing Approach 1 and Approach 2 reveal that after $520,000$ function evaluations, the average distance $f(\mathbf{x})/N$ is 0.0210 and 0.0081 for g_1 respectively, and 0.0174 and 0.0279 for g_2 as well.

c) Approach 2 gave worse results in the case of g_2 in the above two comparisons, because g_2 has more turns in the function domain than g_1 (three turns over eight for g_1 and four turns over one for g_2) and the training set of g_2 contains less information when the size of training set is the same.

4) a) Using Approach 3 ($G_{local} = 20, G_{global} = 50$), the objective function value stated in (9) is $9.4636 \cdot 10^{-5}$ with an 11-RBF network for approximating g_1 and $1.5935 \cdot 10^{-4}$ with a 12-RBF network for approximating g_2, as shown in Table II.

b) After $77,000$ objective function evaluations, the best objective function value in (8) was 5.0674 using Approach 1 and 0.0973 using Approach 3 for g_1, 4.5245 using Approach 1 and 0.1262 using Approach 3 for g_2.

c) As shown in Table II: (*i*) for g_1, the optimization process got the best objective function value in (8) 0.0973 after $77,000$ function evaluations using Approach 3, and 0.3337 after $520,000$ function evaluations using Approach 2; and (*ii*) for g_2, the optimization process got the best objective function value in (8) 0.1262 after $84,000$ function evaluations using Approach 3 and 1.1437 after $520,000$ function evaluations using Approach 2.

d) Seen from the above two comparisons, Approach 3 worked most efficiently among these three.

TABLE III

EXPERIMENTAL RESULTS WITH TEST FUNCTIONS IN (18)–(21)

Function	Method	Training error	Testing error	Maximum error	N	p
g_3	DE	120	130	0.0270	400	6
	EA	22	24		400	6
g_4	DE	1.0986	1.1035	0.0138	1000	5
	EA		1.4		1000	6
g_5	DE	940	970	0.3475	900	10
g_6	DE	98.724	150	0.3419	900	11

Note: Training error and testing error are of the order of 10^{-5}.

e) Table III shows more results of using Approach 3 (short as DE), including comparisons with the method proposed by Esposito et al [4]—an incremental algorithm for growing RBF networks by means of an evolutionary strategy and showed their strategy can achieve better performance than a greedy algorithm [32] and Wavelet Neural Network [33] both in terms of net size and in terms of computation time, which is short as EA: (*i*) the method by Esposito et al worked better in g_3 with testing error having the value of $2.4 \cdot 10^{-4}$ than the DE-based approach having the testing error of $1.3 \cdot 10^{-3}$ when the number of nodes is 6, but worse in g_4 with testing error of the value of $1.4 \cdot 10^{-5}$ with six RBFs than the DE-based approach having the testing error of $1.1.0409 \cdot 10^{-5}$ with five RBFs; and (*ii*) the DE-based approach obtained an MSE less than $9.7 \cdot 10^{-3}$ (on testing set) for test function g_5 with 10 nodes and an MSE less than $1.5 \cdot 10^{-3}$ (on testing set) for test function g_6 with 11 nodes.

B. Experiments with Real-Life Data

The proposed Approach 1 ($p = 4$) and Approach 3 were also applied to a set of 1-dimensional nuclear magnetic resonance (1D ^{13}C NMR) spectra of oil samples, where the control parameters are taken from Table I. Key experimental results are:

1) a) Fig. 4 shows the spectra measured, which is complicated, and reconstructed by the RBF networks with Approach 1. Fig. 4a shows the measured spectra of the 18 oil samples transformed into a data format that can be used in Matlab. $N = 101$ out of 4501 spectral values are selected for each sample. Fig. 4b shows the approximated spectra of the 18 samples found by the RBF networks for each sample respectively. Fig. 5 shows the optimization processes finding the parameters of the RBF networks.

b) The results show that the suggested method is feasible for finding particular RBF parameters for each spectrum so that each sample can be seen to be significantly different from each other and can be classified correctly. The results illustrate some of the advantages of training RBF networks using DE in the analysis of complex spectroscopic data, i.e., RBF networks training by DE-based methods will automatically focus on the most relevant substructures of the compounds.

2) a) Table IV shows the numerical results obtained, using Approach 3. For each sample, training set V contains every second point of the full spectrum starting from the beginning, and the rest is put in testing set V_t.

b) Working samples ($V_t^{(i)}$, $V_f^{(i)}$,$i = 1, 2, ..., 18$, where $V_t^{(i)}$ and $V_f^{(i)}$ are the test set and full spectrum of the ith sample) were input to the respective RBF networks learned to see if the working samples will be classified properly. The process can be explained in the following: given the ith working sample's spectrum, get its outputs from each of the learned RBF networks, find the difference between them, i.e., the Mean Square Error MSE_{ij}; this working sample is classified into the class *c*, which has the smallest difference, i.e., $\mathrm{MSE}_{ic} = \mathrm{minimum}\{\mathrm{MSE}_{ij}, j = 1, 2, ..., 18\}$. The experimental results and classification results are shown in Table V and Table VI.

c) From Table V and Table VI, it can be seen that most of the samples were classified correctly except that: (i) $V_t^{(8)}$ was classified as class 17 since $\mathrm{MSE}_{8,17}$ is slightly smaller than $\mathrm{MSE}_{8,8}$ and $V_f^{(8)}$ could be put into class 8, 15 or 17 because $\mathrm{MSE}_{8,8} = \mathrm{MSE}_{8,15} = \mathrm{MSE}_{8,17}$; ($ii$) $V_t^{(14)}$ was classified as class 2 since $\mathrm{MSE}_{14,2}$ is slightly smaller than $\mathrm{MSE}_{14,14}$ and $V_f^{(14)}$ could be put into class 2 or 14 because $\mathrm{MSE}_{14,2} = \mathrm{MSE}_{14,14}$; ($iii$) the 16th sample was classified as class 10 since $\mathrm{MSE}_{16,10}$ is slightly smaller than $\mathrm{MSE}_{16,16}$; (iv) $V_t^{(18)}$ could be classified as class 9 or 18 since their differences are equal and $V_f^{(18)}$ can be put into class 9 or 18 because of the same reason. The classification percentage is 77.78.

V. CONCLUSION

The Differential Evolution algorithm and two DE-based methods were applied to training Radial Basis Function networks with variable widths for approximating functions and for reconstructing data, which represents particular features of a set of oil samples. The choice of the optimal network parameters corresponds to the minimum Euclidean distance or the Mean Square Error between the desired and actual outputs. The tests based on a set of test functions and real-life data demonstrated the approaches. The obtained results suggest that the original DE and the DE-based approaches are effective in optimizing the parameters of Gaussian-type RBF networks and the DE-based growing approach is the most efficient among

TABLE IV

EXPERIMENTAL RESULTS OF 1-DIMENSIONAL NUCLEAR MAGNETIC RESONANCE SPECTRA

Sample	Training error	Testing error	Maximum error	p	Size of V	Size of V_t
s_1	$3.3877 \cdot 10^{-4}$	$8.2423 \cdot 10^{-4}$	0.6252	10	2251	2250
s_2	$1.8610 \cdot 10^{-4}$	$2.9071 \cdot 10^{-4}$	0.3959	10	2251	2250
s_3	$1.3 \cdot 10^{-3}$	$1.4 \cdot 10^{-3}$	0.4637	10	2251	2250
s_4	$1.2 \cdot 10^{-3}$	$1.6 \cdot 10^{-3}$	0.4664	10	2251	2250
s_5	$3.5107 \cdot 10^{-4}$	$5.0843 \cdot 10^{-4}$	0.3303	10	2251	2250
s_6	$1.8 \cdot 10^{-3}$	$1.9 \cdot 10^{-3}$	0.3749	15	2251	2250
s_7	$1.1 \cdot 10^{-3}$	$1.3 \cdot 10^{-3}$	0.4319	15	2251	2250
s_8	$6.3609 \cdot 10^{-4}$	$7.9752 \cdot 10^{-4}$	0.3964	15	2251	2250
s_9	$1.3396 \cdot 10^{-4}$	$1.9786 \cdot 10^{-4}$	0.2325	10	2251	2250
s_{10}	$8.9967 \cdot 10^{-5}$	$2.5386 \cdot 10^{-4}$	0.4483	10	2251	2250
s_{11}	$8.2 \cdot 10^{-3}$	$8.5 \cdot 10^{-3}$	0.8376	10	2251	2250
s_{12}	$8.7 \cdot 10^{-3}$	$9.3 \cdot 10^{-3}$	0.6655	10	2251	2250
s_{13}	$2.4224 \cdot 10^{-4}$	$4.0089 \cdot 10^{-4}$	0.4354	10	2251	2250
s_{14}	$1.0848 \cdot 10^{-4}$	$4.0359 \cdot 10^{-4}$	0.5609	10	2251	2250
s_{15}	$6.8524 \cdot 10^{-4}$	$7.7366 \cdot 10^{-4}$	0.2739	9	2251	2250
s_{16}	$3.9652 \cdot 10^{-5}$	$1.1833 \cdot 10^{-4}$	0.3686	1	2251	2250
s_{17}	$1.6997 \cdot 10^{-4}$	$2.4077 \cdot 10^{-4}$	0.2287	10	2251	2250
s_{18}	$5.7802 \cdot 10^{-4}$	$7.0871 \cdot 10^{-4}$	0.3583	10	2251	2250

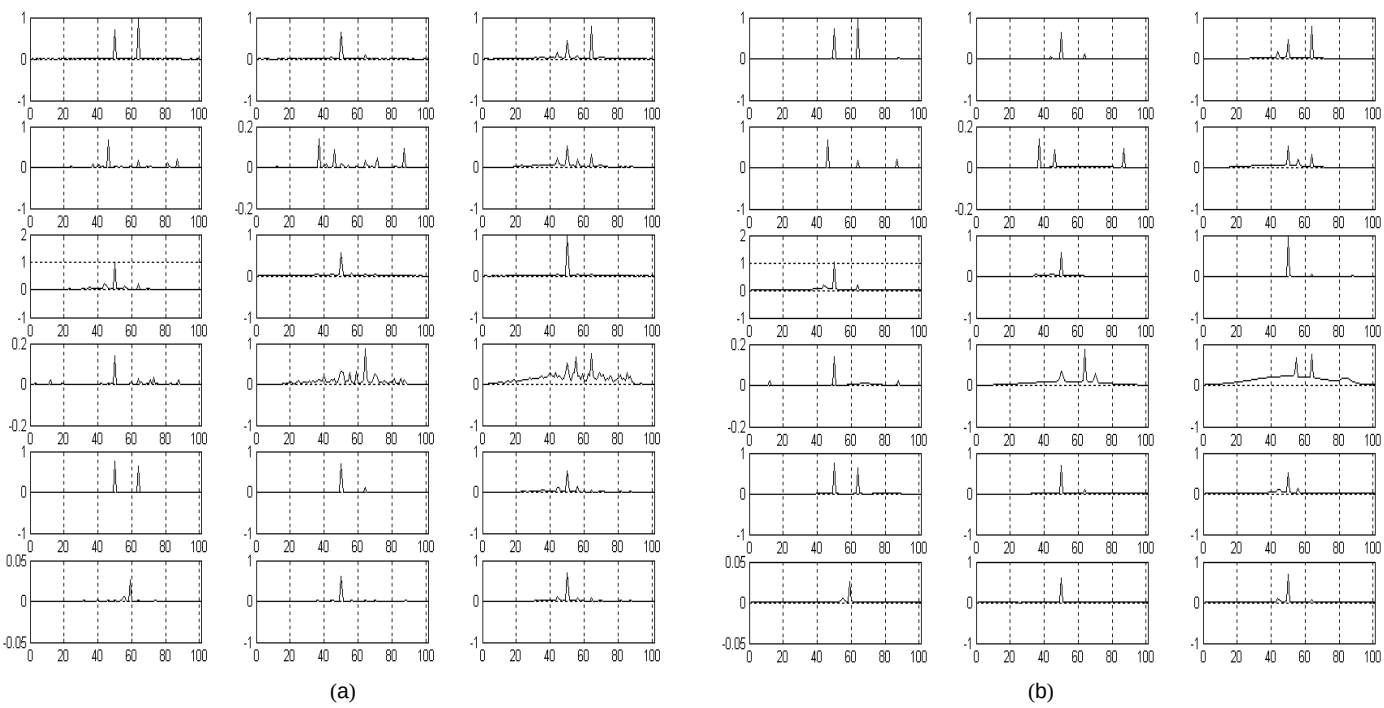

Fig. 4. Spectra of the 18 samples: the samples are indexed from left to right top to bottom. (a) Measured spectra of the 18 oil samples: 101 out of 4501 spectral values are selected for each sample; (b) Reconstructed spectra of the 18 samples, each of which is approximated by a 4-centroid RBF network

these three. The comparison to another incremental algorithm reported in the literature has showed the DE-based growing approach performs well in terms of the net size.

Future research should include tests with higher data dimensionalities and with less smooth data as well as possible ways of improving the classification percentage and the efficiency of the Differential Evolution based training process.

ACKNOWLEDGEMENTS

This work was partial supported by the Fortum Foundation of the Fortum Company, Finland. The authors would like to thank Mika Ala-Korpela and Igor Monastyrnyi for their assistance with the spectral data.

TABLE V

THE DIFFERENCE BETWEEN THE REAL SPECTRA AND THE ESTIMATED FROM RBF NETWORKS

Working sample	the Mean Square Error (10^{-4})																	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$V_t^{(1)}$	**8**	17	19	51	37	44	40	37	28	36	122	431	14	18	36	37	25	29
$V_f^{(1)}$	**6**	18	20	53	39	46	39	37	28	38	119	425	14	19	36	38	26	30
$V_t^{(2)}$	23	**3**	22	41	27	36	22	18	8	22	122	433	21	12	18	24	7	11
$V_f^{(2)}$	21	**2**	21	41	27	36	20	17	8	23	117	426	22	11	17	24	9	12
$V_t^{(3)}$	35	29	**14**	55	44	32	29	30	35	38	96	350	32	32	33	32	30	37
$V_f^{(3)}$	34	30	**14**	55	46	32	27	29	36	40	91	344	34	31	33	32	31	38
$V_t^{(4)}$	64	49	56	**16**	19	74	65	49	49	44	127	424	54	51	55	47	47	50
$V_f^{(4)}$	60	47	55	**14**	19	71	62	46	47	43	121	416	54	48	52	45	45	48
$V_t^{(5)}$	54	34	45	19	**5**	69	55	37	34	26	133	445	47	40	43	34	31	36
$V_f^{(5)}$	50	34	44	20	**4**	67	53	36	33	27	127	437	48	37	40	33	30	35
$V_t^{(6)}$	53	33	32	66	56	**19**	26	31	34	50	101	301	50	43	27	30	33	33
$V_f^{(6)}$	50	32	32	66	57	**19**	25	31	36	52	99	295	53	42	27	31	35	35
$V_t^{(7)}$	50	25	32	67	55	28	**13**	20	23	45	102	326	43	33	19	31	22	25
$V_f^{(7)}$	48	25	32	67	56	27	**12**	20	25	47	100	320	45	33	19	33	24	27
$V_t^{(8)}$	45	17	31	45	31	42	16	8	9	22	113	391	36	26	8	20	**6**	11
$V_f^{(8)}$	41	16	30	44	31	40	15	**8**	10	23	110	385	38	24	**8**	21	**8**	13
$V_t^{(9)}$	33	08	31	43	29	41	19	13	**3**	24	122	428	28	20	11	23	5	4
$V_f^{(9)}$	31	8	30	43	30	40	17	13	**3**	25	119	422	31	18	11	23	6	5
$V_t^{(10)}$	51	26	42	37	23	68	45	32	26	**4**	132	451	43	37	35	29	22	27
$V_f^{(10)}$	46	25	40	37	23	65	43	30	24	**4**	126	443	43	32	32	27	21	26
$V_t^{(11)}$	133	120	102	135	132	100	101	110	125	123	**84**	216	128	129	102	100	116	123
$V_f^{(11)}$	130	120	103	135	132	100	100	110	127	126	**81**	211	130	128	102	101	118	124
$V_t^{(12)}$	314	300	281	315	312	244	271	284	304	309	239	**95**	308	308	258	258	297	301
$V_f^{(12)}$	311	300	281	315	312	243	271	285	306	312	238	**91**	311	308	259	261	299	303
$V_t^{(13)}$	13	16	24	46	34	51	42	38	28	36	130	450	**6**	11	39	41	24	31
$V_f^{(13)}$	12	16	24	47	35	51	41	37	28	37	126	443	**6**	12	39	41	26	32
$V_t^{(14)}$	22	**4**	24	41	28	44	30	25	13	27	128	450	14	6	26	31	12	17
$V_f^{(14)}$	20	**5**	23	41	29	43	28	24	13	27	124	442	16	**5**	25	32	13	18
$V_t^{(15)}$	48	22	37	50	38	37	17	11	13	29	109	366	42	33	**7**	19	11	12
$V_f^{(15)}$	45	22	37	50	38	36	16	11	15	31	108	361	46	32	**8**	21	13	15
$V_t^{(16)}$	51	27	40	34	20	65	50	31	25	**16**	141	460	43	36	35	19	22	26
$V_f^{(16)}$	46	25	38	33	19	62	47	29	23	**16**	135	452	42	31	32	17	21	24
$V_t^{(17)}$	38	11	30	40	25	45	21	10	5	18	122	431	31	22	11	21	**3**	7
$V_f^{(17)}$	34	11	29	39	25	43	19	10	5	19	119	423	33	19	10	21	**4**	8
$V_t^{(18)}$	38	13	31	44	31	33	17	12	**6**	26	112	385	34	25	8	18	7	**6**
$V_f^{(18)}$	35	13	31	45	33	33	17	13	**7**	29	111	379	37	24	9	20	9	**7**

Note: The bold items are the ones having the smallest value in their respective rows.

REFERENCES

[1] T. Poggio and F. Girosi. Networks for approximation and learning, *Proc. IEEE*, MIT, Cambridge, MA, USA, September 1990, vol. 78, no. 9, pp. 1481–1497, 1990.

[2] J. Park and J. W. Sandberg. Universal approximation using radial-basis-function networks, *Neural Computation*, vol. 3, pp. 246–257, 1991.

[3] S. Haykin, *Neural networks: a comprehensive foundation*, Macmillan College Publishing Company, New York, 1994.

[4] A. Esposito, M. Marinaro, D. Oricchio, and S. Scarpetta. Approximation of continuous and discontinuous mappings by a growing neural RBF-based algorithm, *Neural Networks*, vol. 13, pp. 651–665, 2000.

[5] Z. Wang and T. Zhu. An efficient learning algorithm for improving generalization performance of radial basis function neural networks, *Neural Networks*, vol. 13, pp. 545–553, 2000.

[6] A. Alexandridis, H. Sarimveis, and G. Bafas. A new algorithm for online structure and parameter adaptation of RBF networks, *Neural Networks*, vol. 16, pp. 1003–1017, 2003.

[7] A. Leonardis and H. Bischof. An efficient MDL-based construction of RBF networks, *Neural Networks*, vol. 11, pp. 963–973, 1998.

[8] J. Liu and J. Lampinen. Differential evolution algorithm as a tool of training radial basis function networks, in *Graduate student workshop of Genetic and Evolutionary Computation Conference*, Seattle, Washington, USA, June 26-30, 2004.

[9] J. Liu, S. Kukkonen, and J. Lampinen. Function approximation with pruned radial basis function networks using a DE-based algorithm: an initial investigation, in 10th *Int'l Conf. on Soft Computing (MENDEL*

TABLE VI

CLASSIFICATION RESULTS

Working sample	class classified	Working sample	class classified	class belong
$V_t^{(1)}$	1	$V_f^{(1)}$	1	1
$V_t^{(2)}$	2	$V_f^{(2)}$	2	2
$V_t^{(3)}$	3	$V_f^{(3)}$	3	3
$V_t^{(4)}$	4	$V_f^{(4)}$	4	4
$V_t^{(5)}$	5	$V_f^{(5)}$	5	5
$V_t^{(6)}$	6	$V_f^{(6)}$	6	6
$V_t^{(7)}$	7	$V_f^{(7)}$	7	7
$V_t^{(8)}$	17	$V_f^{(8)}$	8,15,17	8
$V_t^{(9)}$	9	$V_f^{(9)}$	9	9
$V_t^{(10)}$	10	$V_f^{(10)}$	10	10
$V_t^{(11)}$	11	$V_f^{(11)}$	11	11
$V_t^{(12)}$	12	$V_f^{(12)}$	12	12
$V_t^{(13)}$	13	$V_f^{(13)}$	13	13
$V_t^{(14)}$	2	$V_f^{(14)}$	2,14	14
$V_t^{(15)}$	15	$V_f^{(15)}$	15	15
$V_t^{(16)}$	10	$V_f^{(16)}$	10	16
$V_t^{(17)}$	17	$V_f^{(17)}$	17	17
$V_t^{(18)}$	9,18	$V_f^{(18)}$	9,18	18

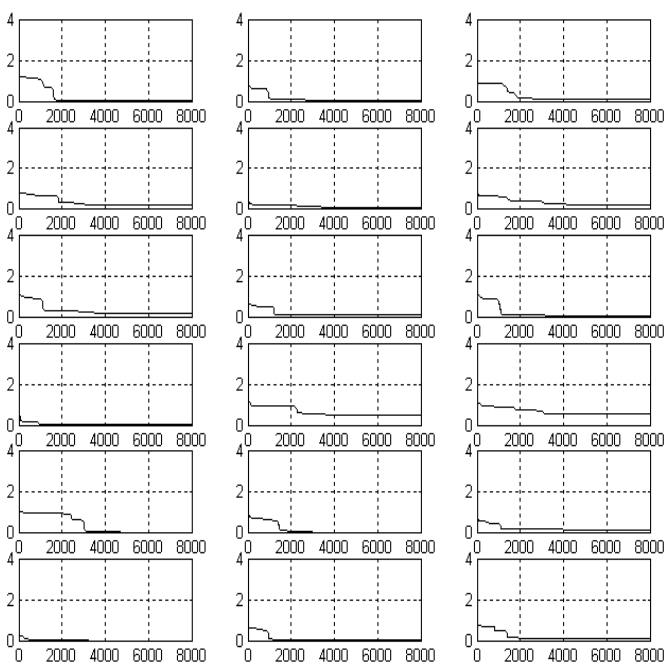

Fig. 5. Optimization processes of finding the RBF parameters. In each figure, the horizontal axis represents generations and the vertical axis represents the objective function values (samples indices are numbered from left to right top to bottom)

2004), Brno, Czech Republic, June 16-18 2004, pp. 139–144.

[10] Q. Zhu, Y. Cai, and L. Liu. A global learning algorithm for a RBF network, *Neural Networks*, vol. 12, pp. 527–540, 1999.

[11] V. P. Plagianakos and M. N. Vrahatis. Neural network training with constrained integer weights, *Proc. 1999 Congress on Evolutionary Computation*, Washington, DC USA, 6-9 July 1999, vol. 3, pp. 2007–2013.

[12] S. A. Billings and G. L. Zheng. Radial basis function networks configuration using genetic algorithms, *Neural Networks*, vol. 8, no. 6, pp. 877–890, 1995.

[13] G. P. J. Schmitz and C. Aldrich. Combinatorial evolution of regression nodes in feedforward neural networks, *Neural Networks*, vol. 12, pp. 175–189, 1999.

[14] A. G. Bors and I. Pitas. Median radial basis function neural network, *IEEE Trans. on Neural Networks*, vol. 7, no. 6, pp. 1351–1364, 1996.

[15] C. F. Fung, S. A. Billings, and W. Luo. On-line supervised adaptive training using radial basis function networks, *Neural Networks*, vol. 9, no. 9, pp. 1597–1617, 1996.

[16] M. Marinaro and S. Scarpetta. On-line learning in RBF neural networks: a stochastic approach, *Neural Networks*, vol. 13, pp. 719–729, 2000.

[17] D. S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks, *Complex Systems*, vol. 2, pp. 321–355, 1988.

[18] 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.

[19] R. J. Kuo and P. H. Cohen. Multi-sensor integration for on-line tool wear estimation through radial basis function networks and fuzzy neural network, *Neural Networks*, vol. 12, pp. 355–370, 1999.

[20] Y. S. Hwang and S. Y. Bang. An efficient method to construct a radial basis function neural network classifier, *Neural Networks*, vol. 10, no. 8, pp. 1495–1503, 1997.

[21] Q. Zhao and Z. Bao. Radar target recognition using a radial basis function neural network, *Neural Networks*, vol. 9, no. 4, pp. 709–720, 1996.

[22] I. Cha and S. A. Kassam. RBFN restoration of nonlinearly degraded images, *IEEE Trans. on Image Processing*, vol. 5, no. 6, pp. 964–975, 1996.

[23] H. Bischof and A. Leonardis. View-based object representation using RBF networks, *Image and Vision Computing*, vol. 19, pp. 619–629, 2001.

[24] A. G. Bors and I. Pitas. Optical flow estimation and moving object segmentation based on median radial basis function network, *IEEE Trans. on Image Processing*, vol. 7, no. 5, pp. 693–702, 1998.

[25] R. Storn and K. Price. Differential Evolution - a simple and efficient heuristic for global optimization over continuous spaces, *Journal of Global Optimization* 11(4): 341–359, December 1997.

[26] R. Storn. Differential evolution design of an IIR-filter, in *Proc. of IEEE Int'l Conf. on Evolutionary Computation*, Nagoya Japan, 20-22 May 1996, pp. 268–273.

[27] R. Storn and K. Price. Differential Evolution - a simple evolution strategy for fast optimization, *Dr. Dobb's Journal* 22(4): 18–24 and 78, April 1997.

[28] D. M. Bates and D. G. Watts. *Nonlinear regression analysis and its applications*, New York: Wiley, 1988.

[29] R. Storn and K. Price. On the usage of Differential Evolution for function optimization, in *1996 Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS)*, Berkeley, CA USA, 19-22 June, pp. 519–523, 1996.

[30] J. Lampinen and I. Zelinka. On stagnation of the differential evolution algorithm, in 6th *Int'l Conf. on Soft Computing (MENDEL 2000)*, Brno, Czech Republic, June 7-9 2000, pp. 76–83.

[31] J. Liu and J. Lampinen. On setting the control parameters of the differential evolution algorithm, in 8th *Int'l Conf. on Soft Computing (MENDEL 2002)*, Brno, Czech Republic, June 5-7 2002, pp. 11–18, 2002.

[32] V. Vinod and S. Ghose. Growing nonuniform feedforward networks for continuous mapping. *Neural computing*, vol. 10, pp. 55-69, 1996

[33] F. Yong and T. Chow: Neural network adaptive wavelets for function approximation. *Intern report.* Department of Electrical Engineering. City University of Hong Kong, 1996