A Differential Evolution Based Incremental Training Method for RBF Networks

Junhong Liu*
Department of Information Technology
Lappeenranta University of Technology
P.O.Box 20
FIN-53851 Lappeenranta, Finland
liu@lut.fi

Jouni Lampinen
Department of Information Technology
Lappeenranta University of Technology
P.O.Box 20
FIN-53851 Lappeenranta, Finland
jlampine@lut.fi

ABSTRACT

The Differential Evolution (DE) is a floating-point encoded evolutionary strategy 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 a DE-based algorithm to training Radial Basis Function (RBF) networks with variables including centres, weights, and widths of RBFs. The proposed algorithm consists of three steps: the first step is the initial tuning, which focuses on searching for the center, weight, and width of a one-node RBF network, the second step is the local tuning, which optimizes the three variables of the one-node RBF network — its centre, weight, and width, and the third step is the global tuning, which optimizes all the parameters of the whole network together. The second step and the third step both use the cycling scheme to find the parameters of RBF network. The Mean Square Error from the desired to actual outputs is applied as the objective function to be minimized. Training the networks is demonstrated by approximating a set of functions, using different strategies of DE. A comparison of the net performances with several approaches reported in the literature is given and shows the resulting network performs better in the tested functions. The results show that proposed method improves the compared approximation results.

Categories and Subject Descriptors

G.1.6 []: *Global optimization*; G.1.2 []: *Nonlinear approximation*; I.1.2 []: *Nonalgebraic algorithms*

General Terms

Algorithms

*Corresponding author

GECCO'05, June 25–29, 2005, Washington, DC, USA.

Keywords

Evolutionary strategies, Neural networks, Optimization, Radial basis functions, Differential evolution

1. INTRODUCTION

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 [16, 18]. Due to their nonlinear approximation properties, RBF networks are able to model complex mappings [5, 7]. 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 RBFs as well as their locations, orientations, 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 RBF networks have been developed and can be divided into four categories [1, 2, 9, 10, 17, 20, 25, 27]: (*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, orientations, widths, weights, or the network structure together. On-line training algorithms adapt the network parameters to the changing data statistics [1, 3, 6, 14]. RBF networks have been successfully applied to a large diversity of applications including interpolation [4, 15], etc.

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. RBFs using Evolutionary techniques have the following advantages:

1. Evolutionary algorithms (EAs) have been successfully applied to finding the global optima of various multidimensional functions where local optima in the space of possible solution are common, while in training RBF networks, the global optimum is usually surrounded by local optima.

2. EAs are able to handle the optimization of parameters for which no gradient information or other auxiliary information is available, while in training incremental RBF networks, the objective function changes with node that increases with generation and the objective function parameters evolve with generation so that offering gradient information or any other auxiliary information would become burdensome.

3. EAs can optimize a broader range of network parameters; even adaptively change the type of the transfer functions of nodes because of discontinuities of the discussed functions [5].

The Differential Evolution (DE) introduced by Price and Storn [22], a floating-point encoded EA for global optimization, has been found to be an efficient, effective, and robust optimization method, especially for problems containing continuous problem variables [19, 22, 23]. This paper applies a DE-based incremental training method to searching for variables — centres, weights, and widths — of the RBF network, which provides the best possible function approximation.

The rest of the paper is structured as follows: the training problem is explained in Sect. 2, the training method is described in Sect. 3, and experimental results are shown in Sect. 4. The conclusion is given in Sect. 5.

2. TRAINING PROBLEM

2.1 Radial Basis Functions Network

RBFs have their origin in the solution of the multivariate interpolation problem [4]. 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:

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

where $\mathbf{v} \in \Re^d$; $\mathbf{x}$ is the vector of variable factors including 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; $r_j(\cdot)$ 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 $\mathbf{c}_j$. In order to simplify the notation, coordinate axes-aligned Gaussian RBF functions are used. When a 1-D Gaussian RBF is centred at the centroids $\mathbf{c}_j$, it follows from (1) that

$$\begin{aligned} \widehat{g}(\mathbf{v},\mathbf{x}) &= \sum_{j=1}^{p} w_j \exp(-\frac{\|\mathbf{v}-\mathbf{c}_j\|^2}{2\sigma_j^2}) \\ &= \sum_{j=1}^{p} w_j exp(-\frac{(v-c_j)^2}{2\sigma_j^2}), \end{aligned} \quad (2)$$

where $\mathbf{x}$ can be written as

$$\begin{aligned} \mathbf{x}^T &= (\mathbf{w}^T, \boldsymbol{\sigma}_1^T, {\mathbf{c}_1}^T, ..., \boldsymbol{\sigma}_j^T, {\mathbf{c}_j}^T, ..., \boldsymbol{\sigma}_p^T, {\mathbf{c}_p}^T)^T \\ &= (w_1, ..., w_p, \sigma_1, c_1, ..., \sigma_j, c_j, ..., \sigma_p, c_p)^T. \end{aligned} \quad (3)$$

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

2.2 Cost Function

Given the number of RBFs and a training set, the network parameters are found such that they minimize the Mean Square Error (MSE) between the desired and actual outputs, i.e., the objective function:

$$f(\mathbf{x}) = \frac{1}{N} \sum_{n=1}^{N} \Big(y_n - \widehat{g}(\mathbf{v}_n, \mathbf{x}) \Big)^2. \quad (4)$$

2.3 Description of Differential Evolution

$$\text{Minimize} : f(\mathbf{x}) : \Re^D \rightarrow \Re, \quad (5)$$

by finding the optimal vector, $\mathbf{x} = (x_1, x_2, ..., x_D)^T$, composed of D parameters, which are usually subject to the lower and upper boundary constraints $\mathbf{x}^{(\mathrm{L})}$ and $\mathbf{x}^{(\mathrm{U})}$:

$$x_k^{(\mathrm{L})} \leq x_k \leq x_k^{(\mathrm{U})}, k = 1, 2, ..., D. \quad (6)$$

DE utilizes a population of parameter vectors (Fig. 1):

$$P_G = (\mathbf{x}_{1,G}, \mathbf{x}_{2,G}, ..., \mathbf{x}_{N_{pop},G}), G = 1, ..., G_{max}, \quad (7)$$

$$\mathbf{x}_{i,G} = (x_{1,i,G}, x_{2,i,G}, ..., x_{D,i,G}), i = 1, ..., N_{pop}, \quad (8)$$

where N_{pop} is the population size, G is the generation index, and G_{max} is the maximum generation.

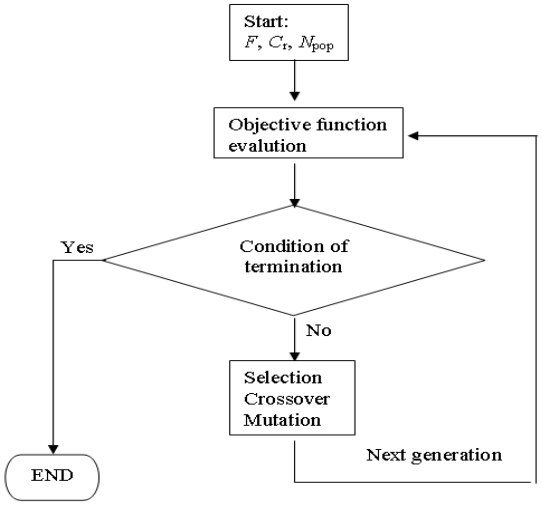

Figure 1: The DE algorithm.

The initial population is chosen randomly to cover the entire parameter space uniformly (unless otherwise stated). A natural way to seed the initial population P_0 is to generate random values within the given boundary constraints:

$$P_0 = \{\mathbf{x}_{i,0}, i = 1, ..., N_{pop}\}, \quad (9)$$

$$\mathbf{x}_{i,0} = \{x_{j,i,0} = rnd_j \cdot (x_j^{(\mathrm{U})} - x_j^{(\mathrm{L})}) + x_j^{(\mathrm{L})}, \quad (10)$$
$$j = 1, ..., D\},$$

where rnd_j denotes a uniformly distributed random value in the range of $[0, 1)$ for each j.

There are several variants of DE (Table 1): (*i*) in mutation, the vector subject to perturbation can be a random member of the current population, the current best member, or the target member, marked as *rand*, *best*, and *rand-to-best* respectively; (*ii*) *1* indicates that one difference vector is used and *2* two difference vectors; (*iii*) *exp* and *bin* are exponential and binomial crossover operations. For example, if Strategy 7 is used, its main ingredients are:

1. Mutation: for each vector $\mathbf{x}_{i,G}$, a perturbed vector $\mathbf{v}$ is generated according to

$$\mathbf{v}_{i,G+1} = \mathbf{x}_{r_1,G} + F \cdot (\mathbf{x}_{r_2,G} - \mathbf{x}_{r_3,G}),$$

where randomly selected $r_1, r_2, r_3 \in \{1, ..., N_{pop}\}$, $r_1 \neq r_2 \neq r_3 \neq i$, and mutation parameter $F \in (0, 1+]$.

2. Crossover: the trial vector is generated as follows:

$$\mathbf{u}_{i,G+1} = (u_{1,i,G+1}, u_{2,i,G+1}, ..., u_{D,i,G+1}),$$

$$u_{j,i,G+1} = \begin{cases} v_{j,i,G+1} & \text{if } rnd_j \leq C_r \vee j = k_i, \\ x_{j,i,G} & \text{otherwise,} \end{cases}$$

where parameter index $j = 1, ..., D$, population index $i = 1, ..., N_{pop}$, generation index $G = 1, ..., G_{max}$, random parameter index $k_i \in \{1, ..., D\}$, and crossover parameter $C_r \in [0, 1]$.

3. Selection: population P_{G+1} for the next generation is selected from the current population P_G and the child population P'_{G+1}, according to the following rule:

$$\mathbf{x}_{i,G+1} = \begin{cases} \mathbf{u}_{i,G+1} & \text{if } f(\mathbf{u}_{i,G+1}) \leq f(\mathbf{x}_{i,G}), \\ \mathbf{x}_{i,G} & \text{otherwise.} \end{cases}$$

A noisy parameter vector is generated by mutation operation. In crossover operation, a trial vector is obtained. If the resulting trial vector yields a lower or equal objective function value than the predetermined vector, the trial vector is sent to the next generation; otherwise, the counterpart is retained [21].

Table 1: Strategies of DE

Strategy	Scheme*
1	DE/best/1/exp
2	DE/rand/1/exp
3	DE/rand-to-best/1/exp
4	DE/best/2/exp
5	DE/rand/2/exp
6	DE/best/1/bin
7	DE/rand/1/bin
8	DE/rand-to-best/1/bin
9	DE/best/2/bin
10	DE/rand/2/bin

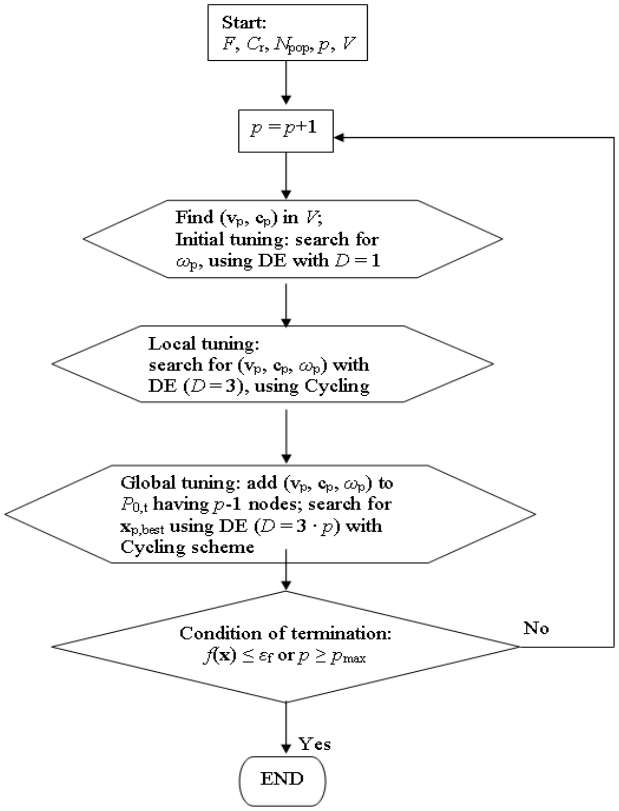

Figure 2: The DE-based training RBFN algorithm.

3. TRAINING METHOD

3.1 Algorithm

Figure 2 shows the algorithm training RBF networks.

3.1.1 Initial tuning

It searches the centre, weight, and width of a one-node RBF network. The sample pair $(\mathbf{v}_i, y_i)$ from set V, having the biggest difference between the desired and actual outputs of the existing RBF network, 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, is chosen. The node has centre $\mathbf{v}_i$, weight $y_i - \hat{g}_{p-1}(\mathbf{v}_i)$, and the width obtained after running DE for G_{local} generations, which minimizes the MSE between the desired and actual outputs of the present RBF network.

3.1.2 Local tuning

The centre, weight, and width of the one-node RBF network are replicated to get a population. Optimal parameters of the one-node RBF network are determined by the cycling scheme, which will repeat running DE G_{local} generations for γ times with the objective function (4) of dimensionality 3.

3.1.3 Global tuning

It searches for the parameters of the whole RBF network by the cycling scheme. p is increased by one and the node from the local tuning is added to each population member from last global tuning. The cycling scheme repeats running DE for γ times, while the function (4) has dimensionality of $D = 3 \cdot p$ and the maximum number of generations of each run is $G_{global} = G_{local} \cdot \lceil \sqrt{p} \rceil$.

3.1.4 Cycling scheme

It searches the parameters of RBF network and is conducted in both the local tuning and global tuning by regenerating the initial population based on the current population. The regeneration can be explained as the following:

$$\mathbf{x}'_{i,0,t+1} = \mathbf{x}_{i,G,t} + \mathbf{m}\mathbf{x}_{i,G,t}, \tag{13}$$

where $\mathbf{x}_{i,G,t}$ is the i^{th} population member of generation G at cycle t, $\mathbf{x}'_{i,0,t+1}$ is the i^{th} initial population member for cycle $t+1$, and $\mathbf{m} = AB\tau$ with A a diagonal matrix having D random values from a normal distribution, B a diagonal matrix having the i^{th} element of $\mathbf{x}^{(\mathrm{U})} - \mathbf{x}^{(\mathrm{L})}$ at b_{ii} $(1 \le i \le D)$, and τ a constant. The cycling scheme consists of γ time repeats, with each accomplished by running DE for G_{local} (the local tuning) or G_{global} generations (the global tuning).

3.1.5 Condition of termination

The process, conducting the initial tuning, local tuning, and global tuning one after one, will continue if $f(x) > \epsilon_f$ and $p < p_{max}$, where ϵ_f is a threshold and p_{max} is the maximum number of RBFs, both of which will be numerically agreed on in the experiments in Sect. 4.

3.1.6 Testing performance

For each run of a function, testing set V_t and training set V are generated anew independently and contain the same number of samples – uniformly distributed random values in the function domain.

3.1.7 Handling constraint violation

The parameters of the objective function are bounded by the defined region and the boundaries of the discussed function, $g(\mathbf{v})$, i.e., $\mathbf{c}_j \in [-0.25 + \mathbf{v}^{(\mathrm{L})}, 0.25 + \mathbf{v}^{(\mathrm{U})}]$, $\boldsymbol{\sigma}_j \in (0, (\mathbf{v}^{(\mathrm{U})} - \mathbf{v}^{(\mathrm{L})})/2]$, and $w_j \in [-2 \cdot |g|_{max}, +2 \cdot |g|_{max}]$, where $\mathbf{v}^{(\mathrm{L})}$ and $\mathbf{v}^{(\mathrm{U})}$ are the vectors of the lower and upper limits of $\mathbf{v}$ respectively, and $|g|_{max}$ is the maximum of the absolute function values of $g(\mathbf{v})$. After regeneration, mutation, and crossover, individuals that go out of the parameters' boundaries should be replaced by random values generated within the boundaries. The objective function values are calculated according to (4) after individuals have been repaired.

3.2 Control Parameters' Setting of DE

The strategies are listed in Table 1 as in [13, 21, 23]. The control parameters' setting affects the algorithm's performance and its values were chosen based on discussions in [8, 11, 21]. All strategies use the same setting: $F = 0.9$, $C_r = 0.9$, and $N_{pop} = 100$.

4. EXPERIMENTAL RESULTS

The proposed approach is applied to single-input and single-output test functions in Table 2 without any noise. For function $g(\mathbf{v})$, training set V is considered as:

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

where d denotes the dimension of $g(\mathbf{v})$, (i.e., $d = 1$ for single-input functions and so on) and $\mathbf{v}_n$, $n = 1, 2, ..., N$, are N uniformly distributed random values in the defined region of the computed function. In order to get general performance, each function was tested for 30 times with each strategy.

Table 2: Test functions

Index	Test function
1	$g_1(v) = 1.1 \cdot (1 - v - 2v^2) \cdot \exp(-v^2/2)$, $v \in [-4, 4]$
2	$g_2(v) = \sin(12v), v \in [0, 1]$
3	$g_3(v) = \sin(20v^2), v \in [0, 1]$
4	$g_4(v) = 1 + (v + 2v^2)\sin(-v^2), v \in [-4, 4]$
5	$g_5(v) = \sin(v) + \sin(3v) + \sin(5v)$, $v \in [-0.35, 3.5]$
6	$g_6(v) = 0.5 \cdot \exp(-v) \cdot sin(6v)$, $v \in [-1, 1]$
7	$g_7(v) = \sin(v) + \sin(3v) + \sin(6v)$, $v \in [-0.35, 3.5]$
8	$g_8(v) = \sum_{n=1}^{4} \left(n \cdot \exp(-n^2 \cdot (v - 4 + 2n)^2)\right)$, $v \in [-4, 4]$
9	$g_9(v) = v, v \in [0, 1]$

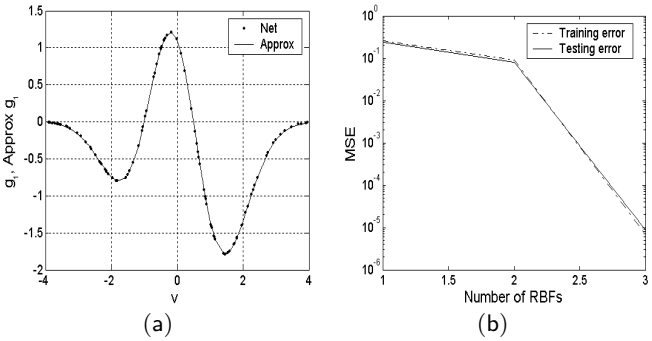

Figure 3: An example with Strategy 1. (a) g_1 and $\hat{g}_1$; (b) The training error and testing error with node.

4.1 Experiments with Test Functions

Each function in Table 2 was used, setting $N = 100$, while ϵ_f and p_{max} were set as the training error and the number of nodes found in [12]. Key experimental results are:

1. Fig. 3 shows an example of the net function, its approximation, the training error, and the testing error. Table 3 and Table 4 list the numerical results.

2. As can be seen from Fig. 3, Table 3, and Table 4 the DE-based trainings are effective with respect to lower MSE and a lower number of nodes. The proposed algorithm is efficient since the functions in Table 2 can be represented by a number of RBFs, each with a relatively small error compared to the respective range of

Table 3: Experimental results with test functions (to be continued)

Func -tion	Method	p	Training			Testing		
			Training error	Max. error	*std*	Testing error	Max. error	*std*
g_1	S_1	3	$6.2688 \cdot 10^{-6}$	$4.2403 \cdot 10^{-5}$	$1.30 \cdot 10^{-3}$	$6.9504 \cdot 10^{-6}$	$5.20 \cdot 10^{-3}$	$1.40 \cdot 10^{-3}$
	S_2	4	$2.8334 \cdot 10^{-4}$	$1.3004 \cdot 10^{-4}$	$1.02 \cdot 10^{-2}$	$3.0721 \cdot 10^{-4}$	$3.87 \cdot 10^{-2}$	$1.03 \cdot 10^{-2}$
	S_3	3	$7.7167 \cdot 10^{-6}$	$6.0969 \cdot 10^{-5}$	$1.50 \cdot 10^{-3}$	$8.6393 \cdot 10^{-6}$	$6.20 \cdot 10^{-3}$	$1.60 \cdot 10^{-3}$
	S_4	5	$1.2000 \cdot 10^{-3}$	$2.3049 \cdot 10^{-4}$	$2.10 \cdot 10^{-2}$	$1.2000 \cdot 10^{-3}$	$7.45 \cdot 10^{-2}$	$2.10 \cdot 10^{-2}$
	S_5	7	$5.0188 \cdot 10^{-4}$	$1.3477 \cdot 10^{-4}$	$1.46 \cdot 10^{-2}$	$5.7918 \cdot 10^{-4}$	$5.83 \cdot 10^{-2}$	$1.52 \cdot 10^{-2}$
	S_6	**3**	$\mathbf{5.8461 \cdot 10^{-6}}$	$\mathbf{4.6424 \cdot 10^{-5}}$	$\mathbf{1.30 \cdot 10^{-3}}$	$\mathbf{6.8661 \cdot 10^{-6}}$	$\mathbf{5.30 \cdot 10^{-3}}$	$\mathbf{1.40 \cdot 10^{-3}}$
	S_7	4	$4.8761 \cdot 10^{-4}$	$1.9768 \cdot 10^{-4}$	$1.19 \cdot 10^{-2}$	$4.6489 \cdot 10^{-4}$	$4.34 \cdot 10^{-2}$	$1.19 \cdot 10^{-2}$
	S_8	3	$7.2710 \cdot 10^{-6}$	$4.4102 \cdot 10^{-5}$	$1.50 \cdot 10^{-3}$	$8.1637 \cdot 10^{-6}$	$6.00 \cdot 10^{-3}$	$1.60 \cdot 10^{-3}$
	S_9	5	$1.1000 \cdot 10^{-3}$	$2.3310 \cdot 10^{-4}$	$1.93 \cdot 10^{-2}$	$1.0000 \cdot 10^{-3}$	$6.67 \cdot 10^{-2}$	$1.88 \cdot 10^{-2}$
	S_{10}	7	$4.9793 \cdot 10^{-4}$	$1.2220 \cdot 10^{-4}$	$1.39 \cdot 10^{-2}$	$5.2305 \cdot 10^{-4}$	$5.15 \cdot 10^{-2}$	$1.38 \cdot 10^{-2}$
	DE0	16	$9.4648 \cdot 10^{-5}$	$2.2400 \cdot 10^{-4}$	$5.40 \cdot 10^{-3}$	$1.3142 \cdot 10^{-4}$	$1.64 \cdot 10^{-2}$	$5.30 \cdot 10^{-3}$
g_2	S_1	**4**	$\mathbf{6.4807 \cdot 10^{-4}}$	$\mathbf{1.7292 \cdot 10^{-4}}$	$\mathbf{2.02 \cdot 10^{-2}}$	$\mathbf{6.6467 \cdot 10^{-4}}$	$\mathbf{1.251 \cdot 10^{-1}}$	$\mathbf{2.01 \cdot 10^{-2}}$
	S_2	7	$3.9397 \cdot 10^{-4}$	$1.0657 \cdot 10^{-4}$	$1.25 \cdot 10^{-2}$	$6.9897 \cdot 10^{-4}$	$1.058 \cdot 10^{-1}$	$1.79 \cdot 10^{-2}$
	S_3	4	$6.0102 \cdot 10^{-4}$	$1.8221 \cdot 10^{-4}$	$1.94 \cdot 10^{-2}$	$6.6546 \cdot 10^{-4}$	$1.284 \cdot 10^{-1}$	$2.02 \cdot 10^{-2}$
	S_4	8	$4.4979 \cdot 10^{-4}$	$1.2539 \cdot 10^{-4}$	$1.39 \cdot 10^{-2}$	$9.1095 \cdot 10^{-4}$	$1.031 \cdot 10^{-1}$	$2.01 \cdot 10^{-2}$
	S_5	9	$7.0127 \cdot 10^{-4}$	$1.5380 \cdot 10^{-4}$	$1.79 \cdot 10^{-2}$	$3.7000 \cdot 10^{-3}$	$2.155 \cdot 10^{-1}$	$3.31 \cdot 10^{-2}$
	S_6	4	$5.1323 \cdot 10^{-4}$	$1.8767 \cdot 10^{-4}$	$1.81 \cdot 10^{-2}$	$7.9480 \cdot 10^{-4}$	$1.416 \cdot 10^{-1}$	$2.23 \cdot 10^{-2}$
	S_7	6	$4.9819 \cdot 10^{-4}$	$1.6998 \cdot 10^{-4}$	$1.38 \cdot 10^{-2}$	$6.8989 \cdot 10^{-4}$	$8.100 \cdot 10^{-2}$	$1.66 \cdot 10^{-2}$
	S_8	4	$4.7896 \cdot 10^{-4}$	$1.4639 \cdot 10^{-4}$	$1.69 \cdot 10^{-2}$	$7.3431 \cdot 10^{-4}$	$1.435 \cdot 10^{-1}$	$2.16 \cdot 10^{-2}$
	S_9	8	$4.7025 \cdot 10^{-4}$	$1.2053 \cdot 10^{-4}$	$1.40 \cdot 10^{-2}$	$7.2490 \cdot 10^{-4}$	$9.340 \cdot 10^{-2}$	$1.82 \cdot 10^{-2}$
	S_{10}	9	$1.0000 \cdot 10^{-3}$	$2.1320 \cdot 10^{-4}$	$2.13 \cdot 10^{-2}$	$1.8000 \cdot 10^{-3}$	$1.256 \cdot 10^{-1}$	$2.71 \cdot 10^{-2}$
	DE0	10	$1.8412 \cdot 10^{-4}$	$8.2400 \cdot 10^{-4}$	$9.40 \cdot 10^{-3}$	$1.1533 \cdot 10^{-4}$	$3.010 \cdot 10^{-2}$	$5.50 \cdot 10^{-3}$
g_3	S_1	**6**	$\mathbf{1.1800 \cdot 10^{-2}}$	$\mathbf{1.9924 \cdot 10^{-4}}$	$\mathbf{9.84 \cdot 10^{-2}}$	$\mathbf{1.390 \cdot 10^{-2}}$	$\mathbf{6.343 \cdot 10^{-1}}$	$\mathbf{9.700 \cdot 10^{-2}}$
	S_2	12	$2.5000 \cdot 10^{-3}$	$1.8365 \cdot 10^{-4}$	$3.33 \cdot 10^{-2}$	$1.510 \cdot 10^{-2}$	$5.030 \cdot 10^{-1}$	$8.290 \cdot 10^{-2}$
	S_3	6	$1.1500 \cdot 10^{-2}$	$2.2820 \cdot 10^{-4}$	$8.92 \cdot 10^{-2}$	$1.500 \cdot 10^{-2}$	$6.971 \cdot 10^{-1}$	$1.013 \cdot 10^{-1}$
	S_4	12	$2.7000 \cdot 10^{-3}$	$3.7319 \cdot 10^{-4}$	$3.50 \cdot 10^{-2}$	$1.050 \cdot 10^{-2}$	$4.660 \cdot 10^{-1}$	$7.260 \cdot 10^{-2}$
	S_5	11	$4.2000 \cdot 10^{-3}$	$3.7551 \cdot 10^{-4}$	$4.21 \cdot 10^{-2}$	$9.300 \cdot 10^{-3}$	$4.172 \cdot 10^{-1}$	$6.860 \cdot 10^{-2}$
	S_6	6	$1.2100 \cdot 10^{-2}$	$2.3301 \cdot 10^{-4}$	$9.38 \cdot 10^{-2}$	$2.260 \cdot 10^{-2}$	$7.409 \cdot 10^{-1}$	$1.240 \cdot 10^{-1}$
	S_7	10	$3.8000 \cdot 10^{-3}$	$2.4183 \cdot 10^{-4}$	$4.10 \cdot 10^{-2}$	$9.300 \cdot 10^{-3}$	$3.668 \cdot 10^{-1}$	$6.420 \cdot 10^{-2}$
	S_8	6	$1.0800 \cdot 10^{-2}$	$1.7173 \cdot 10^{-4}$	$9.16 \cdot 10^{-2}$	$1.920 \cdot 10^{-2}$	$7.843 \cdot 10^{-1}$	$1.189 \cdot 10^{-1}$
	S_9	11	$4.7000 \cdot 10^{-3}$	$3.0143 \cdot 10^{-4}$	$4.46 \cdot 10^{-2}$	$1.330 \cdot 10^{-2}$	$4.654 \cdot 10^{-1}$	$8.050 \cdot 10^{-2}$
	S_{10}	11	$4.3000 \cdot 10^{-3}$	$3.9439 \cdot 10^{-4}$	$4.37 \cdot 10^{-2}$	$1.020 \cdot 10^{-2}$	$4.482 \cdot 10^{-1}$	$7.270 \cdot 10^{-2}$
	DE0	14	$8.6318 \cdot 10^{-4}$	$7.0700 \cdot 10^{-4}$	$1.79 \cdot 10^{-2}$	$7.732 \cdot 10^{-4}$	$6.970 \cdot 10^{-2}$	$1.680 \cdot 10^{-2}$
g_4	S_1	8	2.0374	$1.0100 \cdot 10^{-2}$	$1.0499 \cdot 10^{0}$	$1.10406 \cdot 10^{1}$	$1.45972 \cdot 10^{1}$	$2.5048 \cdot 10^{0}$
	S_2	8	2.8345	$2.4700 \cdot 10^{-2}$	$1.1608 \cdot 10^{0}$	$1.22190 \cdot 10^{1}$	$1.58814 \cdot 10^{1}$	$2.6610 \cdot 10^{0}$
	S_3	**7**	**5.0079**	$\mathbf{1.3000 \cdot 10^{-2}}$	$\mathbf{1.7083 \cdot 10^{0}}$	$\mathbf{1.50913 \cdot 10^{1}}$	$\mathbf{1.63633 \cdot 10^{1}}$	$\mathbf{3.0321 \cdot 10^{0}}$
	S_4	8	2.5437	$1.5100 \cdot 10^{-2}$	$1.0966 \cdot 10^{0}$	$1.11552 \cdot 10^{1}$	$1.49150 \cdot 10^{1}$	$2.4646 \cdot 10^{0}$
	S_5	8	2.8898	$1.0600 \cdot 10^{-2}$	$1.1637 \cdot 10^{0}$	$1.35024 \cdot 10^{1}$	$1.75178 \cdot 10^{1}$	$2.8050 \cdot 10^{0}$
	S_6	**8**	**2.1082**	$\mathbf{1.1900 \cdot 10^{-2}}$	$\mathbf{1.0720 \cdot 10^{0}}$	$\mathbf{9.70850 \cdot 10^{0}}$	$\mathbf{1.48984 \cdot 10^{1}}$	$\mathbf{2.3313 \cdot 10^{0}}$
	S_7	8	3.2409	$1.1500 \cdot 10^{-2}$	$1.2671 \cdot 10^{0}$	$1.52609 \cdot 10^{1}$	$1.82320 \cdot 10^{1}$	$3.0016 \cdot 10^{0}$
	S_8	8	2.2163	$1.2100 \cdot 10^{-2}$	$1.0371 \cdot 10^{0}$	$1.30727 \cdot 10^{1}$	$1.61196 \cdot 10^{1}$	$2.7109 \cdot 10^{0}$
	S_9	8	2.5608	$9.7000 \cdot 10^{-3}$	$1.0957 \cdot 10^{0}$	$1.23909 \cdot 10^{1}$	$1.61828 \cdot 10^{1}$	$2.6356 \cdot 10^{0}$
	S_{10}	7	5.2230	$1.6800 \cdot 10^{-2}$	$1.6732 \cdot 10^{0}$	$1.29511 \cdot 10^{1}$	$1.42184 \cdot 10^{1}$	$2.7251 \cdot 10^{0}$
	DE0	22	1.9026	$8.1361 \cdot 10^{-2}$	$1.0579 \cdot 10^{-1}$	$1.81710 \cdot 10^{0}$	$7.08310 \cdot 10^{0}$	$9.9070 \cdot 10^{-1}$
g_5	S_1	6	$1.0000 \cdot 10^{-3}$	$3.0637 \cdot 10^{-4}$	$2.27 \cdot 10^{-2}$	$7.3000 \cdot 10^{-3}$	$2.252 \cdot 10^{-1}$	$4.44 \cdot 10^{-2}$
	S_2	10	$1.9000 \cdot 10^{-3}$	$2.7178 \cdot 10^{-4}$	$2.80 \cdot 10^{-2}$	$8.0000 \cdot 10^{-3}$	$4.054 \cdot 10^{-1}$	$6.15 \cdot 10^{-2}$
	S_3	6	$1.2000 \cdot 10^{-3}$	$2.3432 \cdot 10^{-4}$	$2.53 \cdot 10^{-2}$	$2.2000 \cdot 10^{-3}$	$1.597 \cdot 10^{-1}$	$3.27 \cdot 10^{-2}$
	S_4	10	$1.9000 \cdot 10^{-3}$	$2.9730 \cdot 10^{-4}$	$2.86 \cdot 10^{-2}$	$1.1400 \cdot 10^{-2}$	$3.930 \cdot 10^{-1}$	$6.66 \cdot 10^{-2}$
	S_5	10	$2.2000 \cdot 10^{-3}$	$5.4841 \cdot 10^{-4}$	$3.18 \cdot 10^{-2}$	$7.9000 \cdot 10^{-3}$	$3.579 \cdot 10^{-1}$	$5.77 \cdot 10^{-2}$
	S_6	6	$1.1000 \cdot 10^{-3}$	$3.1156 \cdot 10^{-4}$	$2.49 \cdot 10^{-2}$	$2.3000 \cdot 10^{-3}$	$1.684 \cdot 10^{-1}$	$3.37 \cdot 10^{-2}$
	S_7	10	$1.7000 \cdot 10^{-3}$	$3.5056 \cdot 10^{-4}$	$2.78 \cdot 10^{-2}$	$1.0800 \cdot 10^{-2}$	$4.023 \cdot 10^{-1}$	$6.63 \cdot 10^{-2}$
	S_8	**6**	$\mathbf{1.0000 \cdot 10^{-3}}$	$\mathbf{2.0291 \cdot 10^{-4}}$	$\mathbf{2.32 \cdot 10^{-2}}$	$\mathbf{1.7000 \cdot 10^{-3}}$	$\mathbf{1.581 \cdot 10^{-1}}$	$\mathbf{2.94 \cdot 10^{-2}}$
	S_9	10	$1.9000 \cdot 10^{-3}$	$2.4729 \cdot 10^{-4}$	$2.86 \cdot 10^{-2}$	$1.1200 \cdot 10^{-2}$	$4.492 \cdot 10^{-1}$	$7.21 \cdot 10^{-2}$
	S_{10}	10	$2.2000 \cdot 10^{-3}$	$5.4108 \cdot 10^{-4}$	$3.08 \cdot 10^{-2}$	$2.2900 \cdot 10^{-2}$	$5.179 \cdot 10^{-1}$	$8.57 \cdot 10^{-2}$
	DE0	10	$4.9735 \cdot 10^{-4}$	$5.1300 \cdot 10^{-4}$	$1.32 \cdot 10^{-2}$	$5.2135 \cdot 10^{-4}$	$5.160 \cdot 10^{-2}$	$1.31 \cdot 10^{-3}$

Note: "S_n" stands for Strategy n. *std* stands for standard deviation. p is the smallest value in 30 runs with the same strategy. Errors are average values of 30 runs.

Table 4: Experimental results with test functions (continuation of Table 3)

Function	Method	p	Training			Testing		
			Training error	Max. error	*std*	Testing error	Max. error	*std*
g_6	S_1	4	$1.5149 \cdot 10^{-4}$	$1.1702 \cdot 10^{-4}$	$9.50 \cdot 10^{-3}$	$2.3666 \cdot 10^{-4}$	$7.86 \cdot 10^{-2}$	$1.21 \cdot 10^{-2}$
	S_2	4	$2.5868 \cdot 10^{-4}$	$2.8790 \cdot 10^{-4}$	$1.07 \cdot 10^{-2}$	$3.8920 \cdot 10^{-4}$	$8.68 \cdot 10^{-2}$	$1.35 \cdot 10^{-2}$
	S_3	**4**	$\mathbf{1.6914 \cdot 10^{-4}}$	$\mathbf{8.8973 \cdot 10^{-5}}$	$\mathbf{1.00 \cdot 10^{-2}}$	$\mathbf{1.9087 \cdot 10^{-4}}$	$\mathbf{6.76 \cdot 10^{-2}}$	$\mathbf{1.07 \cdot 10^{-2}}$
	S_4	6	$2.2036 \cdot 10^{-4}$	$8.7193 \cdot 10^{-5}$	$9.30 \cdot 10^{-3}$	$3.5864 \cdot 10^{-4}$	$7.00 \cdot 10^{-2}$	$1.27 \cdot 10^{-2}$
	S_5	7	$2.4233 \cdot 10^{-4}$	$1.2773 \cdot 10^{-4}$	$9.90 \cdot 10^{-3}$	$4.3162 \cdot 10^{-4}$	$6.64 \cdot 10^{-2}$	$1.33 \cdot 10^{-2}$
	S_6	4	$1.3907 \cdot 10^{-4}$	$8.4124 \cdot 10^{-5}$	$9.10 \cdot 10^{-3}$	$2.0890 \cdot 10^{-4}$	$7.87 \cdot 10^{-2}$	$1.15 \cdot 10^{-2}$
	S_7	4	$2.8093 \cdot 10^{-4}$	$1.4095 \cdot 10^{-4}$	$1.20 \cdot 10^{-2}$	$4.1309 \cdot 10^{-4}$	$8.46 \cdot 10^{-2}$	$1.44 \cdot 10^{-2}$
	S_8	4	$1.3156 \cdot 10^{-4}$	$8.9528 \cdot 10^{-5}$	$8.90 \cdot 10^{-3}$	$2.5018 \cdot 10^{-4}$	$8.16 \cdot 10^{-2}$	$1.25 \cdot 10^{-2}$
	S_9	5	$3.0128 \cdot 10^{-4}$	$9.4716 \cdot 10^{-5}$	$1.09 \cdot 10^{-2}$	$4.1289 \cdot 10^{-4}$	$7.40 \cdot 10^{-2}$	$1.34 \cdot 10^{-2}$
	S_{10}	7	$2.8934 \cdot 10^{-4}$	$1.1403 \cdot 10^{-4}$	$1.03 \cdot 10^{-2}$	$4.0724 \cdot 10^{-4}$	$6.81 \cdot 10^{-2}$	$1.29 \cdot 10^{-2}$
	DE0	7	$1.1842 \cdot 10^{-4}$	$4.4200 \cdot 10^{-4}$	$6.70 \cdot 10^{-3}$	$7.1339 \cdot 10^{-5}$	$2.08 \cdot 10^{-2}$	$5.00 \cdot 10^{-3}$
g_7	S. 1	**6**	$\mathbf{2.7 \cdot 10^{-3}}$	$\mathbf{2.4544 \cdot 10^{-4}}$	$\mathbf{4.39 \cdot 10^{-2}}$	$\mathbf{3.00 \cdot 10^{-3}}$	$\mathbf{2.745 \cdot 10^{-1}}$	$\mathbf{4.57 \cdot 10^{-2}}$
	S_2	10	$6.8 \cdot 10^{-3}$	$5.8102 \cdot 10^{-4}$	$5.70 \cdot 10^{-2}$	$8.80 \cdot 10^{-3}$	$2.864 \cdot 10^{-1}$	$6.34 \cdot 10^{-2}$
	S_3	6	$2.7 \cdot 10^{-3}$	$2.3279 \cdot 10^{-4}$	$4.44 \cdot 10^{-2}$	$3.10 \cdot 10^{-3}$	$2.664 \cdot 10^{-1}$	$4.70 \cdot 10^{-2}$
	S_4	10	$6.6 \cdot 10^{-3}$	$4.7762 \cdot 10^{-4}$	$5.64 \cdot 10^{-2}$	$1.04 \cdot 10^{-2}$	$3.178 \cdot 10^{-1}$	$6.97 \cdot 10^{-2}$
	S_5	10	$6.0 \cdot 10^{-3}$	$7.1615 \cdot 10^{-4}$	$5.27 \cdot 10^{-2}$	$1.04 \cdot 10^{-2}$	$3.961 \cdot 10^{-1}$	$7.10 \cdot 10^{-2}$
	S_6	6	$2.9 \cdot 10^{-3}$	$2.0089 \cdot 10^{-4}$	$4.48 \cdot 10^{-2}$	$3.30 \cdot 10^{-3}$	$2.787 \cdot 10^{-1}$	$4.84 \cdot 10^{-2}$
	S_7	10	$6.8 \cdot 10^{-3}$	$4.4925 \cdot 10^{-4}$	$5.67 \cdot 10^{-2}$	$1.55 \cdot 10^{-2}$	$3.856 \cdot 10^{-1}$	$7.48 \cdot 10^{-2}$
	S_8	6	$2.8 \cdot 10^{-3}$	$1.8938 \cdot 10^{-4}$	$4.44 \cdot 10^{-2}$	$3.40 \cdot 10^{-3}$	$2.744 \cdot 10^{-1}$	$4.75 \cdot 10^{-2}$
	S_9	10	$8.4 \cdot 10^{-3}$	$6.2800 \cdot 10^{-4}$	$6.31 \cdot 10^{-2}$	$1.17 \cdot 10^{-2}$	$3.170 \cdot 10^{-1}$	$7.21 \cdot 10^{-2}$
	S_{10}	10	$7.0 \cdot 10^{-3}$	$5.0563 \cdot 10^{-4}$	$5.80 \cdot 10^{-2}$	$2.59 \cdot 10^{-2}$	$4.928 \cdot 10^{-1}$	$9.77 \cdot 10^{-2}$
	DE0	10	$1.9 \cdot 10^{-3}$	$1.1740 \cdot 10^{-3}$	$2.75 \cdot 10^{-2}$	$1.60 \cdot 10^{-3}$	$1.174 \cdot 10^{-1}$	$2.55 \cdot 10^{-2}$
g_8	S_1	**4**	$\mathbf{5.0706 \cdot 10^{-5}}$	$\mathbf{5.2218 \cdot 10^{-6}}$	$\mathbf{4.900 \cdot 10^{-3}}$	$\mathbf{2.2048 \cdot 10^{-4}}$	$\mathbf{7.0900 \cdot 10^{-2}}$	$\mathbf{9.500 \cdot 10^{-3}}$
	S_2	4	$2.5900 \cdot 10^{-2}$	$7.9489 \cdot 10^{-4}$	$1.012 \cdot 10^{-1}$	$4.0500 \cdot 10^{-2}$	$6.5820 \cdot 10^{-1}$	$1.306 \cdot 10^{-1}$
	S_3	4	$6.4792 \cdot 10^{-5}$	$5.5160 \cdot 10^{-6}$	$4.800 \cdot 10^{-3}$	$3.8000 \cdot 10^{-3}$	$1.9060 \cdot 10^{-1}$	$2.260 \cdot 10^{-2}$
	S_4	4	$5.7000 \cdot 10^{-2}$	$2.1000 \cdot 10^{-3}$	$1.507 \cdot 10^{-1}$	$6.6500 \cdot 10^{-2}$	$6.2290 \cdot 10^{-1}$	$1.581 \cdot 10^{-1}$
	S_5	4	$5.9300 \cdot 10^{-2}$	$2.1000 \cdot 10^{-3}$	$1.566 \cdot 10^{-1}$	$1.059 \cdot 10^{-1}$	$1.0151 \cdot 10^{0}$	$2.196 \cdot 10^{-1}$
	S_6	4	$4.8815 \cdot 10^{-5}$	$1.2173 \cdot 10^{-5}$	$4.700 \cdot 10^{-3}$	$3.7576 \cdot 10^{-4}$	$7.1000 \cdot 10^{-2}$	$9.900 \cdot 10^{-3}$
	S_7	4	$2.8800 \cdot 10^{-2}$	$4.9589 \cdot 10^{-4}$	$1.086 \cdot 10^{-1}$	$3.6900 \cdot 10^{-2}$	$5.8720 \cdot 10^{-1}$	$1.248 \cdot 10^{-1}$
	S_8	4	$1.4000 \cdot 10^{-3}$	$8.1319 \cdot 10^{-6}$	$1.030 \cdot 10^{-2}$	$2.9000 \cdot 10^{-3}$	$8.200 \cdot 10^{-2}$	$1.570 \cdot 10^{-2}$
	S_9	4	$6.1600 \cdot 10^{-2}$	$3.3000 \cdot 10^{-3}$	$1.518 \cdot 10^{-1}$	$9.7500 \cdot 10^{-2}$	$8.5280 \cdot 10^{-1}$	$1.896 \cdot 10^{-1}$
	S_{10}	4	$7.8800 \cdot 10^{-2}$	$2.6000 \cdot 10^{-3}$	$1.742 \cdot 10^{-1}$	$9.6800 \cdot 10^{-2}$	$8.2430 \cdot 10^{-1}$	$1.969 \cdot 10^{-1}$
	DE0	4	$1.8868 \cdot 10^{-5}$	$1.8800 \cdot 10^{-4}$	$3.700 \cdot 10^{-3}$	$1.4703 \cdot 10^{-5}$	$1.8800 \cdot 10^{-2}$	$3.200 \cdot 10^{-3}$
g_9	S_1	**3**	$\mathbf{2.3271 \cdot 10^{-7}}$	$\mathbf{8.5283 \cdot 10^{-6}}$	$\mathbf{2.8277 \cdot 10^{-4}}$	$\mathbf{3.0140 \cdot 10^{-7}}$	$\mathbf{1.90 \cdot 10^{-3}}$	$\mathbf{3.3595 \cdot 10^{-4}}$
	S_2	3	$9.8649 \cdot 10^{-6}$	$4.3543 \cdot 10^{-5}$	$1.9000 \cdot 10^{-3}$	$1.2589 \cdot 10^{-5}$	$1.11 \cdot 10^{-2}$	$2.2000 \cdot 10^{-3}$
	S_3	3	$2.6935 \cdot 10^{-7}$	$9.6234 \cdot 10^{-6}$	$3.1556 \cdot 10^{-4}$	$3.4114 \cdot 10^{-7}$	$2.00 \cdot 10^{-3}$	$3.6233 \cdot 10^{-4}$
	S_4	4	$2.0736 \cdot 10^{-5}$	$4.2483 \cdot 10^{-5}$	$2.6000 \cdot 10^{-3}$	$2.3813 \cdot 10^{-5}$	$1.38 \cdot 10^{-2}$	$2.9000 \cdot 10^{-3}$
	S_5	5	$2.5199 \cdot 10^{-5}$	$4.2751 \cdot 10^{-5}$	$3.1000 \cdot 10^{-3}$	$2.8449 \cdot 10^{-5}$	$1.39 \cdot 10^{-2}$	$3.3000 \cdot 10^{-3}$
	S_6	3	$2.3943 \cdot 10^{-7}$	$6.4724 \cdot 10^{-6}$	$2.8993 \cdot 10^{-4}$	$3.2458 \cdot 10^{-7}$	$2.10 \cdot 10^{-3}$	$3.5655 \cdot 10^{-4}$
	S_7	3	$9.8293 \cdot 10^{-6}$	$5.1352 \cdot 10^{-5}$	$1.9000 \cdot 10^{-3}$	$9.2539 \cdot 10^{-6}$	$9.60 \cdot 10^{-3}$	$1.8000 \cdot 10^{-3}$
	S_8	3	$2.3357 \cdot 10^{-7}$	$8.6406 \cdot 10^{-6}$	$2.7981 \cdot 10^{-4}$	$3.3171 \cdot 10^{-7}$	$2.00 \cdot 10^{-3}$	$3.5957 \cdot 10^{-4}$
	S_9	3	$3.2613 \cdot 10^{-5}$	$5.5223 \cdot 10^{-5}$	$3.3000 \cdot 10^{-3}$	$3.4828 \cdot 10^{-5}$	$1.65 \cdot 10^{-2}$	$3.4000 \cdot 10^{-3}$
	S_{10}	5	$3.1501 \cdot 10^{-5}$	$4.2401 \cdot 10^{-5}$	$3.4000 \cdot 10^{-3}$	$4.4804 \cdot 10^{-5}$	$1.91 \cdot 10^{-2}$	$4.0000 \cdot 10^{-3}$
	DE0	8	$7.6906 \cdot 10^{-6}$	$1.5100 \cdot 10^{-4}$	$2.0000 \cdot 10^{-3}$	$7.1346 \cdot 10^{-6}$	$1.02 \cdot 10^{-2}$	$1.8000 \cdot 10^{-3}$

the original function. For example, in approximating function g_1 with Strategy 1, averagely, the training error is $6.2688 \cdot 10^{-6}$, the testing error is $6.9504 \cdot 10^{-6}$, and the standard deviation during testing is $1.4 \cdot 10^{-3}$, while only three RBFs are used and the original function is in the range of $[-1.7897, 1.2077]$.

3. Strategies 1, 3, 6, and 8 worked better than other strategies in each function: (*i*) lower values of errors — training error, testing error, maximum error and *std* with a smaller number of nodes (e.g., g_1); (*ii*) lower values of errors with almost the same number of nodes (e.g., g_9); (*iii*) lower or higher values of errors with a lower number of nodes (e.g., g_6 and g_3).

4. The method in [12], short as DE0, is different from the proposed method in two ways: (*i*) it selects centres and decides weights of the networks heuristically, then uses DE for local and global tuning to search for the widths of RBFs; (*ii*) it does not include the cycling scheme. These differences results in that the proposed method has better performance: (*i*) in g_1 and

g_9, using lower than half of the nodes required by DE0, Strategies 1, 3, 6, and 8 performed better than DE0, i.e., each had a smaller error, while the rest strategies each got a higher error; (*ii*) in g_2, Strategies 1, 3, 6, and 8 used lower than half of the nodes required by DE0 but achieved results comparable to those by DE0 while and the rest strategies got worse results with a lower number of nodes than required by DE0; (*iii*) in g_4, using smaller than half of the nodes required by DE0, all strategies achieved results slightly worse than those by DE0; (*iv*) in g_3 and g_5, Strategies 1, 3, 6, and 8 achieved results close to those by DE DE0, using almost half of the nodes required by DE0, but each of the rest strategies got close or worse results with a smaller or the same number of nodes required by DE0; (*v*) in g_6 and g_7, all strategies got results comparable to those by DE0, using same-sized (Strategies 2, 4, 5, 7, 9, 10) or smaller-sized (Strategies 1, 3, 6, 8) RBF networks; (*vi*) in g_8, using the same number of nodes required by DE0, Strategies 1, 3, and 6 got results comparable to those by DE0, while each of the rest strategies got a higher error.

4.2 Comparisons with Reported Approaches

Table 5 shows more results of four functions, including comparison with three methods proposed in the literature. Esposito *et al* showed an incremental algorithm for growing RBF networks by means of an evolutionary strategy (short as EA) [5], which can achieve better performance than a greedy algorithm (short as FE) [24] and Wavelet Neural Network (short as AW) [26] in terms of net size and computation time. ϵ_f and p_{max} were set for each function based on the experimental results in [5]. It can be seen that the proposed algorithm outperformed the three methods in the tests, since:

1. Strategies 1, 3, 6, and 8 performed better than other strategies in each function, i.e., each with a smaller error while using a smaller number of nodes.

2. g_5: each strategy performed better than FE; using a lower number of nodes than required by EA, Strategies 1, 3, 6, and 8 offered results closer to those by EA than other strategies.

3. g_6: each strategy got results comparable to those by AW with smaller than half of the nodes required by AW, except that Strategy 5 got worse results with one third of the nodes; Strategies 1, 3, 6, and 8 produced results close to those by EA with lower than half of the nodes required by EA, while the rest of the strategies each produced close or worse results with a little higher than half of the nodes required by EA.

4. g_8: Strategies 1, 3, 6, and 8 performed better than EA, i.e., each with a smaller error, using a lower number of nodes than required by EA, while the rest of the strategies performed much more worse than EA.

5. g_9: Strategies 1, 2, 3, 6, 7, and 8 performed better than EA, i.e., each with a lower error, using a lower number of nodes than required by EA; Strategies 4, 5, 9, and 10 each had results close to those by EA with a smaller number of nodes than required by EA.

5. CONCLUSION

A new DE-based approach was developed and applied to training Gaussian RBF networks with variables including centres, weights, and widths of RBFs for function approximation. The choice of the optimal network parameters corresponds to the minimum Mean Square Error between the desired and actual outputs. The tests based on a set of functions and different strategies of DE demonstrated the approach. The obtained results suggest that the proposed approach is effective in optimizing the parameters of Gaussian-type RBF networks. The comparison to other incremental algorithms reported in the literature has shown that the DE-based RBF network-growing approach combined with cycling scheme performed better in terms of the lower MSE with smaller network in the tested cases, especially Strategies 1, 3, 6, and 8.

Future research should include tests with higher data dimensionalities and with less smooth data.

6. ACKNOWLEDGMENTS

The research is supported by the East Finland Graduate School in Computer Science and Engineering, and the authors would like to thank Jouni Sampo for discussion.

7. REFERENCES

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

[2] S. A. Billings and G. L. Zheng. Radial basis function networks configuration using genetic algorithms. *Neural Networks*, 8(6):877–890, 1995.

[3] A. G. Bors and I. Pitas. Median radial basis function neural network. *IEEE Trans. on Neural Network*, 7(6):1351–1364, 1996.

[4] D. S. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1998.

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

[6] C. F. Fung, S. A. Billings, and W. Luo. On-line supervised adaptive training using radial basis function networks. *Neural Networks*, 9(9):1597–1617, 1996.

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

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

[9] A. Leonardis and H. Bischof. An efficient mdl-based construction of rbf networks. *Neural Networks*, 11:963–973, 1998.

[10] J. Liu, S. Kukkonen, and J. Lampinen. Function approximation with pruned radial basis function networks using a de-based algorithm: an initial investigation. In *Proc. of* 10^{th} *Int'l Conf. on Soft Computing (MENDEL)*, pages 139–144. Brno, Czech Republic, June 16-18, 2004.

Table 5: Comparisons of experimental results

g_n	Method	Training error	Testing error	Max. error	N	p
g_5	S_1	$\mathbf{8.10 \cdot 10^{-2}}$	$\mathbf{1.67 \cdot 10^{-1}}$	**1.177**	40	**4**
	S_2	$1.59 \cdot 10^{-2}$	$1.20 \cdot 10^{-1}$	1.046	40	5
	S_3	$\mathbf{2.00 \cdot 10^{-3}}$	$\mathbf{7.34 \cdot 10^{-2}}$	**0.607**	40	**5**
	S_4	$1.05 \cdot 10^{-1}$	$2.17 \cdot 10^{-1}$	1.412	40	4
	S_5	$8.86 \cdot 10^{-2}$	$2.93 \cdot 10^{-1}$	1.457	40	4
	S_6	$8.86 \cdot 10^{-2}$	$1.69 \cdot 10^{-1}$	1.219	40	4
	S_7	$9.17 \cdot 10^{-2}$	$2.25 \cdot 10^{-1}$	1.485	40	4
	S_8	$2.00 \cdot 10^{-3}$	$5.84 \cdot 10^{-2}$	0.719	40	5
	S_9	$7.87 \cdot 10^{-2}$	$2.05 \cdot 10^{-1}$	1.368	40	4
	S_{10}	$1.54 \cdot 10^{-2}$	$1.71 \cdot 10^{-1}$	1.238	40	6
	FF	$2 \cdot 10^{-1}$			40	6
	EA	$5.00 \cdot 10^{-3}$	$4.00 \cdot 10^{-3}$	0.18	40	6
g_6	S_1	$\mathbf{3.18 \cdot 10^{-6}}$	$\mathbf{1.33 \cdot 10^{-4}}$	**0.047**	100	**7**
	S_2	$7.22 \cdot 10^{-6}$	$1.39 \cdot 10^{-4}$	0.055	100	13
	S_3	$3.00 \cdot 10^{-6}$	$2.22 \cdot 10^{-4}$	0.071	100	10
	S_4	$3.73 \cdot 10^{-6}$	$1.30 \cdot 10^{-4}$	0.061	100	17
	S_5	$1.35 \cdot 10^{-5}$	$4.20 \cdot 10^{-3}$	0.134	100	15
	S_6	$\mathbf{3.77 \cdot 10^{-6}}$	$\mathbf{1.76 \cdot 10^{-4}}$	**0.063**	100	**6**
	S_7	$2.81 \cdot 10^{-6}$	$2.24 \cdot 10^{-4}$	0.076	100	16
	S_8	$4.09 \cdot 10^{-6}$	$1.16 \cdot 10^{-4}$	0.050	100	8
	S_9	$4.65 \cdot 10^{-6}$	$2.34 \cdot 10^{-4}$	0.056	100	17
	S_{10}	$7.49 \cdot 10^{-6}$	$1.37 \cdot 10^{-4}$	0.064	100	16
	AW	$1.6 \cdot 10^{-6}$			100	45
	EA	$1.1 \cdot 10^{-6}$	$4.2 \cdot 10^{-6}$		100	25
g_8	S_1	$\mathbf{3.64 \cdot 10^{-5}}$	$\mathbf{3.98 \cdot 10^{-5}}$	**0.025**	400	**4**
	S_2	$1.23 \cdot 10^{-2}$	$1.29 \cdot 10^{-2}$	0.293	400	6
	S_3	$7.63 \cdot 10^{-5}$	$8.61 \cdot 10^{-5}$	0.037	400	4
	S_4	$1.78 \cdot 10^{-2}$	$1.88 \cdot 10^{-2}$	0.409	400	6
	S_5	$2.77 \cdot 10^{-2}$	$2.90 \cdot 10^{-2}$	0.454	400	6
	S_6	$3.21 \cdot 10^{-5}$	$3.87 \cdot 10^{-5}$	0.028	400	4
	S_7	$9.80 \cdot 10^{-3}$	$1.09 \cdot 10^{-2}$	0.389	400	6
	S_8	$6.50 \cdot 10^{-5}$	$7.02 \cdot 10^{-5}$	0.037	400	4
	S_9	$1.57 \cdot 10^{-2}$	$1.62 \cdot 10^{-2}$	0.430	400	6
	S_{10}	$3.46 \cdot 10^{-2}$	$3.47 \cdot 10^{-2}$	0.569	400	6
	EA	$2.2 \cdot 10^{-4}$	$2.4 \cdot 10^{-4}$		400	6
g_9	S_1	$\mathbf{2.74 \cdot 10^{-7}}$	$\mathbf{2.80 \cdot 10^{-7}}$	**0.002**	1000	**3**
	S_2	$9.38 \cdot 10^{-6}$	$9.39 \cdot 10^{-6}$	0.010	1000	3
	S_3	$2.76 \cdot 10^{-7}$	$2.83 \cdot 10^{-7}$	0.003	1000	3
	S_4	$1.74 \cdot 10^{-5}$	$1.79 \cdot 10^{-5}$	0.012	1000	4
	S_5	$3.60 \cdot 10^{-5}$	$3.66 \cdot 10^{-5}$	0.015	1000	5
	S_6	$2.66 \cdot 10^{-7}$	$2.71 \cdot 10^{-7}$	0.003	1000	3
	S_7	$9.27 \cdot 10^{-6}$	$9.53 \cdot 10^{-6}$	0.012	1000	3
	S_8	$2.81 \cdot 10^{-7}$	$2.87 \cdot 10^{-7}$	0.003	1000	3
	S_9	$3.29 \cdot 10^{-5}$	$3.37 \cdot 10^{-5}$	0.019	1000	3
	S_{10}	$3.64 \cdot 10^{-5}$	$3.70 \cdot 10^{-5}$	0.016	1000	5
	EA		$1.4 \cdot 10^{-5}$		1000	6

Note: "S_n" stands for Strategy n; N is the size of training set and testing set; p is the smallest value in 30 runs with the same strategy; errors are average values of 30 runs.

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

[12] J. Liu and J. Lampinen. Growing rbf networks for function approxiamtion by a de-based method. In *Proc. of 1^{st} Int'l Symposium on Computational and Information Science*, pages 399–406. Shanghai, China, Dec 16-18, 2004.

[13] J. Liu and J. Lampinen. A fuzzy differential evolution algorithm. *Soft Computing*, (to appear).

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

[15] 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*, 5:595–603, 1992.

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

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

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

[19] K. Price, R. Storn, and J. A. Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Berlin/Heidelberg, Germany, (in print).

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

[21] R. Storn and K. Price. On the usage of differential evolution for function optimization. In *Proc. of 1996 Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS)*, pages 519–523. Berkeley, CA USA, June 19-22, 1996.

[22] 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.

[23] 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.

[24] V. Vinod and S. Ghose. Growing nonuniform feedforward networks for continuous mapping. *Neural computing*, 10:55–69, 1996.

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

[26] F. Yong and T. W. S. Chow. Neural network adaptive wavelets for function approximation. In *Proc. of European Symposium on Artificial Neural Networks*, pages 345–350. Bruges, Belgium, D-Facto public., Apr 16-18, 1997.

[27] Q. Zhu, Y. Cai, and L. Liu. A global learning algorithm for a rbf network. *Neural Networks*, 12:527–540, 1999.