Applied Soft Computing 49 (2016) 485–497

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Cooperative learning for radial basis function networks using particle swarm optimization

Alex Alexandridis [a,*], Eva Chondrodima [a,b], Haralambos Sarimveis [b]

[a] Department of Electronic Engineering, Technological Educational Institute of Athens Agiou Spiridonos, Aigaleo 12210, Greece
[b] School of Chemical Engineering, National Technical University of Athens, 9, Heroon Polytechniou Str., 15780 Zografou, Greece

ARTICLE INFO

Article history:
Received 8 November 2015
Accepted 18 August 2016
Available online 22 August 2016

Keywords:
Cooperative learning
Cooperative swarms
Neural networks
Particle swarm optimization
Radial basis function

ABSTRACT

This paper presents a new evolutionary cooperative learning scheme, able to solve function approximation and classification problems with improved accuracy and generalization capabilities. The proposed method optimizes the construction of radial basis function (RBF) networks, based on a cooperative particle swarm optimization (CPSO) framework. It allows for using variable-width basis functions, which increase the flexibility of the produced models, while performing full network optimization by concurrently determining the rest of the RBF parameters, namely center locations, synaptic weights and network size. To avoid the excessive number of design variables, which hinders the optimization task, a compact representation scheme is introduced, using two distinct swarms. The first swarm applies the non-symmetric fuzzy means algorithm to calculate the network structure and RBF kernel center coordinates, while the second encodes the basis function widths by introducing a modified neighbor coverage heuristic. The two swarms work together in a cooperative way, by exchanging information towards discovering improved RBF network configurations, whereas a suitably tailored reset operation is incorporated to help avoid stagnation. The superiority of the proposed scheme is illustrated through implementation in a wide range of benchmark problems, and comparison with alternative approaches.

1. Introduction

Neural network (NN) training refers to the procedure of determining the network parameters, so as to adjust the NN behavior to a set of given prototypes. The training task is naturally formulated as an optimization problem, where the design variables are the network parameters and the objective function is a cost function, expressing the disparity between the predictions that the network casts, and the target data.

Early attempts to tackle the NN training problem were based on conventional optimization techniques; gradient descent forms the basis of the pioneering algorithm of back-propagation and its many variants [1]. Later-on, gradient descent was hybridized with Newton's method to produce the Levenberg-Marquardt algorithm, a very popular technique in NN training [2]. Similar methodologies have been extensively used to accomplish the NN training task, mainly due to their simplicity; however, it soon became clear that there is much room for improvement in the solutions they provide. This is due to multiple undesirable characteristics associated with the objective function, including multimodality, possibly trapping the algorithm in non-satisfactory local optima, and noise contamination, which is typically present in the training examples. The large number of NN parameters that need to be identified, explosively increases the search space size, thus magnifying the negative effect of multimodality. These factors make the discovery of the global optimal solution rather improbable and even hinder the task of obtaining a satisfactory sub-optimal set of NN parameters. Furthermore, non-differentiability with respect to some of the NN parameters, e.g. the size of the hidden layer(s), prohibits the use of derivative-based methods, and often leads to tedious trial-and-error procedures to resolve the respective part of the training problem.

Not surprisingly, techniques based on evolutionary computation (EC) were conscripted in NN training to obviate the aforementioned difficulties. Preliminary approaches [3] were focused on the first NN architecture to appear, i.e. multilayer feedforward (MFF) networks. In these early works, EC techniques replaced the gradient descent methods in a straightforward manner, using populations of individuals which encoded the weights and biases of the network. More elaborate EC-based techniques for weight adjustment were also proposed, including differential

* Corresponding author.
E-mail address: alexx@teiath.gr (A. Alexandridis).

http://dx.doi.org/10.1016/j.asoc.2016.08.032
1568-4946/

evolution (DE), and a hybrid method combining DE and unscented Kalman filter [4].

Though such approaches were found to better cope with the multimodal objective function, they still left out of the optimization problem the network structure, i.e. the numbers of hidden layers, and neurons per layer. Naturally, the next step was to suitably tailor EC techniques to incorporate network structure selection mechanisms to the optimization framework. New methods for evolving NN architectures, with applications in time series forecasting have been proposed in [5,6]. A novel technique based on particle swarm optimization (PSO) for automatically designing multilayered NNs by evolving to the optimal network configuration has been introduced in [7]. A multiobjective PSO algorithm with the ability to adapt the NN topology was successfully applied for pixel classification in satellite imagery [8]. A modified bat algorithm with a new solution representation for both optimizing the NN weights and structure has been proposed in [9]. Furthermore, the use of EC for accomplishing network training and optimization has been broadened, to tackle the challenges imposed by different learning schemes. Such approaches include the development of evolutionary fuzzy ARTMAP NNs to deal with the classification of imbalanced data set problems, with application in semiconductor manufacturing operations [10], and two novel approaches for building sparse least square support vector machines based on genetic algorithms [11].

Radial basis function (RBF) networks [12] offer a simpler alternative to most of the NN architectures, comprising only one hidden layer, with a linear connection to the output node. Though these properties help to simplify – and possibly expedite – the training procedure of RBF networks, the associated optimization problem is still rife with the previously mentioned undesirable characteristics, which call out for non-conventional optimization schemes. EC-based approaches for training RBF networks include, among others, genetic algorithms for optimizing the network structure and RBF kernel center coordinates [13]; a method combining evolutionary and gradient-based learning to estimate the architecture, weights and node topology of generalized RBF classifiers [14]; a bee-inspired algorithm to automatically select the number, location and widths of basis functions [15]; and a methodology for building hybrid RBF NNs, where a genetic algorithm is exploited to optimize the essential design parameters of the model [16].

Special focus is also given on applying PSO to solve the RBF training problem; an improved RBF learning approach for real time applications based on PSO is proposed in [17], whereas in [18,19] the authors use PSO to construct polynomial RBF networks. A new method for developing RBF regression models using PSO, in conjunction with the non-symmetric fuzzy means (NSFM) algorithm is presented in [20]. The particular technique offers important advantages, including improved prediction accuracies, faster computational times and parsimonious network structures; on the other hand, its use is restricted to RBFs with fixed-width functions and, therefore, it cannot accommodate for some popular RBF selections like the Gaussian function. The use of fixed-width RBFs also curtails the flexibility of the multi-dimensional approximation surface constructed by the RBF network, possibly diminishing the accuracy of the produced model and/or requiring more kernels to achieve satisfactory results.

In this paper, we remove the fixed-width requirement and present a new cooperative learning EC framework for fully optimizing RBF networks, which allows to use any activation function satisfying the radial basis hypothesis. The proposed method adopts a cooperative approach to PSO, which utilizes multiple swarms to optimize different components of the solution vector [21]. Cooperative EC techniques have recently been used with great success in solving complex optimization problems [22–24]; in addition, they are particularly well-suited for NN optimization [25–28], as they approach the modularity of the training problem in a natural way. Nevertheless, there are surprisingly few publications in the subject of training RBF networks with cooperative techniques [29,30]. The proposed approach makes use of two cooperative swarms, controlling different sets of parameters of the RBF network, but ultimately working together towards training networks with improved properties. Contrary to most of the existing EC-based methods for full network optimization, we introduce a more compact encoding of the optimization problem with less design variables, which helps the adopted EC technique to discover improved solutions, ultimately leading to models with higher approximation accuracies. This is achieved by introducing a modification to the p-nearest neighbor technique [12], so that it can be used to encode the RBF widths within the EC framework. Furthermore, in contrast to [20], we extend the applicability of the produced RBF optimizer to regression and classification problems alike. The proposed method is compared to the approach followed in [20], but also to two additional methodologies through a series of experiments; results verify that the new scheme is clearly superior in many aspects.

The rest of this work is organized as follows: In the second section, we give a short introduction to the RBF architecture, discuss RBF network training and model selection, and briefly present the NSFM algorithm. Section 3 starts by describing the incorporation of PSO to form the PSO-NSFM technique, continues with the modifications needed for using variable-width basis functions and ultimately presents the cooperative PSO framework for RBF training. A range of experiments in regression and classification datasets, evaluating the proposed approach and comparing it to different methods, is presented in Section 4. The paper concludes by summarizing the merits of the proposed approach.

2. RBF networks

The neurons of a typical RBF network, which can be used either for function approximation or classification tasks, are grouped in three layers. The input layer has the same dimensionality N with the input space and performs no calculations but only distributes the input variables towards the hidden layer. The latter is comprised of L units, also known as neurons or RBF kernels; each neuron is characterized by a center vector $\hat{\mathbf{u}}_l$. A nonlinear transformation takes place inside the hidden layer, creating a mapping between the input space and the space of hidden neurons. When the network is presented with an input vector $u(k)$, each neuron l triggers a different activity $\mu_l(\mathbf{u},(k))$, calculated as the Euclidean norm between the input vector and the respective center.

Neuron activity is then given as input to a function with radial symmetry, hence the name RBF networks. In order to demonstrate the use of variable-width basis functions, we employ the Gaussian function, given in (1); however the method can be trivially modified to accommodate for any other variable-width basis function, or even basis functions with more adjustable parameters.

$$g_l(\mu) = \exp\left(-\frac{{\mu_l}^2}{{\sigma_l}^2}\right), \quad l = 1, \ldots, L \tag{1}$$

where σ_l is the width of the lth kernel. The responses of the hidden neurons $z(k)$ become:

$$z(k) = \left[g(\mu_1(\mathbf{u}(k))), g(\mu_2(\mathbf{u}(k))), \ldots, g(\mu_L(\mathbf{u}(k)))\right] \tag{2}$$

The output layer consists of M summation units, where M is the number of output variables. The numerical output $\hat{y}_m(k)$ for each summation unit is produced by linearly combining the hidden neuron responses:

$$\hat{y}_m(k) = \mathbf{z}(k) \cdot \mathbf{w}_m = \sum_{l=1}^{L} w_{l,m} g(\mu_l(\mathbf{u}(k))) \tag{3}$$

where $\mathbf{w}_m = \left[w_{1,m}, w_{2,m}, \ldots, w_{L,m}\right]^T$ is the synaptic weight vector corresponding to output m. When the RBF network is used as a classifier, the prediction for the output class $\mathscr{C}(k)$ corresponds to the summation unit triggering the maximum numerical output [1]:

$$\mathscr{C}\left(k\right) = \arg\max_m \hat{y}_m\left(k\right), \quad m = 1,2,\ldots,M \tag{4}$$

2.1. RBF network training and model selection

Training of an RBF network corresponds to finding optimal values for the following network-related parameters:

- Network size (number of kernels L)
- Kernel center coordinates $\hat{\mathbf{u}}_1, \hat{\mathbf{u}}_2, \ldots, \hat{\mathbf{u}}_L$
- Kernel widths $\boldsymbol{\sigma} = [\sigma_1, \sigma_2, \ldots, \sigma_L]$
- Synaptic weights $W = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_M,]$

For a fixed number of kernels L, the procedure of RBF network training corresponds to the solution of the following optimization problem:

$$\min_{\hat{\mathbf{u}}_1,\hat{\mathbf{u}}_1,\ldots,\hat{\mathbf{u}}_L,\boldsymbol{\sigma},\mathbf{W}} f\left(\hat{\mathbf{u}}_1, \hat{\mathbf{u}}_2, \ldots, \hat{\mathbf{u}}_L, \boldsymbol{\sigma}, \mathbf{W}, \mathbf{U}_{\text{train}}, \mathbf{Y}_{\text{train}}\right) \tag{5}$$

where f is a cost function, providing a metric for the deviation between the network predictions and the actual values, and $\mathbf{U}_{\text{train}}$ and $\mathbf{Y}_{\text{train}}$ are matrices containing the available input and output training data, respectively. Based on the above, the total number of scalar design variables equals to $L(N+M+1)$. This number can grow very large; for instance, for a small-sized training problem with five input variables, one output variable and 15 hidden neurons, the total of scalar design variables already exceeds 100, whereas even for medium-sized problems, it can easily grow to the order of tens of thousands. Keeping in mind that the objective function is highly multi-modal, it is easily understood that locating the global optimum is, in most cases, quite improbable; in fact, even the calculation of a satisfying sub-optimal solution can be rather cumbersome.

The overwhelming complexity of the optimization task can be tackled by decomposing the training problem in separate phases. Exploiting the linear relationship between hidden and output layers, linear regression (LR) in matrix form can be applied for calculating the weights:

$$\boldsymbol{W}^T = \boldsymbol{Y}_{\text{train}}{}^T \cdot \boldsymbol{Z} \cdot \left(\mathbf{Z}^T \cdot \boldsymbol{Z}\right)^{-1} \tag{6}$$

where $\mathbf{Z} = [\mathbf{z}(1), \mathbf{z}(2), \ldots, \mathbf{z}(K)]^T$ and K is the number of training examples. The advantage is that for given values of the centers and widths, a global optimum is guaranteed, as far as the weights are concerned. Unfortunately, calculation of the rest of the network parameters remains a difficult task with no optimality guaranteed.

The training procedure is typically followed by model validation and eventually model selection, i.e. selection of the number of RBF kernels. A large number of kernels would produce improved solutions to (5); however, it could lead to overfitting, i.e. poor performance when the model is applied on new data that have not been used during the training procedure. Validation refers to measuring the generalization ability of the model, i.e. its predictive accuracy when applied on a new set of input-output data $\left\{\mathbf{U}_{\text{val}}, \mathbf{Y}_{\text{val}}\right\}$. The model that performs better in the validation procedure is finally selected. Notice that selection of the number of kernels through this additional step further increases the size of the search space.

2.2. The NSFM algorithm

The calculation of RBF kernel center coordinates is usually based on non-supervised clustering techniques applied on the input space. A typical method is the k-means algorithm [31], which, however, suffers from multiple disadvantages, including long convergence times and lack of an integrated mechanism for determining the network structure. The fuzzy means (FM) algorithm [32,33] was introduced as an alternative to k-means, for selecting the network size and center coordinates. Recently, a non-symmetric variant of the FM algorithm was presented, namely the NSFM algorithm [34], offering increased accuracy over the original approach. A brief discussion about the NSFM algorithm is given below; more details can be found in the original publication.

Following the non-symmetric partition concept, the domains of all input variables are partitioned into s_i one-dimensional (1-D) triangular fuzzy sets, where $i = 1,2,\ldots, N$. Each fuzzy set is characterized by a center element $a_{i,j}$ and a width $\delta\alpha_i$. The next step is to define a set of multi-dimensional fuzzy subspaces, where each subspace $\mathbf{A}^l$, $l = 1,\ldots, S$ comprises a single 1-D set for each input variable. The total number of multi-dimensional fuzzy subspaces S is given by:

$$S = \prod_{i=1}^{N} s_i \tag{7}$$

The objective of the algorithm is to suitably select the RBF network kernel centers among the centers of the fuzzy subspaces, by extracting only a subset of them; this subset should be small, but sufficient to describe the distribution of data in the input space. The selection is based on a metric $d_r^l(\mathbf{u}(k))$, expressing the distance between the center belonging to fuzzy subspace $\mathbf{A}^1$ and input data vector $\mathbf{u}(k)$. In the non-symmetric case, d_r^l is given by the following elliptical-shaped function:

$$d_r^l(\mathbf{u}(\mathrm{k})) = \sqrt{\left(\frac{\sum_{i=1}^{N}\left(a_{i,j_i}^l - u_i(k)\right)^2}{N\left(\delta a_i\right)^2}\right)} \tag{8}$$

The algorithm finally selects a subset of centers, so that the multi-dimensional ellipses defined by the corresponding fuzzy subspaces cover all the training vectors. Following a non-iterative procedure, which requires a single pass of the training data, the algorithm can accomplish the center calculation stage efficiently, even for large datasets.

3. Methodology

3.1. PSO-NSFM

When using the NSFM algorithm, the problem of optimizing the network size and kernel center coordinates, boils down to discovering the optimal fuzzy partition for the input space. This is a significant improvement in terms of facilitating the RBF training optimization task, as the total of design variables is reduced to N, i.e. the dimensionality of input space. The PSO-NSFM algorithm [20] uses PSO for exploring the space of solutions and determining the optimal non-symmetric partition. In this case, each individual is encoded to reflect the input space partitioning. To be more specific, the elements of particle $s_i(t)$ represent the number of fuzzy sets assigned to each dimension, at iteration t:

$$\mathbf{s}_i(t) = \left[s_1(t), \quad s_2(t), \quad \ldots, \quad s_N(t)\right]^T, i = 1, \ldots, P \tag{9}$$

where P is the swarm population. A velocity vector $\mathbf{v}_i(t)$ is used at each iteration t to update the particle positions. Velocities are calculated taking into account a cognitive and a social term,

which are based on the distances of each individual particle position to its personal best position and the global best position achieved by the swarm, respectively. Velocity vector elements are bounded by a velocity clamping mechanism, employed to control the exploration-exploitation trade-off.

3.2. Incorporating variable width

As it incorporates no mechanism for optimizing the widths of the basis functions, the PSO-NSFM algorithm can work only with fixed-width basis functions, such as the thin-plate-spline (TPS). This limitation deprives the methodology from additional flexibility that could potentially help to adjust the multi-dimensional surface constructed by the RBF network, so as to better approximate the training examples.

A simplistic approach in order to optimize the basis function widths is to directly add them as extra dimensions in an extended particle $\mathbf{\Lambda}_i$ of the form:

$$\mathbf{\Lambda}_1(t) = \left[\underbrace{S_1(t), \ldots, S_N(t)}_{\text{Fuzzy partition}}, \underbrace{\sigma_1(t), \ldots, \sigma_{L_i(t)}(t)}_{\text{RBF widths}} \right]^T \tag{10}$$

where $L_i(t)$ is the number of kernels selected by NSFM for the RBF network corresponding to the i-th particle, at iteration t. Notice that such an implementation could lead to having particles of different dimension in the same population, but also to a potential change in the size of each individual particle per iteration. Traditional PSO-based techniques cannot handle such situations, but even if an appropriate modification was improvised, the number of design variables in the resulting optimization problem would be considerably increased.

In order to obviate this obstacle, we resort to a heuristic method known as the p-nearest neighbor approach [12], modifying it so that it can be used in an EC context. The original p-nearest neighbor technique selects the width of each basis function $\sigma_j(t)$ as the root-mean squared distance to its p nearest neighbors, using the following equation:

$$\sigma_j(t) = \sqrt{\frac{1}{p}\sum_{k=1}^{p} \|\hat{\mathbf{u}}_k - \hat{\mathbf{u}}_j\|^2}, j = 1, \ldots, L_i(t) \tag{11}$$

where $\hat{\mathbf{u}}_k$ are the p nearest centers to center $\hat{\mathbf{u}}_j$. The adoption of this heuristic allows us to substitute the widths of all individual RBF kernels in the particle, with a single parameter; thus, not only is it possible to maintain constant dimensionality per particle and per iteration, but also the number of extra design variables for adjusting the widths is reduced to only one.

However, the selection of the appropriate parameter to be inserted as the additional dimension in each particle for controlling the RBF widths requires special attention. Though it may seem intuitive to directly use the number of nearest neighbors p, such a selection is not appropriate in all cases. The reason is that p characterizes the RBF widths only indirectly, through the nearest neighbors covered. The resulting widths though, also depend on the number of kernels; this means that the same value for p could have a profoundly diverse effect in the context of a different number of kernels, which could result from a different fuzzy partitioning. This is important, because, as the PSO evolutionary procedure unfolds, the number of kernels corresponding to the network generated by each given particle may vary significantly, thus preventing the algorithm from properly exchanging information between different iterations. For example, velocity calculation requires comparing the current particle positions, to the best personal positions, which were probably calculated in a previous iteration. If the number of kernels happens to vary significantly between these two iterations, then the value of p corresponding to the best particle positions is out of context in relation to the current particle positions. A visual example for a 1-D input space is given in Fig. 1, where it can be seen that a value of $p = 2$ allocates considerably wider Gaussians in the case of 3 RBF kernels, compared to the case of 6 RBF kernels.

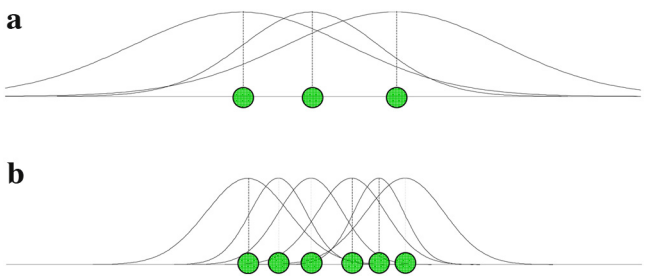

Fig. 1. Width allocation for $p = 2$ in a 1-D input space containing: (a) 3 RBF kernels, and (b) 6 RBF kernels.

In order to avoid the dependence of the widths to the context of the number of kernels, and thus the fuzzy partitioning, we define the neighbor coverage percentage κ_i for each particle, calculated as follows:

$$\kappa_i(t) = \frac{p_i(t)}{L_i(t)} \cdot 100 \tag{12}$$

Normalizing the number of nearest neighbors p_i, with the total of RBF kernels per particle L_i, creates a parameter which can be used for controlling the RBF widths, while at the same time allowing to exchange information between particles with a different number of kernels. Thus, the structure of the particles now becomes:

$$\mathbf{\Lambda}_i(t) = \left[\underbrace{s_1(t), \quad \ldots, \quad s_N(t),}_{\text{Fuzzy partition}} \quad \underbrace{\kappa_i(t)}_{\text{neighbor coverage}} \right]^T \tag{13}$$

3.3. Cooperative PSO for full RBF network optimization

The particle structure used by the PSO-NSFM approach could be easily modified to incorporate particles of the form described in (13), thus achieving concurrent optimization of center and widths, while maintaining a relatively low number of design variables, namely $N+1$. However, one can discern two distinct parts in the particle, which control a different attribute of the produced RBF network; the first part, containing the fuzzy partition, controls the kernel center calculation, while the second part, containing the neighbor coverage, controls the Gaussian width selection. This remark provides the motivation for breaking down the particles into their constituent parts and utilizing two separate swarms, one containing the fuzzy partition information and the other containing the neighbor coverage information. The two swarms can then evolve separately, improving their performance through cooperation.

Improving the optimization procedure by using cooperation between agents has a strong intuitive appeal and has been used successfully in genetic algorithm techniques [35], separating the solution vector to multiple chromosomes which belong to different populations. A similar concept has also been used in PSO context, where the solution vector is split to a number of particles belonging to different swarms, which evolve autonomously, but exchange information during the process. The first variant of the cooperative PSO algorithm (CPSO-S) [36] decomposes a solution vector of N_d components to exactly N_d 1-D particles. A newest approach, known as CPSO-S_K [21], allows for the solution vector to be split in K parts,

where $K \leq N_d$. It is possible that each one of the produced parts has a different dimensionality.

In this work, the CPSO-S_K algorithm is suitably adapted to tackle the problem of concurrent optimization of centers and widths in a cooperative learning framework for RBF network training. Based on the existence of two distinct parts in the formulation of the solution vector, as shown in (13), the utilization of two swarms comes as a natural choice. The particles of the first swarm, encoding the fuzzy partition are denoted as $P_1.\mathbf{\Lambda}_i$, whereas particles belonging to the second swarm, are denoted as $P_2.\mathbf{\Lambda}_i$:

$$P_1.\mathbf{\Lambda}_i(t) = \left[s_1(t), \quad s_2(t), \quad \ldots, \quad s_N(t) \right]^T \tag{14}$$

$$P_2.\mathbf{\Lambda}_i(t) = \kappa_i(t) \tag{15}$$

It should be noted that it is feasible to further decompose $P_1.\mathbf{\Lambda}_i$ to 1-D particles, each one containing only the number of fuzzy sets for a single input space dimension. However, it is possible that the numbers of fuzzy sets s_i are correlated for some, or even all the input variables; in this case, the independent changes made by the different swarms would have a detrimental effect for the correlated variables.

For each swarm, the particles are updated according to:

$$P_k.\mathbf{\Lambda}_i(t+1) = P_k.\mathbf{\Lambda}_i(t) + P_k.\mathbf{v}_i(t+1), k = 1,2 \tag{16}$$

The elements of the first swarm velocity vector $P_1.\mathbf{v}_i(t)$ are calculated as follows:

$$P_1.v_{ij}(t+1) = \text{round}\left\{ wP_1.v_{ij}(t) + c_1 r_{1j}(t)\left[P_1.y_{ij}(t) - P_1.s_{ij}(t)\right] + c_2 r_{2j}(t)\left[P_1.\hat{y}_j(t) - P_1.s_{ij}(t)\right] \right\} \tag{17}$$

where:

$P_1.v_{ij}(t)$ stands for particle velocity $i = 1, \ldots, P$, in dimension $j = 1, \ldots, N$, at iteration t,

$P_1.s_{ij}(t)$ stands for particle position $i = 1, \ldots, P$, in dimension $j = 1, \ldots, N$, at iteration t,

$P_1.y_{ij}(t)$ stands for the personal best position of particle $i = 1, \ldots, P$, in dimension $j = 1, \ldots, N$, at iteration t,

$P_1.\hat{y}_j(t)$ stands for the global best position achieved by the particles of the first swarm in dimension $j = 1, \ldots, N$, at iteration t,

w is the inertia weight,

c_1 and c_2 are acceleration constants,

$r_{1j}(t)$ and $r_{2j}(t)$ are sampled randomly within the range [0 1], following a uniform distribution.

As far as the second swarm is concerned, the corresponding velocities $P_2.v_i(t)$ are updated using:

$$P_2.v_i(t+1) = wP_2.v_i(t) + c_1 r_1(t)[P_2.y_i(t) - P_2.\kappa_i(t)] + c_2 r_2(t)\left[P_2.\hat{y}(t) - P_2.\kappa_i(t)\right] \tag{18}$$

where:

$P_2.v_i(t)$ stands for particle velocity $i = 1, \ldots, P$, at iteration t,

$P_2.\kappa_i(t)$ stands for particle position $i = 1, \ldots, P$, at iteration t,

$P_2.y_i(t)$ stands for the personal best position achieved by particle $i = 1, \ldots, P$, at iteration t,

$P_2.\hat{y}(t)$ stands for the global best position achieved by the particles of the second swarm, at iteration t,

The exploration-exploitation trade-off in each swarm is controlled by a velocity clamping constant, bounding the elements of the velocity vectors between predefined values:

$$P_k.v_{ij}(t+1) = \begin{cases} P_k.v_{ij}(t+1), \text{ if} \| P_k.v_{ij}(t+1) \| < P_k.V_{\max} \\ \pm P_k.V_{\max}, \text{ otherwise} \end{cases} \tag{19}$$

where $P_k.V_{\max}$ is the velocity clamping constant, which can be different for each swarm.

Although each swarm controls a distinctive part of the RBF optimization procedure via the NSFM algorithm and the nearest neighbor coverage (NNC) heuristic, respectively, information from both swarms is needed in order to fully train an RBF network and ultimately calculate its fitness value. Thus, a context vector is required to provide suitable context for evaluating the individuals from each swarm. In this work we adopt the scheme proposed in [21], where the context vector is built by concatenating each particle from one swarm, with the global best particle from the other. This is accomplished by defining the network RBF $\left(P_k.\mathbf{\Lambda}_i, P_{(2k)^{2-k}}.\hat{y}\right)$, $k = 1, 2$, as an RBF network to be trained using the particle $\mathbf{\Lambda}_i$, $i = 1, \ldots, P$, from one swarm and the global best particle $\hat{\mathbf{y}}$ from the other swarm. Thus, in order to calculate the fitness of all the particles in swarm P_k, P different RBF networks are trained by alternating between the swarm particles $P_k.\mathbf{\Lambda}_i$, while invariably employing the global best particle of the complementary swarm $P_{(2k)^{2-k}}.\hat{y}$.

After fully training the RBF network, the corresponding fitness value is estimated by applying a suitably chosen error-related criterion to the validation set. Due to the different nature of the function approximation and classification problems, we distinguish two criteria, depending on the task at hand; in this work we adopt the Root Mean Square Error (RMSE) for function approximation problems and the Matthews correlation coefficient (MCC) for classification problems [37]. MCC is a metric designed for assessing classification performance; contrary to the standard accuracy% metric, which only takes into account the overall success rate of the classifier, MCC uses the whole confusion matrix, including the successes, as well as failures, per individual class. Thus, it is well suited for assessing classifier performance, even for datasets with imbalanced class distribution. MCC is calculated as follows:

$$\text{MCC} = \frac{\sum_{k,l,m=1}^{M} C_{kk}C_{ml} - C_{lk}C_{km}}{\sqrt{\sum_{k=1}^{M}\left(\sum_{l=1}^{M} C_{lk}\right)\left(\sum_{\substack{f,g=1 \\ f \neq k}}^{M} C_{gf}\right)}\sqrt{\sum_{k=1}^{M}\left(\sum_{l=1}^{M} C_{kl}\right)\left(\sum_{\substack{f,g=1 \\ f \neq k}}^{M} C_{fg}\right)}} \tag{20}$$

where C_{ij} is the number of elements of class i that have been assigned to class j. MCC lives in the range $[-1\ 1]$, 1 indicating perfect classification. The proposed method could also be trivially adapted to incorporate alternative error-based criteria.

The described context for evaluating the particles belonging to both swarms presents two important advantages with respect to the RBF optimization procedure. First, the fitness function is evaluated after each distinctive part of the solution vector is updated, as opposed to standard PSO where fitness is calculated only after updating all components; this results in finer-grained credit assignment, avoiding situations where an update could improve one part of the solution vector, but possibly impair another.

The second advantage is related to the increased amount of combinations of different individuals from different swarms, which boost the solution diversity. On the other hand, higher diversity comes at the cost of increased computational cost, as the number of fitness function evaluations per iteration is essentially doubled. In RBF network training, this could be a serious drawback, as each training cycle may carry significant computational burden, depend-

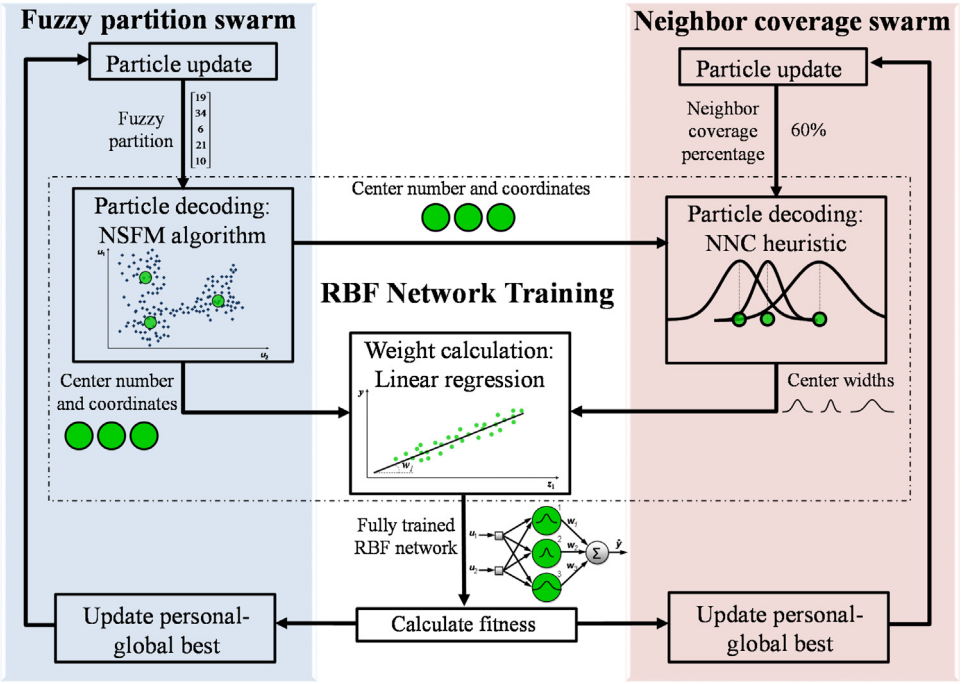

Fig. 2. A schematic overview of the two cooperative swarms, working towards optimizing RBF network models.

ing on the sizes of the network and the dataset. Nevertheless, the proposed evaluation context can help to alleviate a significant part of the imposed computational load, by exchanging additional information between the swarms. The most computationally expensive part of the training procedure is the calculation of RBF centers, due to the large number of distance calculations involved [33]. By setting the fuzzy partition swarm to run always first and then passing the RBF centers of the global best particle to the neighbor coverage swarm, we guarantee that the burdensome RBF center calculation stage will be executed only for the particles of the first swarm. Thus, though the number of fitness function evaluations is doubled by the cooperative approach, the corresponding computational load increases by a significantly smaller factor.

The personal best position $P_k.\mathbf{y}_i(t)$ for each particle i of each swarm k represents the best result found for this particle up to iteration t. Calculation is based on evaluating the particle's performance on the validation dataset, by applying the selected fitness function f:

$$P_k.\mathbf{y}_i(t+1) = \begin{cases} P_k.\mathbf{y}_i(t), \text{if} f\left(\text{RBF}\left(P_k.\boldsymbol{\Lambda}_i(t+1), P_{(2k)^{2-k}}.\hat{y}(t)\right)\right) \\ \geq f\left(\text{RBF}\left(P_k.\mathbf{y}_i(t), P_{(2k)^{2-k}}.\hat{y}(t-1)\right)\right) \\ \\ P_k.\boldsymbol{\Lambda}_i(t+1)\ , \text{otherwise} \end{cases} \tag{21}$$

The global best position $P_k.\hat{y}(t)$ for each swarm k represents the best result found by the respective swarm up to iteration t:

$$P_k.\hat{y}(t) = arg\min_{P_k.\mathbf{y}_i(t),} f\left(\text{RBF}\left(P_k.\mathbf{y}_i(t), P_{(2k)^{2-k}}.\hat{y}(t-1)\right)\right),\ i = 1, \ldots P \tag{22}$$

Notice that (21)–(22) are suitable for minimization problems, but can be trivially modified to accommodate for error criteria which should be maximized, e.g. MCC.

A common problem in PSO-based techniques is the problem of stagnation, which refers to a situation where, as the number of iterations increases, the particles move too close to the swarm's global best solution, and thus have a very limited potential for space exploration and possibly, further improvement. In the case of cooperative swarms, the negative effects of stagnation can be augmented by the fact that calculation of each swarm's fitness, also involves the global best solution of the complementary swarm. Consider the following hypothetical situation: for a given global best vector of the fuzzy partition swarm, selection of narrow Gaussians results to improved fitness. Assuming that the global best of the fuzzy partition swarm remains constant for a long period, it is likely that the particles of the neighbor coverage swarm have been suitably adapted and thus have converged inside a small region of the search space, corresponding to low neighbor coverage percentages, which indeed favor narrow Gaussians. At this point, the global best of the fuzzy partition swarm changes and moves to a new position, where an even better performance could be achieved with wide Gaussians. Unfortunately, now it is not possible for the particles belonging to the neighbor coverage swarm to shift accordingly, as they have all stagnated in an area which can only produce narrow Gaussians.

To help the algorithm deal with such situations, the particles of the swarms are being reset to new random positions when the following two conditions hold simultaneously: A) one swarm has converged in a small region of the input space and B) the global best position of the complementary swarm has changed. Convergence of swarm k is assessed by calculating the normalized sum of distances to the global best of the swarm $P_k.S_{dnorm}$, as follows:

$$P_k.S_{dnorm}(t) = \frac{P_k.S_d(t)}{P_k.S_d(0)} \tag{23}$$

where $P_k.S_d(t)$ stands for the sum of distances of all particles, to the global best at iteration t:

$$P_k.S_d(t) = \sum_{i=1}^{P} \left[\|P_k.\boldsymbol{\Lambda}_i(t) - P_k.\hat{y}(t)\|\right] \tag{24}$$

When $P_k.S_{dnorm}(t)$ becomes smaller than a convergence constant ε, then swarm k is considered to be converged.

The pseudo-code for the proposed cooperative PSO with variable-width basis functions algorithm (COOP-VW) is given in Algorithm 1. A schematic overview of the two cooperative swarms, working towards optimizing RBF network models can be found in Fig. 2.

Algorithm 1 – COOP-VW Algorithm

Input: $\{\mathbf{U}_{\text{train}}, \mathbf{Y}_{\text{train}}\}, \{\mathbf{U}_{\text{val}}, \mathbf{Y}_{\text{val}}\}$: Training - validation data
$s_{\min}, s_{\max}, \kappa_{\min} \kappa_{\max}$: NSFM and NNC parameters,
P: Swarms population,
$c_1, c_2, w, P_k, V_{\max}$: PSO operational parameters,
ε, ξ: swarm convergence and stopping parameters

Output: $L_f, \hat{\mathbf{U}}_f, \mathbf{w}_f$: Optimized RBF network parameters

For i=1:P **Do:**
  Initialize the particles $P_1.\mathbf{\Lambda}_i(0)$ and $P_2.\mathbf{\Lambda}_i(0)$ at random numbers
  Use NSFM, NNC (11)-(12) and LR (6) to fully train $\text{RBF}(P_1.\mathbf{\Lambda}_i(0), P_2.\mathbf{\Lambda}_i(0))$ on $\{\mathbf{U}_{\text{train}}, \mathbf{Y}_{\text{train}}\}$
  Calculate fitness $\text{RBF}(P_1.\mathbf{\Lambda}_i(0), P_2.\mathbf{\Lambda}_i(0))$ on $\{\mathbf{U}_{\text{val}}, \mathbf{Y}_{\text{val}}\}$ and set $P_1.\mathbf{y}_i(0)$ and $P_2.\mathbf{y}_i(0)$
End For
Calculate global bests $P_1.\hat{\mathbf{y}}(0)$ and $P_2.\hat{\mathbf{y}}(0)$ (22)
Begin with the first iteration: $t \leftarrow 1$
While the stopping criterion ξ has not been met, **Do:**
  Start with the fuzzy partition swarm: $k \leftarrow 1$
  If $t>1$ and $P_2.\hat{\mathbf{y}}(t-1) \neq P_2.\hat{\mathbf{y}}(t-2)$ and $P_1.S_{dnorm}(t-1) < \varepsilon$
  Then reset particles $P_1.\mathbf{\Lambda}_i(t)$, $i=1,\ldots,P$
  For i=1:P **Do:**
    Use NSFM, NNC (11)-(12) and LR (6) to fully train $\text{RBF}(P_1.\mathbf{\Lambda}_i(t), P_2.\hat{\mathbf{y}}(t-1))$ on $\{\mathbf{U}_{\text{train}}, \mathbf{Y}_{\text{train}}\}$
    Calculate fitness $\text{RBF}(P_1.\mathbf{\Lambda}_i(t), P_2.\hat{\mathbf{y}}(t-1))$ on $\{\mathbf{U}_{\text{val}}, \mathbf{Y}_{\text{val}}\}$ and calculate $P_1.\mathbf{y}_i(t)$ (21)
  End for
  Calculate global best $P_1.\hat{\mathbf{y}}(t)$ (22)
  For i=1:P **Do:**
    For j=1:N **Do:**
      Update the velocity vector $P_1.v_{ij}(t+1)$ (17), (19)
    End for
    Update particle positions $P_1.\mathbf{\Lambda}_i(t+1)$ (16)
  End For
  Proceed to the neighbor coverage swarm: $k \leftarrow 2$
  If $t>1$ and $P_1.\hat{\mathbf{y}}(t) \neq P_1.\hat{\mathbf{y}}(t-1)$ and $P_2.S_d(t-1) < \varepsilon$
  Then reset particles $P_2.\mathbf{\Lambda}_i(t)$, $i=1,\ldots,P$
  For i=1:P **Do:**
    Use the RBF centers corresponding to $P_1.\hat{\mathbf{y}}(t)$, NNC (11)-(12) and LR (6) to fully train $\text{RBF}(P_2.\mathbf{\Lambda}_i(t), P_1.\hat{\mathbf{y}}(t))$ on $\{\mathbf{U}_{\text{train}}, \mathbf{Y}_{\text{train}}\}$
    Calculate fitness $\text{RBF}(P_2.\mathbf{\Lambda}_i(t), P_1.\hat{\mathbf{y}}(t))$ on $\{\mathbf{U}_{\text{val}}, \mathbf{Y}_{\text{val}}\}$ and calculate $P_2.\mathbf{y}_i(t)$ (21)
  End For
  Calculate global best $P_2.\hat{\mathbf{y}}(t)$ (22)
  For i=1:P **Do:**
    Update the velocity vector $P_2.v_i(t+1)$ (18), (19)
    Update particle positions $P_2.\mathbf{\Lambda}_i(t+1)$ (16)
  End For
  Proceed with the next iteration: $t \leftarrow t+1$
End While

4. Results and discussion

4.1. Experimental setup

The COOP-VW algorithm was tested on a variety of real world and synthetic machine learning benchmark datasets, addressing both tasks of function approximation and classification. The real world datasets are available online at the UCI Machine Learning Repository [38]. An overview of the employed datasets is given in Table 1, depicting the respective numbers of input and output variables, and the total number of examples. A short description for each dataset is given next.

Table 1
Benchmark datasets.

Dataset	Dataset origin[a]	#of inputs	#of outputs[b]	#of examples
Function Approximation Datasets				
Auto Price	RW	15	1	159
Energy	RW	8	1	768
Friedman	S	5	1	1000
Parkinsons	RW	16	1	5875
Samad	S	3	1	1500
Classification Datasets				
Cardiotocography	RW	21	10	2126
Iris	RW	4	3	150
Leaf	RW	14	30	340
Waveform	S	21	3	5000

[a] RW: Real-World datasets, S: Synthetic datasets.
[b] For classification datasets, the number of outputs equals the number of classes.

Table 2
Operational parameters for COOP-VW.

Parameter	Symbol	Value
Fuzzy set domain	$[s_{\min}\ s_{\max}]$	[4 50]
Neighbor coverage domain	$[\kappa_{\min}\ \kappa_{\max}]$	[2% 99%]
Population	P	20
Inertia	w	0.8
Nostalgia	c_1	1.5
Envy	c_2	1.5
Velocity clamping for fuzzy sets[a]	$P_1.V_{\max}$	S:5, M:15, L:25
Velocity clamping for neighbor coverage	$P_2.V_{\max}$	30
Maximum number of iterations	ξ	2000
Convergence constant	ε	0.1

[a] Value depending on input space dimensionality: Small problems (S): 1–4 input variables, Medium problems (M): 5–8 input variables, Large problems (L): more than 8 input variables [20].

• Auto price

This dataset concerns prediction of the price of a car, as a function of the car's specifications, e.g. engine size, horsepower, etc. Only continuous numerical variables were used as inputs.

• Energy efficiency

The aim in this dataset is to estimate the heating load requirements of buildings, i.e. the energy efficiency, as a function of the building parameters [39].

• Friedman

The Friedman function is used to generate this synthetic dataset [40]:

$$y = 5\left[2\sin(\pi x_1 x_2) + 4(x_3 - 0.5)^2 + 2x_4 + x_5\right] + n \tag{25}$$

Gaussian noise n $\sim N(0, 0.8)$ is added to the output. Inputs are sampled randomly within the range [0, 1], following a uniform distribution.

• Parkinsons telemonitoring

This dataset is composed of a range of voice measurements from 42 people with early-stage Parkinson's disease [41]. The objective is to estimate the motor unified Parkinson's disease rating scale (UPDRS) scores from 16 biomedical voice measures.

• Samad

The Samad function is used to generate this synthetic dataset [42]:

$$y = \frac{1}{1 + \exp\left[-\exp(x_1) + (x_2 - 0.5)^2 + \sin(\pi x_3)\right]} + n \tag{26}$$

Gaussian noise n $\sim N(0, 0.025)$ is added to the output. Inputs are sampled randomly within the range [0, 1], following a uniform distribution.

• Cardiotocography FHR

The cardiotocography dataset contains several fetal cardiotocograms (CTGs), which were automatically processed with their respective diagnostic features. The aim is to infer the fetal heart rate (FHR) morphologic pattern.

• Iris

This is perhaps the best known database to be found in the pattern recognition literature. The dataset contains 3 different target classes of 50 instances each, where each class refers to a type of iris plant.

• Leaf

This database comprises 30 different plant species, which should be classified based on shape and texture data extracted by photographs of the leaf specimens [43].

• Waveform database generator

This is a simulated dataset generated by written code in C. It is comprised of 21 attributes and data are divided in 3 different wave classes.

All datasets were randomly split in three subsets. Aside from the training and validation subsets, which are used for network parameter calculation and model selection, respectively, it is important to test the resulting RBF network performance on a third independent subset. This is due to overfitting that could occur with respect to the validation dataset, as the optimization procedure uses the latter to evaluate fitness. The available data were allocated to the three subsets randomly, using 50% of the data for training, 25% for validation and 25% for testing. In the case of classification problems, data belonging to the same classes were evenly distributed among the three subsets.

For comparison purposes, two different EC-based RBF optimization techniques were also tested. The first is the original algorithm [20], employing a single swarm and a basis function of fixed width, namely the TPS function (SINGLE-FW); a comparison against this method puts to test the efficiency of the variable-width approach against a successful optimization technique for fixed-width RBF networks. The second EC-based comparison approach involved a non-cooperative variation using a single swarm and variable-width Gaussian basis functions (SINGLE-VW). In this case, a single swarm is employed, populated by individuals which include both the fuzzy partition and neighbor coverage, following the configuration described by (13). This technique is used to assess the benefits of the cooperative learning approach against a similar, but non-cooperative one. Finally, a non-EC standard approach based on a different architecture and training algorithm was also tested, namely MFF networks using Levenberg-Marquardt for training [2].

To better evaluate the results, exhaustive search was applied on the dataset with the lower number of inputs, i.e. the Samad dataset, so as to determine the global optimal solution for the RBF network parameters. As this is a very computationally expensive method, it was practically infeasible to implement in datasets of larger size.

Table 3
Results for function approximation datasets – RMSE, fuzzy partition, neighbor coverage percentage, number of neurons and iterations needed to discover the best solution.

Dataset	Algorithm	RMSE (Validation)	RMSE (Testing)	Fuzzy partition	Neighbor coverage	Number of neurons[a]	Iterations to discover best
Auto Price	COOP-VW	$\mathbf{1.80 \times 10^3}$ ($\mathbf{1.98 \times 10^3} \pm 5.93 \times 10^1$)	$\mathbf{1.70 \times 10^3}$ ($\mathbf{1.81 \times 10^3} \pm 5.70 \times 10^1$)	[4 5 17 4 13 9 50 4 13 21 4 12 50 7 29]	2.00%	**50** (60 ± 6)	1224 (**801** ± 421)
	SINGLE-VW	1.83×10^3 ($1.99 \times 10^3 \pm 6.33 \times 10^1$)	1.84×10^3 ($1.83 \times 10^3 \pm 5.79 \times 10^1$)	[4 5 17 4 4 12 50 4 4 4 50 50 6 47 50]	2.00%	56 (59 ± 8)	1738 (1293 ± 560)
	SINGLE-FW	2.17×10^3 ($2.38 \times 10^3 \pm 1.28 \times 10^2$)	2.11×10^3 ($2.30 \times 10^3 \pm 9.06 \times 10^1$)	[4 4 29 50 50 23 7 34 5 10 24 4 35 4 37]	–	57 (**57** ± 5)	**832** (825 ± 295)
	MFF	2.18×10^3 ($2.61 \times 10^3 \pm 2.87 \times 10^2$)	2.14×10^3 ($2.53 \times 10^3 \pm 3.24 \times 10^2$)	–	–	[5 5]	–
Energy	COOP-VW	$\mathbf{5.22 \times 10^{-1}}$ ($\mathbf{5.38 \times 10^{-1}} \pm 7.68 \times 10^{-3}$)	$\mathbf{5.10 \times 10^{-1}}$ ($\mathbf{5.16 \times 10^{-1}} \pm 1.99 \times 10^{-2}$)	[50 15 4 4 17 6 7 14]	99.00%	**174** (**198** ± 20)	**64** (**732** ± 365)
	SINGLE-VW	5.29×10^{-1} ($5.56 \times 10^{-1} \pm 2.32 \times 10^{-2}$)	5.31×10^{-1} ($5.40 \times 10^{-1} \pm 3.19 \times 10^{-2}$)	[6 6 50 9 50 4 50 49]	99.00%	212 (208 ± 23)	1702 (1239 ± 687)
	SINGLE-FW	9.57×10^{-1} ($1.00 \times 10^{0} \pm 5.08 \times 10^{-2}$)	8.95×10^{-1} ($9.84 \times 10^{-1} \pm 9.81 \times 10^{-2}$)	[9 44 4 5 50 20 8 22]	–	313 (345 ± 31)	1643 (1367 ± 417)
	MFF	1.17×10^{0} ($1.68 \times 10^{0} \pm 3.89 \times 10^{-1}$)	1.08×10^{0} ($1.58 \times 10^{0} \pm 3.13 \times 10^{-1}$)	–	–	[39 27]	–
Friedman	COOP-VW	$\mathbf{8.65 \times 10^{-1}}$ ($\mathbf{8.84 \times 10^{-1}} \pm 9.54 \times 10^{-3}$)	$\mathbf{1.01 \times 10^{0}}$ ($\mathbf{1.01 \times 10^{0}} \pm 2.16 \times 10^{-2}$)	[4 7 10 4 11]	99.00%	99 (**104** ± 11)	**78** (**51** ± 33)
	SINGLE-VW	8.76×10^{-1} ($9.40 \times 10^{-1} \pm 1.30 \times 10^{-1}$)	1.04×10^{0} ($1.05 \times 10^{0} \pm 1.01 \times 10^{-1}$)	[4 28 4 6 4]	93.64%	**84** (131 ± 48)	426 (97 ± 59)
	SINGLE-FW	9.08×10^{-1} ($9.33 \times 10^{-1} \pm 1.61 \times 10^{-2}$)	1.06 ($1.07 \pm 3.03 \times 10^{-2}$)	[8 4 4 32 4]	–	114 (116 ± 66)	245 (949 ± 285)
	MFF	1.03×10^{0} ($1.19 \times 10^{0} \pm 1.22 \times 10^{-1}$)	1.11×10^{0} ($1.20 \times 10^{0} \pm 1.30 \times 10^{-1}$)	–	–	[12 6]	–
Parkinsons	COOP-VW	$\mathbf{6.56 \times 10^{0}}$ ($\mathbf{6.62 \times 10^{0}} \pm 3.10 \times 10^{-2}$)	$\mathbf{6.55 \times 10^{0}}$ ($\mathbf{6.60 \times 10^{0}} \pm 3.71 \times 10^{-2}$)	[8 12 15 4 50 33 4 4 9 18 50 12 48 27 50 4]	5.34%	426 (**403** ± 97)	**868** (**657** ± 442)
	SINGLE-VW	6.57×10^{0} ($\mathbf{6.62 \times 10^{0}} \pm 4.82 \times 10^{-2}$)	6.56×10^{0} ($6.63 \times 10^{0} \pm 8.61 \times 10^{-2}$)	[4 17 9 50 4 8 21 4 6 10 50 17 50 50 32 4]	2.00%	434 (**403** ± 82)	1646 (1078 ± 594)
	SINGLE-FW	6.57×10^{0} ($6.64 \times 10^{0} \pm 3.02 \times 10^{-2}$)	6.58×10^{0} ($6.63 \times 10^{0} \pm 2.58 \times 10^{-1}$)	[32 26 14 4 4 9 19 4 12 15 12 4 48 50 6 50]	–	**341** (423 ± 73)	999 (949 ± 122)
	MFF	6.71×10^{0} ($6.91 \times 10^{0} \pm 1.22 \times 10^{-1}$)	7.04×10^{0} ($6.82 \times 10^{0} \pm 1.49 \times 10^{-1}$)	–	–	[12 10]	–
Samad	COOP-VW	$\mathbf{2.39 \times 10^{-2}}$ ($\mathbf{2.42 \times 10^{-2}} \pm 9.14 \times 10^{-5}$)	$\mathbf{2.71 \times 10^{-2}}$ ($\mathbf{2.73 \times 10^{-2}} \pm 3.05 \times 10^{-4}$)	[5 4 39]	93.41%	**83** (88 ± 7)	104 (**85** ± 50)
	SINGLE-VW	2.41×10^{-2} ($2.43 \times 10^{-2} \pm 3.01 \times 10^{-4}$)	2.73×10^{-2} ($2.74 \times 10^{-2} \pm 6.27 \times 10^{-4}$)	[17 4 14]	73.15%	91 (91 ± 10)	315 (146 ± 103)
	SINGLE-FW	2.47×10^{-2} ($2.48 \times 10^{-2} \pm 2.11 \times 10^{-4}$)	2.80×10^{-2} ($2.79 \times 10^{-2} \pm 4.40 \times 10^{-4}$)	[4 6 38]	–	89 (**87** ± 19)	**89** (178 ± 72)
	MFF	2.47×10^{-2} ($3.10 \times 10^{-2} \pm 3.70 \times 10^{-3}$)	2.82×10^{-2} ($3.47 \times 10^{-2} \pm 4.16 \times 10^{-3}$)	–	–	[11 11]	–

Results from 30 runs per dataset are presented for all methods. The first number in each cell represents the run that scored the lowest RMSE in the validation dataset; next is given in parentheses the average value of all runs, followed by the standard deviation. The best result for each distinct evaluation criterion is highlighted in bold.

[a] The following form is adopted for MFF networks: [1st layer neurons 2nd layer neurons].

Table 4
Results for classification datasets—MCC, fuzzy partition, neighbor coverage percentage, number of neurons and iterations needed to discover the best solution.

Dataset	Algorithm	MCC (Validation)	MCC (Testing)	Fuzzy partition	Neighbor coverage	Number of neurons[a]	Iterations to discover best
Cardio-	COOP-VW	$\mathbf{8.27 \times 10^{-1}}$ ($\mathbf{8.08 \times 10^{-1}} \pm 1.31 \times 10^{-2}$)	$\mathbf{7.88 \times 10^{-1}}$ ($7.84 \times 10^{-1} \pm 1.44 \times 10^{-2}$)	[17 4 4 4 4 50 4 4 4 49 4 4 4 31 4 15 50 4 7 50 4]	96.25%	**248** (**495** ± 228)	**1073** (**995** ± 325)
tocography	SINGLE-VW	8.25×10^{-1} ($8.05 \times 10^{-1} \pm 1.34 \times 10^{-2}$)	7.87×10^{-1} ($7.84 \times 10^{-1} \pm 1.61 \times 10^{-2}$)	[4 7 45 7 4 4 4 4 50 50 4 50 6 4 4 4 48 50 4 50 4]	4.53%	480 (526 ± 229)	1892 (1502 ± 255)
	SINGLE-FW	8.25×10^{-1} ($8.07 \times 10^{-1} \pm 1.35 \times 10^{-2}$)	$\mathbf{7.88 \times 10^{-1}}$ ($\mathbf{7.85 \times 10^{-1}} \pm 1.63 \times 10^{-2}$)	[4 50 50 4 4 17 50 4 16 4 4 4 4 4 48 4 4 4 5 50 50]	–	342 (635 ± 252)	1493 (1097 ± 445)
	MFF	7.81×10^{-1} ($7.54 \times 10^{-1} \pm 2.61 \times 10^{-2}$)	7.86×10^{-1} ($7.77 \times 10^{-1} \pm 2.53 \times 10^{-2}$)	–	–	[36 33]	–
Iris	COOP-VW	$\mathbf{1.00 \times 10^{0}}$ ($\mathbf{1.00 \times 10^{0}} \pm 0.00 \times 10^{0}$)	$\mathbf{1.00 \times 10^{0}}$ ($\mathbf{9.73 \times 10^{-1}} \pm 3.66 \times 10^{-2}$)	[24 4 14 6]	66.56%	**21** (**41** ± 13)	**2** (**2.13** ± 0.43)
	SINGLE-VW	$\mathbf{1.00 \times 10^{0}}$ ($\mathbf{1.00 \times 10^{0}} \pm 0.00 \times 10^{0}$)	$\mathbf{1.00 \times 10^{0}}$ ($9.64 \times 10^{-1} \pm 3.89 \times 10^{-2}$)	[16 20 31 9]	84.56%	44 (46 ± 13)	4 (3.09 ± 1.09)
	SINGLE-FW	$\mathbf{1.00 \times 10^{0}}$ ($\mathbf{1.00 \times 10^{0}} \pm 0.00 \times 10^{0}$)	$\mathbf{1.00 \times 10^{0}}$ ($9.64 \times 10^{-1} \pm 4.60 \times 10^{-2}$)	[31 27 16 50]	–	64 (52 ± 12)	3 (2.20 ± 0.41)
	MFF	$\mathbf{1.00 \times 10^{0}}$ ($9.70 \times 10^{-1} \pm 1.58 \times 10^{-2}$)	9.59×10^{-1} ($9.47 \times 10^{-1} \pm 2.83 \times 10^{-2}$)	–	–	[39 30]	–
Leaf	COOP-VW	$\mathbf{9.08 \times 10^{-1}}$ ($\mathbf{8.96 \times 10^{-1}} \pm 9.48 \times 10^{-3}$)	$\mathbf{8.50 \times 10^{-1}}$ ($\mathbf{8.00 \times 10^{-1}} \pm 2.82 \times 10^{-2}$)	[4 4 7 4 17 50 11 4 4 50 50 4 8 4]	58.17%	**94** (105 ± 13)	**692** (**728** ± 305)
	SINGLE-VW	$\mathbf{9.08 \times 10^{-1}}$ ($8.93 \times 10^{-1} \pm 1.37 \times 10^{-2}$)	7.81×10^{-1} ($7.94 \times 10^{-1} \pm 4.09 \times 10^{-2}$)	[4 23 4 4 4 43 7 43 4 49 50 4 19 4]	99.00%	98 (**104** ± 16)	1418 (1398 ± 589)
	SINGLE-FW	8.97×10^{-1} ($8.83 \times 10^{-1} \pm 9.86 \times 10^{-3}$)	7.12×10^{-1} ($7.62 \times 10^{-1} \pm 3.22 \times 10^{-2}$)	[10 46 48 28 5 45 14 13 38 11 4 44 48 12]	–	147 (129 ± 15)	1979 (1223 ± 528)
	MFF	7.46×10^{-1} ($6.52 \times 10^{-1} \pm 4.74 \times 10^{-2}$)	7.01×10^{-1} ($6.31 \times 10^{-1} \pm 5.47 \times 10^{-2}$)	–	–	[35 33]	–
Waveform	COOP-VW	$\mathbf{8.40 \times 10^{-1}}$ ($\mathbf{8.33 \times 10^{-1}} \pm 3.49 \times 10^{-3}$)	$\mathbf{8.02 \times 10^{-1}}$ ($\mathbf{8.05 \times 10^{-1}} \pm 7.79 \times 10^{-3}$)	[4 4 4 4 30 4 4 4 39 6 4 5 4 4 4 5 4 4 4 4 4]	47.00%	**43** (61 ± 23)	1219 (**660** ± 441)
	SINGLE-VW	8.36×10^{-1} ($8.30 \times 10^{-1} \pm 3.59 \times 10^{-3}$)	7.88×10^{-1} ($8.01 \times 10^{-1} \pm 8.01 \times 10^{-3}$)	[4 4 4 4 16 4 49 4 4 5 4 4 6 4 4 4 9 4 10 4 4]	2.00%	54 (70 ± 25)	1565 (1022 ± 586)
	SINGLE-FW	8.34×10^{-1} ($8.29 \times 10^{-1} \pm 3.51 \times 10^{-3}$)	8.01×10^{-1} ($8.00 \times 10^{-1} \pm 7.98 \times 10^{-3}$)	[4 4 4 4 47 26 4 4 4 4 5 4 9 4 4 4 4 4 4 4 4]	–	57 (**49** ± 18)	**1170** (1044 ± 524)
	MFF	8.10×10^{-1} ($7.94 \times 10^{-1} \pm 7.62 \times 10^{-3}$)	7.95×10^{-1} ($7.98 \times 10^{-1} \pm 5.52 \times 10^{-3}$)	–	–	[39 5]	–

Results from 30 runs per dataset are presented for all methods. The first number in each cell represents the run that scored the higher MCC in the validation dataset; next is given in parentheses the average value of all runs, followed by the standard deviation. The best result for each distinct evaluation criterion is highlighted in bold.

[a] The following form is adopted for MFF networks: [1st layer neurons 2nd layer neurons].

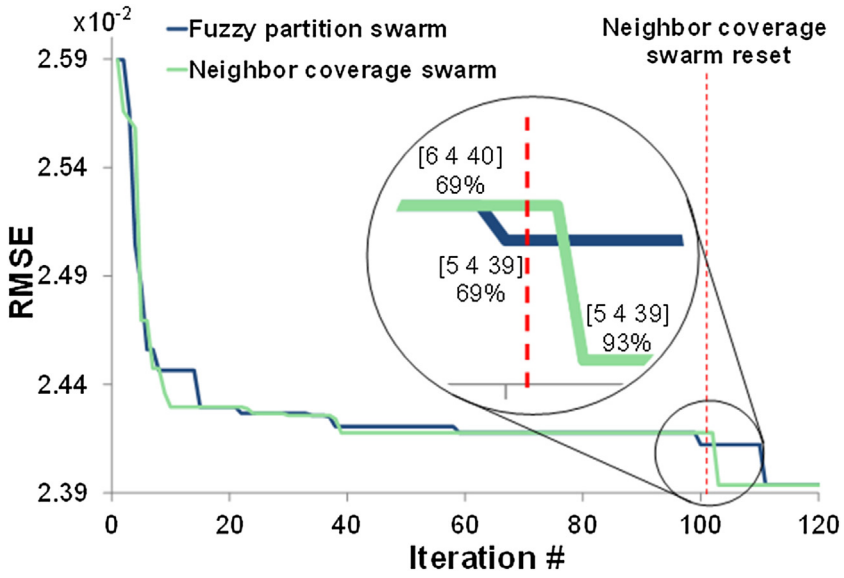

Fig. 3. Best solution provided by COOP-VW, for the Samad dataset: Change of RMSE per iteration in the validation subset. The magnified area focuses on the point where the neighbor coverage swarm locates the global optimum, following a successful reset.

4.2. Methods configuration—parameter selection

The operational parameters used by COOP-VW, which were applied to all experiments, are given in Table 2. The same parameters, where applicable, were also used for SINGLE-VW, whereas the parameters reported in [20] were utilized for SINGLE-FW.

Parameter selection was based on suggestions in literature, in conjunction with trial and error [44]. A different constant for velocity clamping was utilized for each swarm. A problem-depended approach was applied for selecting the fuzzy set swarm velocity clamping constant, which was linked to the input space dimensionality [20]. Allowing higher velocities in problems with high input dimensionality was found to improve results. On the other hand, the same inertia weight w and acceleration coefficients c_1 and c_2 were used for both swarms.

As far as the parameters associated with RBF training and the NSFM algorithm are concerned, the selection was common in all three RBF-based approaches, along the guidelines provided in [20]. Finally it should be noted that larger values of population size were found to further improve the results, however, they are not recommended for larger datasets, due to the increased computational load.

Experiments were run 30 times for all tested methods, due to their stochastic nature. In the case of MFF networks, two hidden layers were used and the selection of the respective number of neurons was performed using exhaustive search. Allowing the number of neurons in each distinct hidden layer to range between 5 and 40, each possible combination of neurons per layer was tested 30 times, starting from different initial weights in each run.

4.3. Results and discussion

Fig. 3 depicts the change of the RMSE criterion in the validation subset per iteration number, as far as the best result found by COOP-VW for the Samad dataset is concerned. The particular dataset is singled-out as it is the only one which allows for global optimum calculation through exhaustive search in acceptable time. The exhaustive search procedure identifies the optimal solution at [5 4 39] fuzzy sets and a neighbor coverage percentage of 93%. It is interesting to note that, based on the exhaustive search results, the search space around the point [5 4 39] generates a number of good sub-optimal solutions which, however, are combined with lower neighbor coverage values ranging between 50 and 70%.

The COOP-VW algorithm locates the global optimum in 2 out of the 30 runs, whereas the SINGLE-VW algorithm fails to do so, but settles for the second-best solution, identified by exhaustive search at [17 4 14] fuzzy sets per input dimension and 73% neighbor coverage percentage. Some insight into the swarms' cooperation and the beneficial effects of swarm resetting can be obtained from Fig. 3. It can be seen that the fuzzy partition swarm approaches the area where the global optimum is located and at iteration 100 the best particle of the swarm lands on [5 4 39]; this particle constitutes the global best up to that point for the fuzzy partition, but when combined with 69% of neighbor coverage, results to a sub-optimal solution. At this iteration, as indicated by a low value for the sum of distances to the global best particle of the neighbor coverage swarm, all the respective particles have converged near the value of 69%; therefore, it would be highly unlikely for a particle of the neighbor coverage swarm to jump to the value of 93% and locate the global optimal solution, and the algorithm would probably stagnate. In the proposed approach, this situation triggers a reset of the neighbor coverage swarm, which enables it to escape from the stagnation point and discover the global optimum.

The results for the function approximation and classification datasets, are synopsized in Tables 3 and 4, respectively, depicting the corresponding cost function value for validation and testing and the number of neurons, for all methods; the fuzzy partition, neighbor coverage percentage, and iterations for discovering the best solution are also included for the three RBF-based methodologies. It can be seen that the proposed approach produces the best accuracy in terms of best and average solutions, in the vast majority of either function approximation, or classification datasets. To be more specific, COOP-VW achieves a significant advantage over the MFF networks, which clearly rank fourth, despite being the only method using exhaustive search for network structure optimization. The COOP-VW and SINGLE-VW algorithms also significantly outperform the SINGLE-FW approach, outlining the merits of extending the original algorithm using variable basis function widths. Note that SINGLE-FW has already been found to produce superior results compared to the standard FM approach [20].

Though the differences in testing accuracy are smaller when comparing between the two variable basis function width algorithms, the proposed method still maintains a clear advantage.

Additionally, the cooperative algorithm exhibits significantly lower standard deviations around the average; this indicates the robustness of the COOP-VW algorithm, which provides more consistent results. It is also important to mention that in all datasets, the higher accuracy exhibited by the COOP-VW technique is combined with fewer iterations needed to discover the best solution, in comparison with the non-cooperative SINGLE-VW. Note that, as discussed in Section 3.3, though in the cooperative approach a single iteration involves more fitness function evaluations compared to the non-cooperative one, computational load is not comparably increased; this is due to the algorithm being designed so as to perform the same number of RBF center calculations per iteration with the non-cooperative framework. Finally, it should be noted that in a large number of datasets, the COOP-VW technique favors networks with smaller structures.

5. Conclusions

This work presents a new method for fully optimizing RBF networks using a cooperative PSO approach. The problem of concurrently optimizing the centers and widths of the basis functions is solved using a specially designed particle encoding, which dismantles the problem in two parts and thus utilizes two distinct swarms. The first swarm is responsible for the selection of the network structure and RBF kernel center coordinates through the NSFM algorithm, whereas the second takes care of width selection by taking into account the nearest neighbor coverage. The two swarms work together towards discovering improved solutions in a cooperative manner, while a specially designed reset operation is introduced to avoid stagnation.

The resulting algorithm is evaluated on 9 different benchmarks, covering a range of real world and simulated datasets, which include function approximation and classification problems. For comparison purposes, three additional methodologies are also implemented, namely RBF networks optimized with non-cooperative PSO using variable and fixed-width functions, and MFF networks trained using the Levenberg-Marquardt algorithm. Results show that the proposed approach greatly outperforms the MFF networks and the fixed-width method, as far as accuracy is concerned. Moreover, the cooperative algorithm not only provides better and more consistent results in terms of accuracy compared to the non-cooperative variable-width method, but also manages to discover the best solution in less iterations. The superior performance is in most cases combined with smaller network structures.

References

[1] S. Haykin, Neural Networks: A Comprehensive Foundation, second ed., Prentice Hall, Upper Saddle River, NJ, 1999.
[2] M.T. Hagan, M. Menhaj, Training feedforward networks with the Marquardt algorithm, IEEE Trans. Neural Netw. 5 (1994) 989–993.
[3] D.B. Fogel, L.J. Fogel, V.W. Porto, Evolutionary Programming for Training Neural Networks, International Joint Conference on Neural Networks (IJCNN), San Diego, CA, USA, 1990, p. 601–5.
[4] R. Bisoi, P.K. Dash, A hybrid evolutionary dynamic neural network for stock market trend analysis and prediction using unscented Kalman filter, Appl. Soft Comput. 19 (2014) 41–56.
[5] J.P. Donate, P. Cortez, G. Gutiérrez Sánchez, A. Sanchis de Miguel, Time series forecasting using a weighted cross-validation evolutionary artificial neural network ensemble, Neurocomputing 109 (2013) 27–32.
[6] J.P. Donate, X. Li, G. Gutiérrez Sánchez, A. Sanchis de Miguel, Time series forecasting by evolving artificial neural networks with genetic algorithms, differential evolution and estimation of distribution algorithm, Neural Comput. Appl. 22 (2013) 11–20.
[7] S. Kiranyaz, T. Ince, A. Yildirim, M. Gabbouj, Evolutionary artificial neural networks by multi-dimensional particle swarm optimization, Neural Netw. 22 (2009) 1448–1462.
[8] R.K. Agrawal, N.G. Bawane, Multiobjective PSO based adaption of neural network topology for pixel classification in satellite imagery, Appl. Soft Comput. 28 (2015) 217–225.
[9] N.S. Jaddi, S. Abdullah, A.R. Hamdan, Optimization of neural network model using modified bat-inspired algorithm, Appl. Soft Comput. 37 (2015) 71–86.
[10] S.C. Tan, J. Watada, Z. Ibrahim, M. Khalid, Evolutionary fuzzy ARTMAP neural networks for classification of semiconductor defects, IEEE Trans Neural Netw. Learn. Syst. (2014).
[11] D.A. Silva, J.P. Silva, A.R. Rocha Neto, Novel approaches using evolutionary computation for sparse least square support vector machines, Neurocomputing 168 (2015) 908–916.
[12] J. Moody, C. Darken, Fast learning in networks of locally-tuned processing units, Neural Comput. 2 (1989) 281–294.
[13] H. Sarimveis, A. Alexandridis, S. Mazarakis, G. Bafas, A new algorithm for developing dynamic radial basis function neural network models based on genetic algorithms, Comput. Chem. Eng. 28 (2004) 209–217.
[14] F. Fernández-Navarro, C. Hervás-Martínez, R. Ruiz, J.C. Riquelme, Evolutionary Generalized Radial Basis Function neural networks for improving prediction accuracy in gene classification using feature selection, Appl. Soft Comput. 12 (2012) 1787–1800.
[15] D.P. Ferreira Cruz, R. Dourado Maia, L.A. da Silva, L.N. de Castro, BeeRBF: a bee-inspired data clustering approach to design RBF neural network classifiers, Neurocomputing (2016), http://dx.doi.org/10.1016/j.neucom.2015.03.106, in press.
[16] W. Huang, S.-K. Oh, W. Pedrycz, Design of hybrid radial basis function neural networks (HRBFNNs) realized with the aid of hybridization of fuzzy clustering method (FCM) and polynomial neural networks (PNNs), Neural Netw. 60 (2014) 166–181.
[17] V. Fathi, G.A. Montazer, An improvement in RBF learning algorithm based on PSO for real time applications, Neurocomputing 111 (2013) 169–176.
[18] S.K. Oh, W.D. Kim, W. Pedrycz, S.C. Joo, Design of K-means clustering-based polynomial radial basis function neural networks (pRBFNNs) realized with the aid of particle swarm optimization and differential evolution, Neurocomputing 78 (2012) 121–132.
[19] S.K. Oh, W.D. Kim, W. Pedrycz, B.J. Park, Polynomial-based radial basis function neural networks (P-RBF NNs) realized with the aid of particle swarm optimization, Fuzzy Sets Syst. 163 (2011) 54–77.
[20] A. Alexandridis, E. Chondrodima, H. Sarimveis, Radial Basis Function network training using a non-symmetric partition of the input space and Particle Swarm Optimization, IEEE Trans Neural Netw. Learn. Syst. 24 (2013) 219–230.
[21] F. van den Bergh, A.P. Engelbrecht, A cooperative approach to particle swarm optimization, IEEE Trans Evol. Comput. 8 (2004) 225–239.
[22] S. Ono, H. Maeda, K. Sakimoto, S. Nakayama, User-system cooperative evolutionary computation for both quantitative and qualitative objective optimization in image processing filter design, Appl. Soft Comput. 15 (2014) 203–218.
[23] W. Zhao, S. Alam, H.A. Abbass, MOCCA-II: a multi-objective co-operative co-evolutionary algorithm, Appl. Soft Comput. 23 (2014) 407–416.
[24] F.B. de Oliveira, R. Enayatifar, H.J. Sadaei, F.G. Guimarães, J.Y. Potvin, A cooperative coevolutionary algorithm for the multi-depot vehicle routing problem, Expert Syst. Appl. 43 (2016) 117–130.
[25] N. Garcia-Pedrajas, C. Hervas-Martinez, J. Munoz-Perez, COVNET: a cooperative coevolutionary model for evolving artificial neural networks, IEEE Trans. Neural Netw. 14 (2003) 575–596.
[26] R. Chandra, M. Frean, M. Zhang, Crossover-based local search in cooperative co-evolutionary feedforward neural networks, Appl. Soft Comput. 12 (2012) 2924–2932.
[27] R. Chandra, M. Zhang, Cooperative coevolution of Elman recurrent neural networks for chaotic time series prediction, Neurocomputing 86 (2012) 116–123.
[28] L. Zhao, F. Qian, Tuning the structure and parameters of a neural network using cooperative binary-real particle swarm optimization, Expert Syst. Appl. 38 (2011) 4972–4977.
[29] M.D. Pérez-Godoy, A.J. Rivera, C.J. Carmona, M.J. del Jesus, Training algorithms for Radial Basis Function Networks to tackle learning processes with imbalanced data-sets, Appl. Soft Comput. 25 (2014) 26–39.
[30] M.D. Pérez-Godoy, A.J. Rivera, F.J. Berlanga, M.J. Del Jesus, CO2RBFN: an evolutionary cooperative-competitive RBFN design algorithm for classification problems, Soft Comput. 14 (2010) 953–971.
[31] C. Darken, J. Moody, Fast adaptive K-means clustering: some empirical results, in: IEEE INNS International Joint Conference On Neural Networks, San Diego, CA, 1990, pp. 233–238.
[32] H. Sarimveis, A. Alexandridis, G. Tsekouras, G. Bafas, A fast and efficient algorithm for training radial basis function neural networks based on a fuzzy partition of the input space, Ind. Eng. Chem. Res. 41 (2002) 751–759.
[33] A. Alexandridis, H. Sarimveis, G. Bafas, A new algorithm for online structure and parameter adaptation of RBF networks, Neural Netw. 16 (2003) 1003–1017.
[34] A. Alexandridis, H. Sarimveis, K. Ninos, A Radial Basis Function network training algorithm using a non-symmetric partition of the input space—application to a model predictive control configuration, Adv. Eng. Softw. 42 (2011) 830–837.
[35] M. Potter, K. De Jong, A cooperative coevolutionary approach to function optimization, in: Y. Davidor, H.-P. Schwefel, R. Männer (Eds.), Parallel Problem Solving from Nature—PPSN III, Springer, Berlin, Heidelberg, 1994, pp. 249–257.
[36] F. van den Bergh, A.P. Engelbrecht, Cooperative learning in neural networks using particle swarm optimizers, South Afr. Comput. J. 26 (2000) 84–90.

[37] J. Gorodkin, Comparing two K-category assignments by a K-category correlation coefficient, Comput. Biol. Chem. 28 (2004) 367–374.
[38] A. Asuncion, D.J. Newman, UCI Machine Learning Repository, Univ. California Irvine, Irvine, CA, 2007.
[39] A. Tsanas, A. Xifara, Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools, Energy Build. 49 (2012) 560–567.
[40] J. Friedman, Multivariate adaptive regression splines, Ann. Stat. 19 (1991) 1–67.
[41] A. Tsanas, M. Little, P. McSharry, L. Ramig, Accurate telemonitoring of Parkinson's disease progression by Noninvasive Speech Tests, IEEE Trans. Biomed. Eng. 57 (2010) 884–893.
[42] T. Samad, Backpropagation with expected source values, Neural Netw. 4 (1991) 615–618.
[43] P.B. Silva, A.S. Marçal, R.A. da Silva, Evaluation of features for leaf discrimination, in: M. Kamel, A. Campilho (Eds.), Image Analysis and Recognition, Springer, Berlin, Heidelberg, 2013, pp. 197–204.
[44] A. Engelbrecht, Computational Intelligence: An Introduction, 2nd ed., John Wiley & Sons, Ltd., Chichester, 2007.