Neural Networks 75 (2016) 150–161

Contents lists available at ScienceDirect

Neural Networks

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

Two fast and accurate heuristic RBF learning rules for data classification

Modjtaba Rouhani *, Dawood S. Javan

Faculty of engineering, Ferdowsi University of Mashhad, Mashhad, Iran

ARTICLE INFO

Article history:
Received 8 March 2015
Received in revised form 24 October 2015
Accepted 22 December 2015
Available online 4 January 2016

Keywords:
Radial Basis Function
Neural network learning
Heuristic learning method
Classification problem
Linear separability in feature space

ABSTRACT

This paper presents new Radial Basis Function (RBF) learning methods for classification problems. The proposed methods use some heuristics to determine the spreads, the centers and the number of hidden neurons of network in such a way that the higher efficiency is achieved by fewer numbers of neurons, while the learning algorithm remains fast and simple. To retain network size limited, neurons are added to network recursively until termination condition is met. Each neuron covers some of train data. The termination condition is to cover all training data or to reach the maximum number of neurons. In each step, the center and spread of the new neuron are selected based on maximization of its coverage. Maximization of coverage of the neurons leads to a network with fewer neurons and indeed lower VC dimension and better generalization property. Using power exponential distribution function as the activation function of hidden neurons, and in the light of new learning approaches, it is proved that all data became linearly separable in the space of hidden layer outputs which implies that there exist linear output layer weights with zero training error. The proposed methods are applied to some well-known datasets and the simulation results, compared with SVM and some other leading RBF learning methods, show their satisfactory and comparable performance.

1. Introduction

Radial Basis Functions (RBFs) are among those neural networks originated from biological concepts and applied to regression as well as classification problems (Moody & Darken, 1989; Powell, 1985). Bormhead and Lowe (1988) presented a method for fast learning of RBF neural networks. Their work was followed by Park and Sandberg (1991) who proved that RBFs can approximate any arbitrary nonlinear function under some mild conditions, to any desirable accuracy. Soon after RBFs were applied to a wide range of engineering problems, like fault recognition (Huang & Tan, 2009; Meng, Dong, Wang, & Wong, 2010), image processing (Ferrari, Bellocchio, Piuri, & Borghese, 2010; Lee & Yoon, 2010), and adaptive control (Cai, Rad, & Chan, 2010; Tsai, Huang, & Lin, 2010; Yu, Xie, Paszczynski, & Wilamowski, 2011). The generalization property of RBFs and other neural networks is widely investigated in the literature. Some authors estimated bounds on the number of hidden neurons required in terms of input dimension, widths of the neurons, the minimal separation among the centers and the possibility of fixing the centers (Girosi & Anzellotti, 1993; Kainen, Kurková, & Sanguineti, 2009, 2012; Mhaskar, 2004a, 2004b). Although many studies place emphasis on the number of hidden neurons, Bartlett (1998) found that the size of the weights may be more important than the size of the network.

The main advantage of traditional RBFs lies in its simple and fast learning rule which in contrast to back propagation networks is not iterative and has no convergence problem. However, early RBF learning rules required a large number of hidden neurons and normally had low generalization performances (Yu et al., 2011). In the last two decades, different RBF learning methods have been proposed to reduce the number of hidden neurons by means of smarter selection of neuron centers and spreads.

Determination of required number of hidden units, their centers and spreads are the main parts of an RBF learning rule. Orr (1995) introduced the regularization and subset selection methods for proper selection of units' centers. Chen, Cowan, and Grant (1991) used orthogonal least squares for optimal addition of new RB units. Huang, Saratchandran, and Sundararajan (2004) proposed the sequential growing and pruning algorithm to design an RBF. Wu and Chow (2004) applied advanced self-organizing map (SOM) to cluster training dataset and to select the center of the clusters as units' center. Xie, Yu, Hewlett, Rózycki, and Wilamowski (2012) implemented second order gradient method to learn an

* Corresponding author. Tel.: +98 513 8805122; fax: +98 513 8807801.
E-mail addresses: rouhani@um.ac.ir (M. Rouhani), dawood.javan@stu-mail.um.ac.ir (D.S. Javan).

http://dx.doi.org/10.1016/j.neunet.2015.12.011
0893-6080/

RBF. In their method, all parameters of the network (hidden layer's spreads and centers together with output weights) are learned iteratively by second order gradient approach. Hien, Huan, and Huynh (2009) and Huan, Hien, and Huynh (2007) proposed a two-phase algorithm for training of an interpolation RBF with many hidden neurons. Gaussian function is one of the most popular activation functions in RBF networks (Blanzieri, 2003; Hartman, Keeler, & Kowalski, 1990). Huan, Hien, and Tue (2011) replaced Euclidean norm in Gaussian function with Mahalanobis norm and calculated spread parameters, directly. In fact, one of the challenges in RBF modeling is the optimization of spread parameters. Yao, Chen, Zhao, and Tooren (2012) proposed an optimization method for spread parameters based on concurrent subspace width optimization. A more complete review of RBF and its learning rules and applications could be found in Yu et al. (2011).

Some authors (Constantinopoulos & Likas, 2006; Jaiyen, Lursinsap, & Phimoltares, 2010; Mao & Huang, 2005; Oyang, Hwang, Ou, Chen, & Chen, 2005; Titsias & Likas, 2001) concentrated on RBF learning rules for classification problems. Constantinopoulos and Likas (2006) considered the idea of adding the new neurons to the most crucial regions. When all neurons are added, every component is split into subcomponents, and each one corresponds to a different class. Mao and Huang (2005) used data structure preserving criterion to add new units. Oyang et al. (2005) constructed an RBF subnetwork for each class of training data to approximate the probability density function of that class. Jaiyen et al. (2010) presented a very fast learning method for classification problems that uses every datum just once and then can discard it. They used versatile elliptic function instead of traditional Gaussian activation function. By their proposed learning rules, RBF network has few neurons but its performance is limited.

In all those learning rules, there are tradeoffs between the number of hidden neurons, the generalization performance, and speed and complexity of learning rule. Networks with more hidden neurons perform better on learning data but may have less generalization ability. Learning rules with better generalization performance goals have to limit the number of hidden neurons and need to optimize the centers and spread of those limited nodes. Although there is a wide range of learning rules for RBF, it seems that the tradeoff between the generalization performance and the complexity of learning rule needs even more attention. Clearly, the learning rules presented in this paper would not solve this dilemma. However, the key point of this paper is to keep learning rule simple and fast, and at the same time the number of the hidden neurons limited by clever selection of RBF centers and spreads.

This paper presents new RBF learning rules for classification problems. The learning rules are one-pass heuristic rules with guaranteed linear separability property in feature space. That is, training data belonging to a class is linearly separable from those training data belonging to its complement set, in radial basis functions space. The training data can be learned with 100% accuracy without extreme increase in the number of neurons. The proposed learning methods are fast and simple, using efficient heuristics in their design. The learning rules set all hidden layer parameters, i.e. centers and spreads of neurons and the required number of neurons, simultaneously. They achieve least error and limited hidden neurons by selecting the neurons' centers and spreads in the way that every newly added unit has the highest coverage region of its own class, leading to a network with less complexity. Limiting the number of hidden neurons together with their near-optimal selection results in an increased generalization performance. The idea of adding new neurons based on maximum coverage achievable was first introduced in Javan, Rajabi Mashhadi, Ashkezari Toussi, and Rouhani (2012) and Javan, Rajabi Mashhadi, and Rouhani (2013) and successfully applied to applications in power systems. Here we extend the idea by two interpretations of maximum coverage of neurons: in terms of their spatial coverage and in terms of the number of data points covered by a neuron.

The rest of this paper is organized as follows. Section 2 presents the main ideas for determination of spreads and centers of hidden units together with learning of output layer parameters and structures. In Section 3, four theorems about the convergence of the learning algorithms, their computational complexity and their guaranteed linear separability property are proved. Section 4 is dedicated to simulation results of applying the proposed methods to some well-known datasets. In Section 5, the simulation results are compared with leading RBF learning rules and some SVM models. Section 6 summarizes the paper.

2. The proposed RBF learning rules

This section presents the main ideas of proposed RBF learning rules. Like the traditional RBF, the proposed RBF consists of a hidden layer of radial basis function neurons and an output layer. The parameters of hidden layer units, i.e. their spreads and centers, are set in the first phase and the parameters of the output layer (if there is any adjustable parameter in the output layer), can be set by traditional Least Mean Squares (LMS) method in the next phase. Two different methods are presented to learn the hidden layer. In both methods, the network has no hidden units at the beginning and the required neurons are added one by one. Every added unit is associated to one of output classes and covers a number of learning data points of its own class. The center and the spread of the unit are set according to the highest coverage possible. There are two heuristic approaches. First, we present the approach based on the selection of neuron with maximum spread, and then the approach based on the selection of neuron with maximum coverage of learning data. To guarantee the linear separability of learning patterns in feature space (i.e. hidden units output), the traditional Gaussian function is replaced by another symmetric "bell-shape" function. The output layer may have two different structures. In the first structure, the layer has a weight matrix learned by the well-known LMS method. The second structure assumes a winner-take-all (WTA) structure for output layer with no adjustable parameter, capable of accurate classification of all training data.

In the proposed RBF structure, each hidden neuron belongs to one of the output classes and represents some of the training data for that class by its activation domain. Among the main ideas of this paper is to choose the hidden neurons with the highest coverage domains, i.e. the ones with largest spread, in the hope that the required number of hidden neuron decreases. In other words, in each iteration of the proposed algorithms, the center and the spread of the new units are selected based on the largest coverage possibility. Here, the largest coverage may have two interpretations. First, in terms of the domain of coverage, the largest coverage leads to selection of the neuron with greatest spread. Second, in terms of training data, the largest coverage leads to selection of the neuron with the most number of data points in its domain. Those two interpretations result in two different learning algorithms, as discussed in this section. Selection of the neuron with largest coverage leads to smaller network, which indeed prevents it from overtraining. Another idea presented in this paper is to slightly modify the activation function of radial basis units to guarantee the separability of training data in RBs' space.

Boundary data have important rule in performance and accuracy of a classifier. Therefore, some approaches like support vector machines (SVMs) have special emphasis on boundary data. On the other hand, a boundary datum can simply be affected by noise and may lead to misclassification. In other words, higher

emphasis on the boundary data increases the sensitivity to noise. The proposed methods try to make a trade-off between those two conflicting concepts by using the boundary data in an indirect way. That is, in the classification problem, the spreads of the neurons of each class are determined by the nearest boundary datum of the other classes while its center is determined by a datum of its own class.

2.1. Rule 1: The learning rule based on the maximum spread

The learning of an RBF can be carried out in two phases. In the first phase, centers and spreads of all hidden units are specified, and then the weight matrix of linear output layer can be calculated. In this paper, some heuristics are used in the first stage to improve the performance of each neuron added to the network. As hidden units of RBF function locally (that is, each neuron is active in a hypersphere around its center and inactive elsewhere), the proposed learning rule tries to allocate the new neurons to have the largest activation region. This will decrease the number of required hidden neurons and consequently will increase the generalization performance. Each newly added unit belongs to one of the output classes and is active (as will be defined) for a number of training patterns of that class and inactive for all patterns from other classes. In the first learning rule presented in this subsection, the center of the new unit is placed at one of the training points and its spread is set equal to the distance of its center to the nearest pattern from opposite class (or classes) in order to limit the activation of the new unit to patterns of its own class. The center point is selected based on the maximum achievable spread. A unit with the largest spread results in more learning patterns covered by its activation domain, and all input patterns would be covered by fewer units. Furthermore, larger spreads make input–output mapping smoother and the generalization performance will increase. As we expect that the unit covers just the patterns of its own class, the maximum spread achievable is limited by the nearest pattern (i.e. a boundary datum) of the other classes. In fact, by selecting the maximum spread for neurons, the boundary data have an indirect effect on decision surface of the classifier. The data points near the boundary of two classes have the most information in the classification problems, however they can easily be affected by noise. This makes reliance on near boundary data (as in SVM model) more risky when the data may be noisy. By the way, in the proposed learning rules the effect of boundary data is in the determination of the spread of the neurons and as the spread is selected to have largest values, a small displacement of boundary datum has no significant influence on the overall performance of the network.

Below are the details of the first learning rule for specification of hidden layer parameters based on the maximum spread for newly added neurons:

Learning rule based on the maximum spread:

Suppose there are P training patterns $\{x_p\}$, $p = 1, \ldots, P$ each belongs to one of C classes

(1) Calculate data distances matrix as follows:

$$D = \left[\|x_i - x_j\|\right]_{P\times P} \quad i, j = 1, \ldots, P. \tag{1}$$

Consider the following sub-matrices of D:

$$\begin{aligned} R_c &= \left[\|x_i - x_j\|\right]_{P_c\times(P-P_c)}, \quad x_i \in \textit{Class } c,\ x_j \notin \textit{Class } c \\ Q_c &= \left[\|x_i - x_j\|\right]_{P_c\times P_c}, \quad x_i,\ x_j \in \textit{Class } c \end{aligned} \tag{2}$$

where, R_c is the distance matrix of class c data points from data of other classes and is obtained by keeping rows of matrix D corresponding to class c and deleting its corresponding columns. And Q_c is the matrix of distance of class c data points to each other, and is obtained by keeping rows and columns of matrix D corresponding to data in class c. P_c is the number of data in class c.

Let the number of neurons in the network be zero, $n = 0$.

(2) Consider the first class, $c = 1$.

(3) Calculate the minimum distance of each datum $x_i \in \textit{Class } c$ from all other data in other classes using corresponding row of R_c matrix:

$$r_c = \left[\min_j \|x_i - x_j\|\right]_{P_c\times 1}, \quad x_i \in \textit{Class } c,\ x_j \notin \textit{Class } c \tag{3}$$

where vector r_c measures the distance of all data in class c from other classes.

(4) Add a new neuron to hidden layer, set $n = n + 1$. The new neuron belongs to class c. Select the largest uncovered value of vector r_c as the spread of new neuron σ_n and the corresponding data as its C_n

$$\begin{aligned} &\sigma_n = \max_i r_{ci} = \max_i \left(\min_j \|x_i - x_j\|\right), \\ &\quad x_i \in \textit{Class } c,\ x_j \notin \textit{Class } c \\ &C_n = x_i, \quad i = \arg\max \left(\min_j \|x_i - x_j\|\right), \\ &\quad x_i \in \textit{Class } c,\ x_j \notin \textit{Class } c. \end{aligned} \tag{4}$$

(5) Consider the row of Q_c corresponding to datum x_i, let all data points of class c that their distances to center are smaller than the spread of new neuron as covered and delete corresponding rows and columns from R_c and Q_c.

(6) If there are uncovered data in class c go back to step 4. Otherwise if all classes are processed, that is if $c = C$, stop. If $c < C$, set $c = c + 1$ and go back to step 3. ■

As the processing of each class of data is carried out separately and the data points of other classes have no direct influence on the process, the order of the classes or the order of the data points has no effect on the results. The proposed learning rule for hidden layer has no adjustable parameter and is not stochastic. Therefore, the design of an RBF needs no parameter adjustment and different executions of the algorithm have the same results. It is worth mentioning that the learning rule is based on a heuristic aiming at increasing generalization performance by limiting the number of hidden neurons. There is no optimization performed and thus the result is expected to be near optimal but not necessarily optimal. We avoid optimization algorithms to prevent convergence problems and speed up the algorithm.

Fig. 1 describes the algorithm by classification of a two class artificial dataset. As shown in Fig. 1(a), for the first neuron of the first class, the point a1 (at the center of the circle) is selected as the center of the first neuron, as it has the largest distance from other class. The distance of this datum from other classes (R_{a1}) is set as the spread of neuron. Then, all data of class 1 covered by the new neuron are discarded from further developments. The same procedure is repeated for the rest of data of class one as in Fig. 1(b). In this example, this was done by three neurons, and we have $R_{a1} > R_{a2} > R_{a3}$.

After all data belonging to class 1 are processed, the learning procedure is repeated for class 2 data and in its first iteration the datum of class 2 with largest distance from class 1 data is selected as the center of new neuron belongs to class 2. The covered data from class 2 are discarded Fig. 1(c) and the procedure goes forward for the rest of the data from class 2 Fig. 1(d). Again, the data from class 2 are covered completely by three hidden neurons. Each datum is covered by a neuron of its own class and the spreads of the neurons are the maximum achievable.

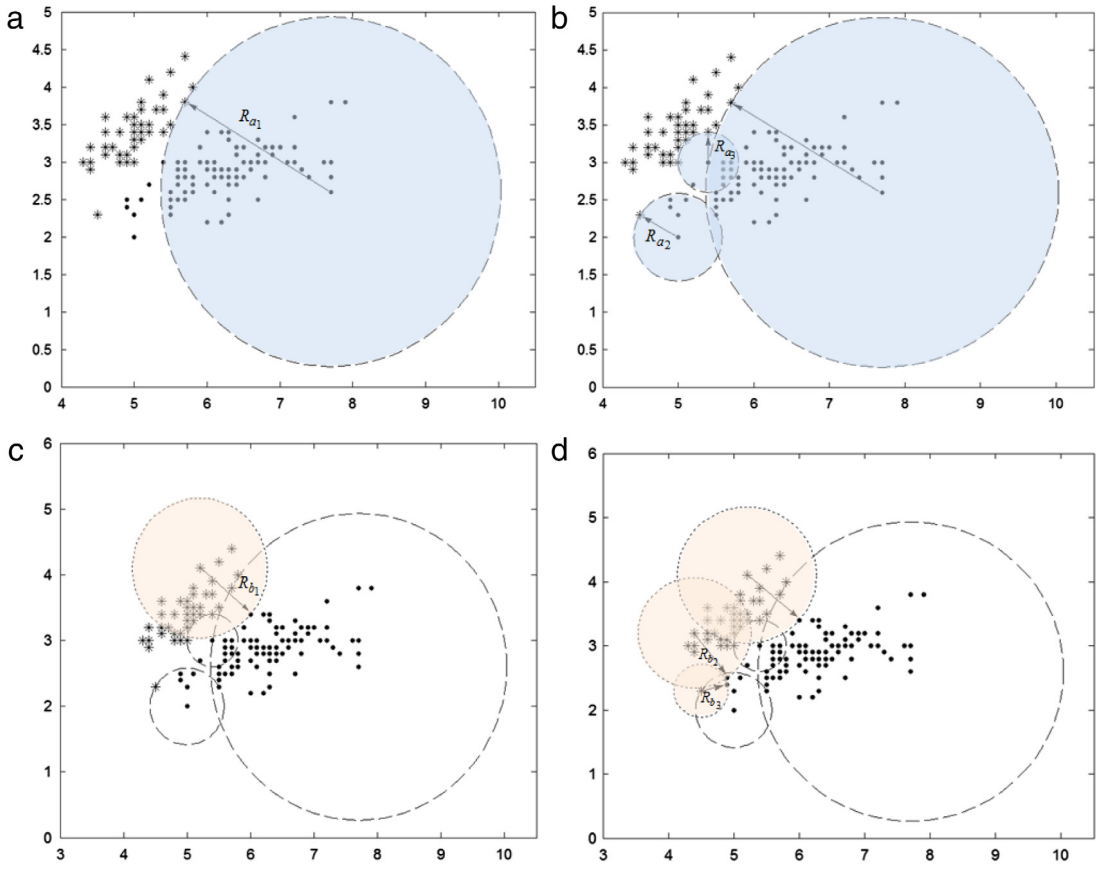

Fig. 1. Classification of an artificial two-dimensional dataset by learning rule based on the maximum spread. See text for description.

2.2. Rule 2: The learning rule based on the maximum data coverage

The second algorithm proposed in this paper is based on another simple heuristic idea. Here, the center of newly added neuron is selected based on the maximization of the number of data which is covered by that neuron. Again, more data covered by each neuron means less neurons are needed for coverage of the entire dataset and thus, higher generalization performance is hopeful.

Learning rule based on the maximum coverage:

Suppose there are P training patterns $\{x_p\}$, $p = 1, \ldots, P$ each belongs to one of C classes

(1) Calculate data distances matrix as follows:

$$D = \left[\|x_i - x_j\|\right]_{P\times P} \quad i, j = 1, \ldots, P. \tag{1}$$

Consider following sub-matrices of D:

$$\begin{aligned} R_c &= \left[\|x_i - x_j\|\right]_{P_c\times(P-P_c)}, \quad x_i \in \textit{Class } c, x_j \notin \textit{Class } c \\ Q_c &= \left[\|x_i - x_j\|\right]_{P_c\times P_c}, \quad x_i, x_j \in \textit{Class } c \end{aligned} \tag{2}$$

where R_c is the distance matrix of class c data points from data of other classes and is obtained by keeping rows of matrix D corresponding to class c and deleting its corresponding columns. And Q_c is the matrix of distance of class c data points to each other and is obtained by keeping rows and columns of matrix D corresponding to data in class c. P_c is the number of data in class c.

Let the number of neurons in the network be zero, $n = 0$.

(2) Consider the first class, $c = 1$.

(3) Calculate the minimum distance of each datum $x_i \in$ *Class* c from all other data in other classes using corresponding row of matrix R_c:

$$r_c = \left[\min_j \|x_i - x_j\|\right]_{P_c\times 1} \quad x_i \in \textit{Class } c,\ x_j \notin \textit{Class } c \tag{3}$$

where, vector r_c measures the distance of all data in class c from other classes.

(4) Count the number of class c data that can be covered by datum x_i:

$$n_c = \left[\operatorname*{count}_j Q_{ij} = \|x_i - x_j\| < r_{ci}\right]_{P_c\times 1}$$
$$x_i, x_j \in \textit{Class } c. \tag{5}$$

(5) Add a new neuron to hidden layer, set $n = n + 1$. The new neuron belongs to class c. Select the datum x_i corresponding to largest value of n_c vector as its center C_n and set the spread of the new neuron σ_n to the corresponding value of data vector r_c.

$$C_n = x_i, \quad i = \arg\max(n_{ci}), \text{ and } x_i \in \textit{Class } c$$
$$\sigma_n = r_{ci}.$$

(6) Consider the row of Q_c corresponding to datum x_i, let all data points of class c that their distances to the center are smaller than spread of new neuron as covered and remove corresponding rows and columns from R_c and Q_c.

(7) If there are uncovered data in class c go back to step 4. Otherwise if all classes are processed $c = C$ stop. If $c < C$, set $c = c + 1$ and go back to step 3. ■

Both algorithms based on the maximum spread and the maximum coverage try to cover all data in a class by the minimum number of

hidden neurons. As will be discussed in the following subsection, the selection of spread of each neuron is based on keeping a pre-specified level of activation of that neuron for data in its own class, in a way that each neuron can be regarded as active in its own spread range and inactive outside it.

2.3. Activation function

As mentioned before, two proposed methods are based on the idea of local activation of each neuron for data point of its own class. That means we expect the hidden neurons to be active (i.e. its output must be greater than a specified level λ_u, $0 < \lambda_u < 1$) for some of training patterns of its own class and be inactive (i.e. its output must be less than a specified level λ_l, $0 < \lambda_l < \lambda_u$) for all patterns of other classes. The well-known Gaussian activation function of RB neurons is as follows:

$$o(r) = \mathrm{e}^{-\frac{r^2}{2\sigma_n^2}} \tag{6a}$$

in which, σ_n is the spread of nth neuron and $r = \|x - c_n\|$ is the distance of point x from the center of neuron c_n. As Gaussian function has just one parameter σ, it can satisfy only one of the two mentioned requirements. That is, one may set σ in order to let output of the neuron be less than λ_l for nearest datum of other classes, or it may be set in order to let output of the neuron be greater than or equal to λ_u for the farthest datum of its own covered class. To achieve both conditions, we need a function with at least two parameters. Liu and Bozdogan (2003) proposed power exponential distribution function

$$o(r) = \mathrm{e}^{-\frac{r^\alpha}{\alpha\sigma_n^\alpha}} \tag{6b}$$

as activation function of an RBF neuron which has additional parameter α. We use the same activation function as in (6b), but reformulate it slightly:

$$o(r) = \mathrm{e}^{\log(\lambda_l)\left(\frac{r^2}{\sigma_n^2}\right)^{\alpha_n}} \tag{7}$$

which has two parameters α_n and σ_n. The output of this function is 1 at the center of the neuron ($r = 0$) and at distances equal to or greater than its spread ($r \geq \sigma_n$) the output is equal to or less than λ_l, regardless of the value of α_n. If we set $\lambda_l = \mathrm{e}^{-\frac{1}{2}} \approx 0.6065$ and $\alpha_n = 1$, the above function (7) is equivalent to Gaussian function in (6). Now, α_n can be set according to the requirement that activation level of the neuron must not be less than λ_u for the farthest covered datum. So we have:

$$\alpha_n = \max\left(1, \frac{\log\left(\frac{\log(\lambda_u)}{\log(\lambda_l)}\right)}{2\log\left(\frac{r_{\max}}{\sigma_n}\right)}\right) \tag{8}$$

where, $r_{\max}$ is the distance of the farthest datum covered by the neuron. Hence, each hidden neuron has three parameters: a vector of its center c_n, and two scalars of its spread σ_n and power α_n. Eq. (8) implies that by setting $\alpha_i = 1$ the output of neuron at $r_{\max}$ is greater than λ_u, and there is no need to modify activation function. Fig. 2 shows the proposed modified activation function.

2.4. Weights of output layer

Normally, the output layer of an RBF network may have linear neurons for regression (function approximation) applications and hard limit ones for classification applications. As our concern in this paper is classification problems, we consider the output layer to consist of C hard limit neurons:

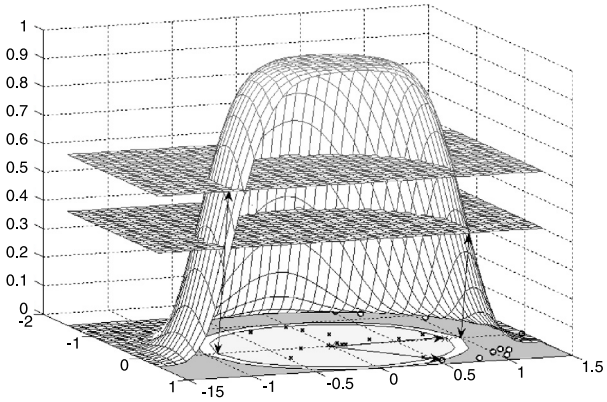

Fig. 2. The modified Gaussian like activation function. Data belongs to a specified class and other classes are marked with "x" and "o", respectively. $\lambda_l = 0.4$, $\lambda_u = 0.6$, $\sigma = 1$, and $r_{\max} = 0.9$.

$$\acute{y}_c = \sum_{n=1}^{N} w_{cn} O_n + w_{c(N+1)} \quad c = 1, \ldots, C \tag{9}$$

$$y_c = \begin{cases} 1 & \text{if } \acute{y}_c > 0 \\ -1 & \text{if } \acute{y}_c \leq 0 \end{cases} \quad c = 1, \ldots, C \tag{10}$$

where, O_n is the output of the nth neuron in hidden layer and $W_{C(N+1)}$ is the matrix of weights and biases. Desired values for each of P learning patterns are defined as follows:

$$\underline{d}_p = [d_{p1} d_{p2} \ldots \acute{d}_{pC}] \tag{11}$$

$$d_{pc} = \begin{cases} 1 & x_p \in \text{class } c \\ -1 & \text{otherwise} \end{cases} \quad c = 1, \ldots, C \text{ and}$$

$$p = 1, \ldots, P. \tag{12}$$

Training of output layer weights and biases are carried out in traditional way by minimization of a sum of square error criteria. After learning of hidden layer by means of each of the two proposed learning rules, all data points x_p, $p = 1, \ldots, P$ are presented to hidden layer and the corresponding hidden layer output O_p is calculated. The matrices D and O are defined as:

$$D = [\underline{d}_1 \underline{d}_2 \ldots \underline{d}_P] \tag{13}$$

$$O_{(N+1)P} = \begin{bmatrix} \underline{O}_1 & \cdots & \underline{O}_P \\ 1 & \cdots & 1 \end{bmatrix}. \tag{14}$$

The ones are added in the last row of matrix O to represent the multipliers of biases. The weight matrix could be calculated directly as:

$$W_{C*(N+1)} = D * O' * (O * O')^{-1}. \tag{15}$$

2.5. Winner-take-all (WTA) output layer

Here, a novel structure for output layer of RBF for classification problems is presented as a winner-take-all (WTA) layer without any adjustable parameter. Here again, there are C output neurons, each associated with one of the C classes of data. The output associated with class c is one, if and only if one of the hidden neurons associated with class c has the highest output among all hidden neurons, all other output neurons are set to minus one (or zero). As it will be proved later, it is guaranteed that all training patterns would be classified correctly, i.e. the learning accuracy is 100%. As there is no adjustable parameter, there is no need

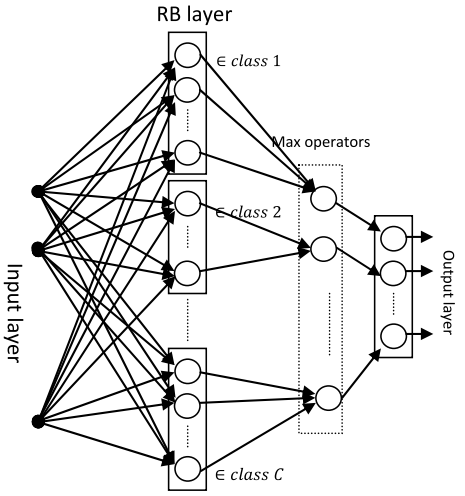

Fig. 3. The RBF structure with winner-take-all output layer.

for learning. Fig. 3 demonstrates the structure of proposed RBF network with WTA output layer. The output calculation can be summarized as follows:

$$\acute{y}_c = \max_n (O_n) \quad O_n \in \text{class } c \tag{16}$$

$$y_c = \begin{cases} 1 & \text{if } \acute{y}_c = \max_n \acute{y}_n \\ -1 & \text{otherwise.} \end{cases} \tag{17}$$

2.6. Learning rule based on the maximum spread and limited neurons

The proposed learning rules attempt to select fewer neurons by the simple heuristic rules, and simulation results show the number of neurons is actually limited in comparison with traditional RBF and the number of support vectors in SVM models. However, one can slightly modify each of the two proposed methods to limit the maximum number of neurons of the network. When a maximum number of neurons is specified, the only required modification to the previous rules is to add neurons in decreasing order of their spread and regardless of the order of the classes. In other words, in the algorithm of Section 2.1 neurons that belong to each class are added together in order of their spreads.

Learning rule based on the maximum spread and limited neurons

Suppose there are P training patterns $\{x_p\}$, $p = 1, \ldots, P$ each belongs to one of C classes

(1) Calculate data distances matrix as follows:

$$D = \left[\|x_i - x_j\|\right]_{P\times P} \quad i, j = 1, \ldots, P.$$

Consider the P valued vector R with components r_i as:

$$r_i = \max_j \|x_i - x_j\| \ , \quad x_i \in \textit{Class } c, \ x_j \notin \textit{Class } c. \tag{18}$$

Therefore elements of vector R are the distances of each datum to other classes.

Let the number of neurons of the network be zero $n = 0$.

(2) Find the largest element $r_i = \max_j r_j$ of R and the corresponding datum $x_i \in$ *Class* c. Add a neuron to network $n = n + 1$ and let the center of new neuron as $C_n = x_i$ and its spread as $\sigma_n = r_i$.

(3) Find the data belong to class c that are closer to neuron's center and mark them as covered.

(4) Stop if all data are covered or the maximum of neurons is achieved, otherwise go back to step 2. ■

A similar modification can be made to algorithm in Section 2.2.

3. Analysis of proposed algorithms

Proposed algorithms in Sections 2.1, 2.2 and 2.6 are clearly convergent after a limited number of iterations. In fact, in each iteration, a new neuron is added and covered data by it is discarded, and the algorithm is repeated for remaining data. At least one datum, the one which is selected as the center of the neuron will be discarded. This could be summarized as follows:

Theorem 1 (*Convergence*)**.** *Algorithms based on the maximum spread and the maximum coverage converge in limited iterations, not more than the size of training data P. The learned RBF networks have at most P hidden neurons. The algorithm based on the maximum spread and limited neurons (Section 2.6) converges in limited iterations, not more than* $N_{\max} \leq P$ *and the resulting network has at most* $N_{\max}$ *neurons.* ■

Now, we prove that for presented algorithms in Sections 2.1 and 2.2 the values of λ_l and λ_u could always be specified to have zero classification error for training patterns of RBF network with hard-limit output layer. For proposed RBF network in Section 2.5 with WTA output layer (see Fig. 3), this is true for all values of λ_l and λ_u, that $0 < \lambda_l < \lambda_u < 1$. The following theorem can be stated:

Theorem 2 (*Linear Separability*)**.** *Consider the RBF network which is learned by each of the algorithms based on the maximum spread or the maximum coverage with the modified Gaussian-like activation function (7). Each two-set partitions of learning classes are linearly separable in N-dimensional space of hidden layer neurons (N is the number of hidden neurons) as* $\lambda_u < 1$ *is large enough and* $0 < \lambda_l$ *is small enough.* ■

One of the results of this theorem is that, each output neuron can be learned to separate each subset of output classes from its complement subset. A special case would be when there is an output neuron for each output class (a network with C outputs) and each neuron is supposed to represent a single class, i.e. to separate that class from other classes. The theorem states that for trained RBF using the proposed algorithms and activation function in (7), one can always find output weights and biases values that minimize training error to zero. Another special case is using the so called coded output. For example, let $C = 2^3 = 8$, then the network may have just 3 output neurons instead of eight neurons and the first neuron is supposed to separate classes $\{1, 2, 3, 4\}$ from classes $\{5, 6, 7, 8\}$, the second has to do the same for classes $\{1, 2, 5, 6\}$ from $\{3, 4, 7, 8\}$ and the last neuron has to separate classes $\{1, 3, 5, 7\}$ from $\{2, 4, 6, 8\}$. Again, the theorem guarantees that there are weights and biases for those output neurons which result in 100% classification accuracy for training patterns.

Proof. Suppose that each training datum belongs to a set Ω_c, $c = 1, \ldots, C$. $M = \{1, 2, \ldots, C\}$ is the set of all training data class indices. A two-set partition of the input classes can be represented by two subsets $\Omega = \bigcup_{c\in A} \Omega_c$ (where $A \subset M$ is the set of the first partition indices) and its complement $\overline{\Omega} = \bigcup_{c\in\bar{A}} \Omega_c$. We prove that if λ_u is large enough and λ_l is small enough, there exists a separating hyperplane

$$f(x) = \sum_{n=1}^{N} w_n O_n(x) - b \tag{19}$$

that separate training data belonging to Ω from those belonging to $\overline{\Omega}$. To prove this, it is sufficient to determine appropriate values for λ_u, λ_l, weights w_n, and bias b.

We define weights as follows:

$$w_n = \begin{cases} 1 & \text{if } O_n \in \text{class } c \text{ and } c \in A \\ 0 & \text{otherwise} \end{cases}, \quad n = 1, \ldots, N. \tag{20}$$

To determine bias, recall that for each training datum $x_p \in \Omega_c \subset \Omega$ at least one of the hidden neurons $O_n \in$ class c has output greater than λ_u. Then by choosing a bias value that satisfies:

$$b < \lambda_u \tag{21}$$

we get

$$f(x_p) = \sum_{O_n \in \text{class } c} O_n(x_p) - b > 0, \quad x_p \in \Omega. \tag{22}$$

In addition, for all data points not belonging to Ω, it is required to have:

$$f(x_p) < 0, \quad x_p \in \overline{\Omega}. \tag{23}$$

As learning rule ensures that for all data points $x_p \in \overline{\Omega}$ the neurons with weights $w_n = 1$ (that is hidden neurons belong to Ω) have maximum outputs of λ_l, Eq. (23) is satisfied if:

$$\lambda_l \sum_{n=1}^{N} w_n < b. \tag{24}$$

It means that λ_u and λ_l need to be chosen according to

$$0 < \lambda_l \left(\sum_{n=1}^{N} w_n \right) < \lambda_u < 1. \tag{25}$$

In other words, let $\lambda_u < 1$ be large enough and $\lambda_l > 0$ be small enough, the bias term could be set to satisfy both (24) and (21).

If the network is required to perform more than one partition, the above condition (25) must be replaced by

$$0 < \lambda_l \max_d \left(\sum_{n=1}^{N} w_{nd} \right) < \lambda_u < 1. \quad \blacksquare \tag{26}$$

For the learning algorithm based on the maximum spread and limited neurons (see Section 2.6), the result of this theorem is valid for those training patterns that are covered by hidden neurons. Although, Theorem 2 says nothing about classification accuracy of test or validation patterns, we expect that by increasing the separability of learning patterns in hidden layer space, the separability of test and validation patterns increase too, which indeed leads to better generalization performance. The proof of theorem suggests a learning method for output layer neurons by (20), (21), and (24). However we do not use those weights in real applications as they are not the unique solution. In fact, (25) is a conservative condition and one may find different weights and biases yield to 100% accuracy without satisfying (25). Besides this, as the following theorem implies, the WTA output layer has zero classification error for a wider range of parameters λ_u and λ_l. Therefore, if zero error is favorable, the WTA output layer is preferred.

Theorem 3 (*Training Accuracy of WTA Layer*)**.** *For an RBF network learned by each of the algorithms based on the maximum spread or the maximum coverage with modified Gaussian-like activation function (7), and with WTA layer as described by (16) and (17) in Section 2.5, training error is zero provided that $\lambda_l < \lambda_u$.*

Proof. For each training datum $x_p \in \Omega_c$ one of the RB neurons associated with class c ($O_n \in$ class c) has activation level greater than λ_u and all RB neurons associated with other classes have activation levels equal to or smaller than λ_l. Therefore if $\lambda_l < \lambda_u$, a hidden neuron belonging to class c is the winner (has the maximum output) and the datum is classified correctly.

Theorem 4 (*Computational Complexity*)**.** *The computational complexity of learning rules based on the maximum spread, the maximum coverage, and the maximum spread and limited neurons are of order two $O(P^2)$ where, P is the number of data.*

Proof. We present the proof for algorithm based on the maximum spread and limited neurons, for other two algorithms the proof is similar. In the first step of the algorithm, a $P \times P$ matrix D is calculated with computational cost of $O(P^2)$. In addition, the P-valued vector R is computed by comparing a maximum of $P - 1$ elements and finding their maximum. This has computational cost of $O(P^2)$, too. The first step is not repeated and its total computational cost is $O(P^2)$. In a loop, steps 2–4 are repeated P times at most. Step 2 finds the maximum value of vector R. At the first iteration, R has P values that is decreasing in each iteration. Thus, computation cost of each iteration of step two is $O(P)$ and its overall cost is $O(P^2)$, as it may be reiterated P times. At step 3, a P-valued row of matrix Q_c is compared with spread of the new neuron. Again, computational cost of each iteration of step 3 is $O(P)$ and its overall cost is $O(P^2)$. Step 4 has no dependence on data size, i.e. its overall computational cost is $O(P)$. Therefore, the computational cost of the whole algorithm is of order two. $\blacksquare$

4. Simulation results

Section 2 presented the main ideas for design and learning of an RBF network. Different RBF architectures could be composed by those algorithms. In this section, performances of the three RBF architectures are evaluated on well-known datasets. Those three architectures are as follows:

- RBF-R: the network has two layers. Hidden layer neurons are trained by learning rule based on the maximum spread (see Section 2.1). The activation functions of hidden neurons are modified Gaussians as was represented in Eq. (7). The second layer has C hardlimit neurons and its weights and biases matrix is set by Eq. (15).
- RBF-N: the network has two layers. Hidden layer neurons are trained by learning rule based on the maximum coverage (see Section 2.2). The activation functions of hidden neurons are modified Gaussians as was represented in Eq. (7). The second layer has C hardlimit neurons and its weights and biases matrix is set by Eq. (15).
- RBF-WTA: the network has two layers. Hidden layer neurons are trained by learning rule based on the maximum spread (see Section 2.1). The activation function is same as RBF-R. The second layer has WTA structure that was presented in Section 2.5 and defined by Eqs. (16) and (17).

The number of neurons in all three proposed network architectures can be limited by the presented method in Section 2.6.

To investigate their performances, proposed networks are verified by well-known datasets and the results are discussed here and will be compared with other methods in the next section. Datasets are chosen to have different number of data, features and classes. An effort is given to choose well known datasets. Table 1 shows datasets that have been used in this study, all from UCI data base (Frank & Asuncion, 2010).

Example 1. The purpose of this example is to analyse the effect of the only parameters of proposed architectures, λ_u and λ_l. A medium size dataset (i.e. Ionosphere) is selected for this example. At each run, 80% of data is used for training and the rest for test. RBF-R and RBF-WTA networks were trained for different values of λ_u and λ_l. The result for RBF-N network was similar to RBF-R network and we omit it here. λ_u increased from 0.1 to 0.9 and λ_l from 0.05 to $\lambda_u - 0.05$ (as it must be smaller than λ_u) both

Table 1
The properties of datasets for examples.

	Dataset	Data size	Features	Classes
1	Thyroid	215	5	3
2	Heart	270	13	2
3	Ionosphere	351	34	2
4	Breast cancer	699	9	2
5	PID	768	8	2
6	Segmentation	2310	19	7
7	Spambase	4601	57	2
8	Wine quality-white	4898	11	7

by 0.05 steps. Fig. 4 shows the results of train and test errors for both networks. Fig. 4(a) shows train error for RBF-R network. The minimum training error of 0.84% is obtained for $\lambda_u = 0.9$ and $\lambda_l = 0.4$. Generally, the training error decreases as the difference between λ_u and λ_l increases. This fact is consistent with Theorem 2 that states the train error can approach zero if λ_u is high enough and λ_l low enough, though the output layer is trained by LS method and not by suggested method of Theorem 2 proof. However, for $\lambda_u > 0.7$ and $\lambda_l < 0.2$, there is a slight increase in training error which can be explained by more local functions of hidden neurons as a consequence of high values for α_n (see Eq. (7)) parameters (see (7)). Testing error of RBF-R network is plotted versus λ_u and λ_l in Fig. 4(b). Minimum testing error of 4.7% is obtained with $\lambda_l = 0.4$ and $\lambda_u = 0.6$. Training error was 1.2% for the same parameters. In fact, those values result the best performance regarding the generalization accuracy. It is clear from this figure that test error increases for lower values of λ_l, which can be again explained by more local functions of hidden neurons as a consequence of high values for α_n parameters. For values $0.4 < \lambda_l < 0.65$ and a little greater λ_u, generalization accuracy is at its maximum and is not sensitive to exact values of those parameters. Fig. 4(c) demonstrates generalization performance of RBF-WTA network. From this figure, One can realize that the generalization performance of RBF-WTA is not sensitive to exact values of λ_u and λ_l. As it was proved by Theorem 3, RBF-WTA network has zero training error for all sets of parameters if $\lambda_l < \lambda_u$, and therefore the result is not shown.

As Example 1 conducts, the exact values of λ_u and λ_l have no significant effect on the results. For all simulations hereafter, those values are selected as $\lambda_u = 0.7$ and $\lambda_l = 0.6065$.

Example 2. In this example, the performances of RBF-N and RBF-WTA networks with limited and predefined number of hidden neurons, as it was proposed by the algorithm in Section 2.6, are studied. The learning algorithms based on the maximum spread and the maximum coverage add as much neurons to the hidden layer to cover all learning data. However, the best generalization performance may be achieved by even less hidden neurons. For this example a large dataset, namely Spambase dataset, is chosen and 80% of data is used for training and 20% for testing in each run. For this dataset, the required number of neurons to cover all train data is 616 neurons. Fig. 5 shows training and testing errors as the number of hidden neurons increases. Figs. 5(a) and (b) show the result for RBF-N and RBF-WTA networks, respectively. As expected, train error of RBF-WTA reaches to zero once the number of neurons reaches the maximum of 616. Both networks have approximately the same levels of training and testing errors for low and medium network sizes, which means there is no overtraining. Train and test errors of RBF-N network are a little less than RBF-WTA, although this is not true for all datasets as we see later. For neurons more than about half of the maximum of required number, the train errors decrease without decrease in test errors which indicates the overtraining of the networks.

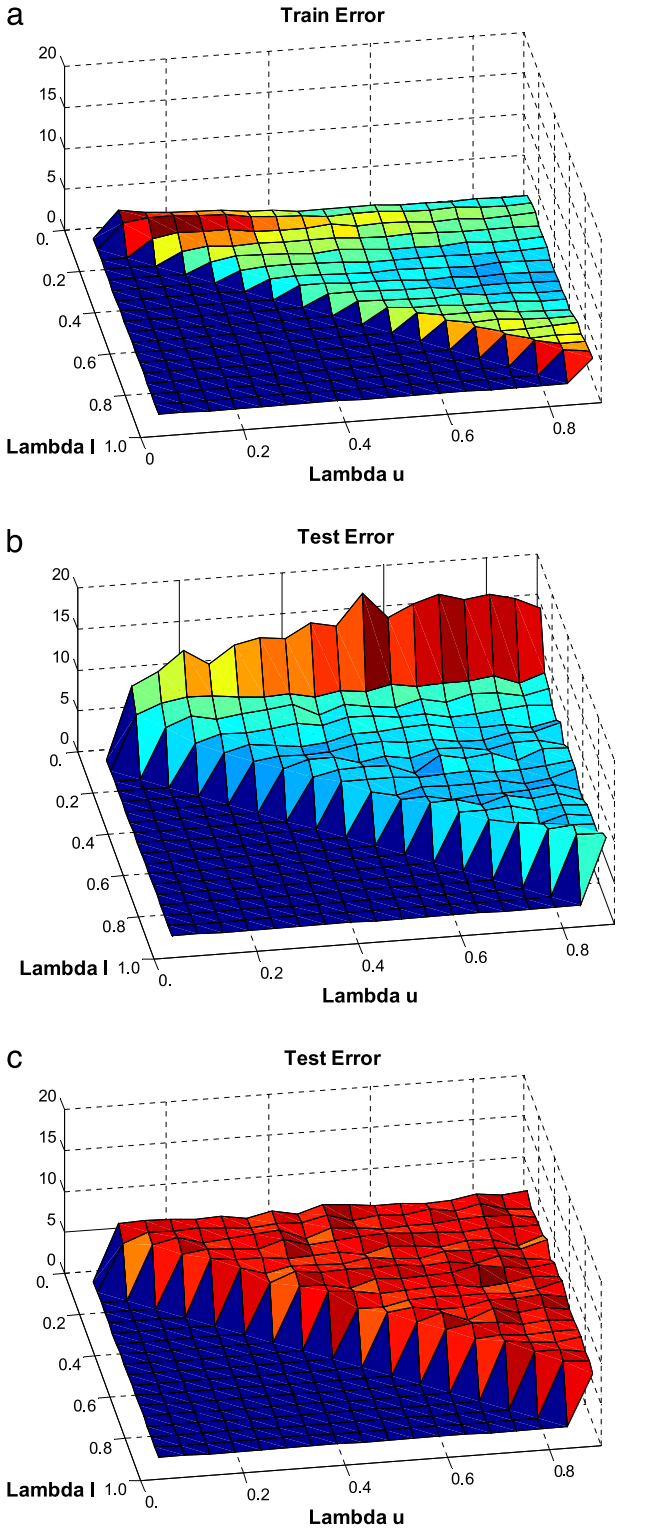

Fig. 4. Test and train errors (in percentages) of proposed algorithms versus parameters of activation function, (a) train error of RBF-R network, (b) test error of RBF-R network, and (c) test error of RBF-WTA network.

Example 3. Test errors of RBF-WTA network for some datasets are presented in Fig. 6. The datasets are PID, Ionosphere, Heart, and Spambase. Since the required numbers of neurons for each of these datasets are different, the horizontal axis is normalized as the ratio of neurons to the maximum of required neurons for each dataset. The vertical axis indicates test error for each of four datasets. The results for RBF-N and RBF-R networks are similar to RBF-WTA.

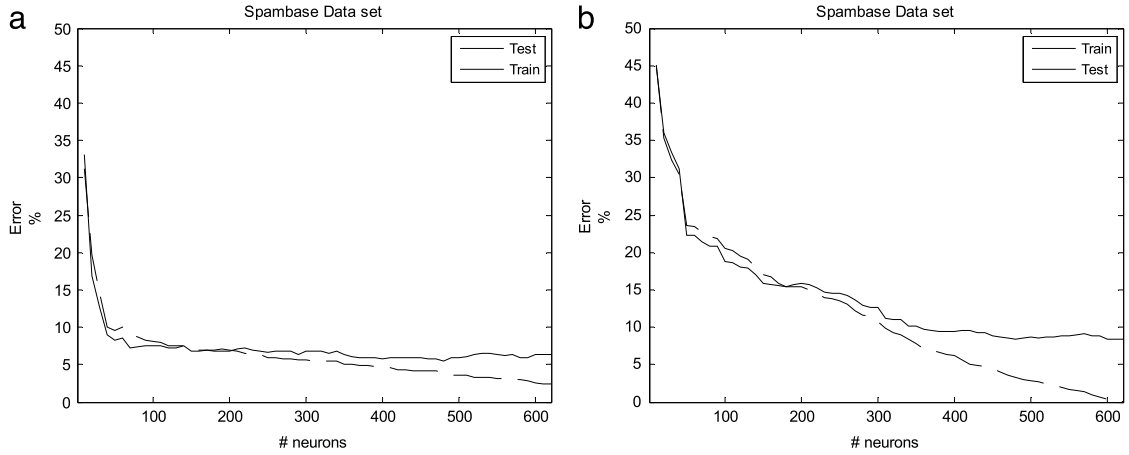

Fig. 5. Test and train errors of RBF-N (left) and RBF-WTA (right) networks for Spambase data versus the number of hidden neurons.

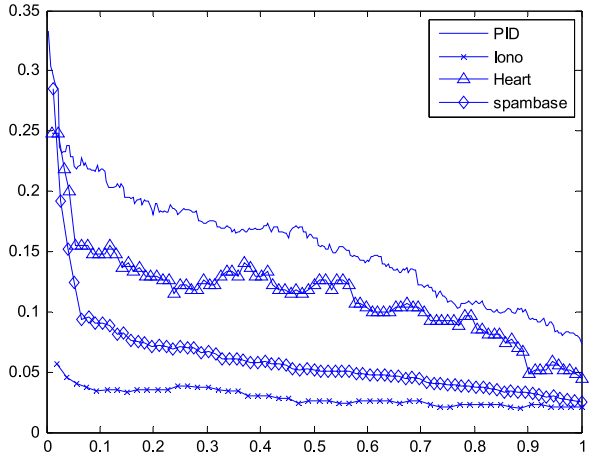

Fig. 6. Test error of RBF-WTA structure as the number of hidden neurons increases.

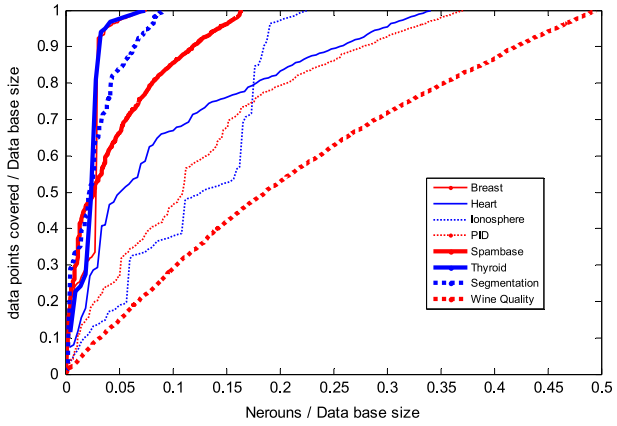

Fig. 7. The ratio of data covered by hidden neurons. The horizontal axis is the ratio of hidden neurons to data size. When all data is covered, the learning algorithm stops.

For Ionosphere dataset, the test error is always below 5% and reaches its minimum for about one-tenth of maximum neurons whereas for PID dataset test error begins with 33% and decreases continuously. For Heart and Spambase datasets, test error has no significant reduction as the number of neurons increases over 30% of its maximum number. Summing up, for three of those four datasets, the optimal number of neurons is much less than the number of neurons which is required to cover all data.

Example 4. Fig. 7 shows data coverage by hidden neurons for all datasets of Table 1. The horizontal axis of this figure is the ratio of hidden neurons to data size. The vertical axis shows the ratio of data points covered by learning rule based on the maximum spread as new neurons are added. For five datasets, the number of required neurons to cover all data points is less than a quarter of data size, indicating that the zero error can be achieved with much less number of neurons than data size (in RBF-WTA network). For other three datasets (Heart, PID, and Wine Quality) the required number of hidden neurons are a little more, i.e. up to 49% of data size for Wine Quality.

5. Comparison to other methods

In this section, results of implementation of proposed methods are compared with some other leading RBF learning methods for classification problems and some support vector machines (SVM). All implementations are carried out by MatLab software. MatLab implements RBF training by newrb.m command which is slightly different from traditional RBF as it selects the most erroneous datum as the center of new neuron (Neural Network Toolbox User's Guide, 2012). Traditional support vector machine is implemented by svmtrain.m command. Simulation results for two other SVM methods are presented, too. Sequential Minimal Optimization (SMO) is an efficient and fast solver for SVM (Platt, 1998). LIB-SVM (Chang & Lin, 2011) is a library toolbox which consists of different efficient SVM methods capable of solving two-class as well as multi-class problems. The results are compared with three other RBF learning methods for classification problem (Constantinopoulos & Likas, 2006; Jaiyen et al., 2010; Titsias & Likas, 2001). Those three models will be refereed here after as C2006, T2001, and J2010, respectively. The 5-fold cross validation is used for all simulations and the results are averaged over 10–20 runs for each dataset. No effort is taken to optimize the values of λ_u and λ_l parameters, but the number of neurons is optimized.

Table 2 presents the results of testing error for proposed networks RBF-R, RBF-N, and RBF-WTA together with the result of other methods. Among eight datasets, in five cases one of the proposed learning rules has the best performance. RBF-WTA network has better generalization as compared with other two proposed methods of this paper. For two-class Heart problem, SVM and LIBSVM methods perform better than other methods and reference T2001 had lowest error for PID dataset. For Spambase dataset SMO-SVM performs slightly better than RBF-R. For this two-class data, the SVM fails due to huge memory usage (out of memory error). As seen from this table, the proposed algorithms have comparable and satisfactory results as compared with other RBF learning rules and SVM methods.

Table 2
Averaged 5-fold-validation test error (%).

	RBF-R	RBF-N	RBF-WTA	SVM	LIBSVM	SMOSVM	RBF	J2010	T2001	C2006
Thyroid	4.4	5.3	**3.7**		4.18		9.8			4.6
Heart	18.1	19.5	19.4	**16.4**	16.67	18.1	21.1	26		
Ionosphere	**4.5**	4.8	5.7	10.5	6.25	6.3	12.9		8.27	
Breast cancer	3.7	3.6	**3.0**	3.3	3.3	3.1	6.7			
PID	24.7	27.9	26.2	26.2	23.82	25.9	29.8		**23.22**	24.1
Segmentation	6.4	5.7	**5.4**		6.2		35.1	21.4		
Spambase	7	6.7	9.5	Out of memory	6.6	**6.33**	7.26	19		
Wine quality	41.6	43.9	**35.9**		42.7		45.6			

Table 3
Model complexity, (average number of hidden neurons for RBFs, and number of SV's for SVM models).

	RBF-R	RBF-N	RBF-WTA	SVM	LIBSVM	SMOSVM	RBF	J2010	T2001	C2006
Thyroid	15.1	18.7	14.6		58.8		43			**3.8**
Heart	24	27	46	216	124	88.8	54	**3.8**		
Ionosphere	65	48	66.6	280.8	105.2	58	70		**12**	
Breast cancer	40	**35**	40	527.9	90.2	46.1	140			
PID	120	264	160	614.4	356.2	347.6	154		14	**4.3**
Segmentation	200	241	200		632.8		462	**11.8**		
Spambase	619.8	699	619.8		1070.8	684.3	460	**3.8**		
Wine quality	2038	**2000**	2040		3498.4		2490			

In addition to generalization performance, the proposed algorithms are compared with other methods based on model complexity, too. For SVM methods model complexity normally is measured as the number of SV's, and for RBF models as the numbers of their hidden neurons. The numbers of hidden neurons or the numbers of SV's are given in Table 3. By considering the number of hidden units, three methods C2006, T2001, and J2010 have less model complexity as compared with proposed methods for all datasets. However, their accuracies are lower than our methods except for PID dataset. Regarding the model complexity, the proposed RBF networks are normally at the middle of those for RBF networks and SVM models. For Breast Cancer dataset, the proposed methods have the best performance regarding the accuracy and complexity.

For further investigation of the proposed learning rules, those rules are compared with five neural network models as reported in Fernández-Delgado, Cernadas, Barro, Ribeiro, and Neves (2014). Table 4 presents the specification of 15 small and medium size datasets used in their study and is adopted here for comparison. Fernández-Delgado et al. (2014) reports the results of their own DKP model. Table 5 summarizes the results. The first three columns are associated with three RBF models proposed in this paper, namely RBF-R, RBF-N, and RBF-WTA. For each dataset, 20 runs are averaged. The rest of columns are adopted from Fernández-Delgado et al. (2014). Among those 8 algorithms, RBF-R wins the most (has the best performance). RBF-R has the best average accuracy (%84.8) on those 15 datasets, followed by RBF-N (%84.2). The average rank of each classifier on all 15 datasets is reported, too. Again, RBF-R has the best average rank and is followed by RBF-N while RBF-WTA and DKP are the next. Based on both average accuracy and average rank, the RBF-R is the best and the RBF-N is the second best algorithm.

Statistical tests suggested in Demšar (2006) are used to further investigate the results. For comparison of multiple classifiers, Demšar suggests the computation of critical difference (CD) with the Bonferroni–Dunn test. For N datasets and k classification algorithms (here $k = 8$ and $N = 15$), the Bonferroni–Dunn test indicates a significantly different performance of two classifiers if the corresponding average ranks differ by at least the critical difference

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \tag{27}$$

where q_α is critical value for a confidence level of α (here, for $\alpha = 0.10$ the critical value is $q_\alpha = 2.450$ and $CD = 2.1$). The average rank of RBF-R is 3.13 and the Bonferroni–Dunn test indicates that its performance is meaningfully better than RBF, MLP and KNN. A less conservative result can be found by Wilcoxon signed ranks test for comparisons of two classifiers. The Wilcoxon signed ranks test indicates sufficient performance difference between RBF-R algorithm and other five algorithms: RBF, MLP, KNN, PNN and RBF-N, with confidence level of at least 0.05.

Table 4
The properties of datasets used in Table 5.

Dataset	Data size	Features	Classes
Hepatitis	80	19	2
Zoo	101	16	7
Wine	178	13	3
Sonar	208	60	2
Glass	214	9	7
Heart	297	13	2
Ecoli	336	7	8
Liver	345	6	2
Ionosphere	351	34	2
Monks-3	432	6	2
Dermatol.	366	34	5
Breast	683	9	2
Pima	768	8	2
Tictactoe	958	9	2
German	1000	24	2

6. Conclusion

This paper presents two fast and accurate RBF learning rules for classification problems. In addition, a slight modification of Gaussian activation function is made which guarantees the linear separability property of RBF. For the proposed winner-take-all output layer, there is no adjustable parameter in output layer and no learning method is needed as well. In learning rules for hidden neurons, center and spread of each added neuron is determined based on the maximum achievable coverage (in terms of covered spatial domain or the number of covered training data). Each hidden unit is associated with one of the output classes. Based on these heuristics, three RBF networks were proposed and tested on well-known datasets. The comparison of results with some other leading RBF learning rules for classification problems and SVM methods indicates that proposed RBF networks perform satisfactory and are compatible. Analytic results show that the learning rules increase the linear separability of train

Table 5
Averaged 5-fold-validation test accuracies (%).

Dataset	RBF-R	RBF-N	RBF-WTA	DKP	MLP	KNN	PNN	RBF	Average
Hepatitis	81.9	81.1	***82.1***	72.0	61.0	56.0	62.0	65.0	71.2
Zoo	95.2	94.3	96.2	89.6	***99.4***	93.5	92.0	83.8	93.8
Wine	***95.5***	94.4	93.5	93.3	82.7	67.3	68.0	94.0	86.0
Sonar	83.8	83.2	83.9	***84.4***	76.9	82.3	83.1	77.9	82.0
Glass	66.1	66.3	69.1	***70.4***	52.8	62.4	70.2	38.7	63.1
Heart	***81.9***	80.5	80.6	79.9	64.1	61.9	59.9	73.5	73.0
Ecoli	78.5	79.3	81.0	92.9	88.9	92.7	***94.4***	69.5	85.6
Liver	62.2	62.8	61.0	65.5	64.2	***66.6***	65.3	53.8	63.7
Ionosphere	***95.5***	95.2	94.3	90.3	87.8	85.8	85.8	81.5	90.1
Monks-3	***99.0***	95.8	68.6	89.6	94.9	97.1	96.8	97.6	91.9
Dermatol.	92.8	92.1	92.7	90.3	***95.9***	94.7	94.9	70.2	91.2
Breast	96.3	96.4	***97.0***	96.1	94.5	95.7	95.9	94.1	95.5
Pima	***75.3***	72.1	73.8	74.7	67.5	73.2	70.5	71.0	72.1
Tictactoe	98.8	***98.8***	96.4	79.2	75.0	80.0	81.7	98.0	86.7
German	69.1	69.9	66.3	***74.1***	70.9	69.4	70.4	71.3	70.8
# Wins	5	1	2	3	2	1	1	0	
Ave.	***84.8***	84.1	82.4	82.8	78.4	78.6	79.4	76.0	81.1
Ave. Rank	***3.13***	3.80	3.87	3.87	5.40	5.27	4.73	5.93	
Total Rank	1	2	3.5	3.5	7	6	5	8	

data in hidden layer space and therefore the train accuracy of 100% is achievable, while the number of neurons is limited. The computation time for learning rules was proved to be of order two with respect to data size. The proposed learning algorithms are not stochastic and do not depend on the order of presentation of data. There are just two parameters for each learning rule, namely λ_l and λ_u, and the simulation results show that the exact values of those parameters have no significant effect on the accuracy of network as long as $\lambda_l < \lambda_u$, relaxing the design effort.

References

Bartlett, P. L. (1998). The sample complexity of pattern classification with neural networks: The size of the weights is more important than the size of the network. *IEEE Transactions on Information Theory*, *44*(2), 525–536.

Blanzieri, E. (2003). *Theoretical interpretations and applications of radial basis function networks, Tech. Rep. DIT-03-023*. Trento, Italy: Inf. Telecomun., Univ. Trento.

Bormhead, D. S., & Lowe, D. (1988). Multivariable functional interpolation and adaptive networks. *Complex Systems*, *2*(3), 321–355.

Cai, L., Rad, A. B., & Chan, W. L. (2010). An intelligent longitudinal controller for application in semiautonomous vehicles. *IEEE Transactions on Industrial Electronics*, *57*(4), 1487–1497.

Chang, C. C., & Lin, C. (2011). LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, *2*(3), 1–27. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Chen, S., Cowan, C. F. N., & Grant, P. M. (1991). Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, *2*(2), 302–309.

Constantinopoulos, C., & Likas, A. (2006). An incremental training method for the probabilistic RBF network. *IEEE Transactions on Neural Networks*, *17*(4), 966–974.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*, 1–30.

Fernández-Delgado, M., Cernadas, E., Barro, S., Ribeiro, J., & Neves, J. (2014). Direct Kernel Perceptron (DKP): Ultra-fast kernel ELM-based classification with non-iterative closed-form weight calculation. *Neural Networks*, *50*, 60–71.

Ferrari, S., Bellocchio, F., Piuri, V., & Borghese, N. A. (2010). A hierarchical RBF online learning algorithm for real-time 3-D scanner. *IEEE Transactions on Neural Networks*, *21*(2), 275–285.

Frank, A., & Asuncion, A. (2010). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science. URL: http://archive.ics.uci.edu/ml.

Girosi, F., & Anzellotti, G. (1993). Rates of convergence for radial basis functions and neural networks. In R. J. Mammone (Ed.), *Artif. neural netw. speech vis.* (pp. 97–113). London, UK: Chapman & Hall.

Hartman, E., Keeler, J. D., & Kowalski, J. M. (1990). Layered neural networks with Gaussian hidden units as universal approximations. *Neural Computation*, *2*(2), 210–215.

Hien, D. T. T., Huan, H. X., & Huynh, H. T. (2009). Multivariate interpolation using radial basis function networks. *International Journal of Data Mining Modelling and Management*, *1*(3), 291–309.

Huan, H. X., Hien, D. T. T., & Huynh, H. T. (2007). A novel efficient two-phase algorithm for training interpolation radial basis function networks. *Signal Processing*, *87*(11), 2708–2717.

Huan, H. X., Hien, D. T. T., & Tue, H. H. (2011). Efficient algorithm for training interpolation RBF networks with equally spaced nodes. *IEEE Transactions on Neural Networks*, *22*(6), 982–998.

Huang, G. B., Saratchandran, P., & Sundararajan, N. (2004). An efficient sequential learning algorithm for growing and pruning RBF (GAP-RBF) networks. *IEEE Transactions on Systems, Man, and Cybernetics*, *34*(6), 2284–2292.

Huang, S., & Tan, K. K. (2009). Fault detection and diagnosis based on modeling and estimation methods. *IEEE Transactions on Neural Networks*, *20*(5), 872–881.

Jaiyen, S., Lursinsap, C., & Phimoltares, S. (2010). A very fast neural learning for classification using only new incoming datum. *IEEE Transactions on Neural Networks*, *21*(3), 381–392.

Javan, D. S., Rajabi Mashhadi, H., Ashkezari Toussi, S., & Rouhani, M. (2012). On-line voltage and power flow contingencies rankings using enhanced radial basis function neural network and kernel principal component analysis. *Electric Power Components and Systems*, *40*, 534–555.

Javan, D. S., Rajabi Mashhadi, H., & Rouhani, M. (2013). A fast static security assessment method based on radial basis function neural networks using enhanced clustering. *Electrical Power and Energy Systems*, *44*(1), 988–996.

Kainen, P. C., Kurková, V., & Sanguineti, M. (2009). Complexity of Gaussian-radial-basis networks approximating smooth functions. *Journal of Complexity*, *25*, 63–74.

Kainen, P. C., Kurková, V., & Sanguineti, M. (2012). Dependence of computational models on input dimension: Tractability of approximation and optimization tasks. *IEEE Transactions on Information Theory*, *58*(2), 1203–1214.

Lee, Y. J., & Yoon, J. (2010). Nonlinear image up sampling method based on radial basis function interpolation. *IEEE Transactions on Image Processing*, *19*(10), 2682–2692.

Liu, Z., & Bozdogan, H. (2003). RBF neural networks for classification using new kernel functions. *Neural Parallel & Scientific Computations*, 41–52.

Mao, K. Z., & Huang, G. (2005). Neuron selection for RBF neural network classifier based on data structure preserving criterion. *IEEE Transactions on Neural*, *16*(6), 1531–1540.

Meng, K., Dong, Z. Y., Wang, D. H., & Wong, K. P. (2010). A self-adaptive RBF neural network classifier for transformer fault analysis. *IEEE Transactions on Power Systems*, *25*(3), 1350–1360.

Mhaskar, H. N. (2004a). On the tractability of multivariate integration and approximation by neural networks. *Journal of Complexity*, *20*, 561–590.

Mhaskar, H. N. (2004b). When is approximation by Gaussian networks necessarily a linear process? *Neural Networks*, *17*, 989–1001.

Moody, J., & Darken, C. J. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation*, *1*(2), 281–294.

Neural Network Toolbox User's Guide (2012). The Math Works, Inc., Natick, MA, USA. URL: http://www.mathworks.com/help/nnet/index.html.

Orr, M. J. L. (1995). Regularization in the selection of radial basis function centers. *Neural Computation*, *7*(3), 606–623.

Oyang, Y., Hwang, S., Ou, Y., Chen, C., & Chen, Z. (2005). Data classification with radial basis function networks based on a Novel Kernel Density Estimation Algorithm. *IEEE Transactions on Neural Networks*, *16*(1), 225–236.

Park, J., & Sandberg, I. W. (1991). Universal approximation using radial-basis function networks. *Neural Computation*, *3*(2), 246–257.

Platt, J. (1998). *Sequential minimal optimization: a fast algorithm for training support vector machines, Technical Report MSR-TR-98-14*. Microsoft research.

Powell, M.J.D. (1985). Radial basis functions for multivariable interpolation: A review. In *Proc. IMA conf. algorithms applicat. funct. data* (pp. 143–167).

Titsias, M. K., & Likas, A. C. (2001). Shared kernel models for class conditional density estimation. *IEEE Transactions on Neural Networks*, *12*(5), 987–997.

Tsai, C. C., Huang, H. C., & Lin, S. C. (2010). Adaptive neural network control of a self-balancing two-wheeled scooter. *IEEE Transactions on Industrial Electronics*, *57*(4), 1420–1428.

Wu, S., & Chow, T. W. S. (2004). Induction machine fault detection using SOM-based RBF neural networks. *IEEE Transactions on Industrial Electronics*, *51*(1), 183–194.

Xie, T., Yu, H., Hewlett, J., Różycki, P., & Wilamowski, B. (2012). Fast and efficient second-order method for training radial basis function networks. *IEEE Transactions on Neural Networks and Learning Systems*, *23*(4), 609–619.

Yao, W., Chen, X., Zhao, Y., & Tooren, M. (2012). Concurrent subspace width optimization method for RBF neural network modeling. *IEEE Transactions on Neural Networks and Learning Systems*, *23*(2), 247–259.

Yu, H., Xie, T. T., Paszczynski, S., & Wilamowski, B. M. (2011). Advantages of radial basis function networks for dynamic system design. *IEEE Transactions on Industrial Electronics*, *58*(12), 5438–5450.