ELSEVIER

Neurocomputing 42 (2002) 147–170

NEUROCOMPUTING

www.elsevier.com/locate/neucom

Searching for a solution to the automatic RBF network design problem

V. David Sánchez A.

Advanced Computational Intelligent Systems, Pasadena, CA 91116-6130, USA

Abstract

While amazing applications have been demonstrated in different science and engineering fields using neural networks and evolutionary approaches, one of the key elements of their further acceptance and proliferation is the study and provision of procedures for the automatic design of neural architectures and associated learning methods, i.e., in general, the study of the systematic and automatic design of artificial brains. In this contribution, connections between conventional techniques of pattern recognition, evolutionary approaches, and newer results from computational and statistical learning theory are brought together in the context of the automatic design of RBF regression networks. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Clustering; Computational learning theory; Evolutionary computation; Genetic algorithms; Neural networks; Pattern recognition; RBF networks; Regression; Statistical learning theory

1. Introduction

A significant number of neural network architectures have been studied in the literature, for a partial list of key references see e.g. [23–25,64]. An integral part of the subject of study here, RBF networks were introduced in [5,36,44]. Extensions were presented among others in [29,43,7,15,42,38,47–49,56,27]. The relationship between neural networks and pattern recognition is becoming increasingly closer as discussed recently in [50] and the references therein, further in-depth treatment of related issues can be found in [41,51,37,20,45,34,4,14]. Evolutionary computation is a fascinating research discipline incorporating areas of study which have become known as evolutionary programming and strategies as well as genetic algorithms

E-mail address: vdavidsanchez@earthlink.net (V. David Sánchez A.).

0925-2312/02/$ - see front matter © 2002 Elsevier Science B.V. All rights reserved.
PII: S0925-2312(01)00600-2

[1]. Characteristic of these stochastic search procedures is its population-based approach. To perform computational tasks, the individuals of the population compete and communicate, as in natural evolutionary systems. The original foundations are reviewed in [26,18,30,13,54] and in their references.

Neural, fuzzy, and evolutionary methods have been very actively researched and developed for key pattern recognition tasks including classification, clustering, and regression, see e.g. [39,2,40,33]. On the other hand, conventional pattern recognition techniques have been used in neural networks, notably clustering algorithms like k-means clustering, specifically for the design of radial basis function (RBF) network architectures. Advances in the application of evolutionary approaches to design neural networks have been reported, key references specially with regards to RBF network design are included e.g. in [3,60,62,12,10].

This paper is subdivided as follows. Section 2 concisely reviews RBF regression networks. Section 3 formulates the problem of their automatic design from a computational learning theoretical perspective. Section 4 discusses evolving solutions to the automatic RBF network design problem based on conventional pattern recognition techniques, constructive procedures, statistical learning theoretical means, and evolutionary approaches. Section 5 discusses a concrete example for the automatic design of an RBF regression network using a constructive procedure to help the reader visualize the ingredients of the design process. Section 6 discusses some results from the perspective of searching for a solution to the automatic design of RBF regression networks. Finally, Section 7 provides conclusions.

2. RBF regression networks

RBF networks that solve nonlinear regression tasks are called RBF regression networks. To get an idea of the regression task itself, the linear multivariate regression case is described in the sequel. The purpose of regression analysis can be summarized as to determine whether and which kind of a quantitative relationship (expressed by a regression equation) exists between two vector variables: the criterion variable $\vec{x} \in \mathbf{R}^n$ and the predictor variable $\vec{y} \in \mathbf{R}^m$, as well as to assess the prediction accuracy of the regression equation. Under zero-mean Gaussian noise $\vec{n} \in \mathbf{R}^m \sim \mathcal{N}(0, \mathbf{V})$, $\mathbf{V}$ is its covariance matrix, the data model is defined by the following regression equation: $\vec{y} = \vec{b} + \mathbf{A} \cdot \vec{x} + \vec{n}$, where $\vec{b} \in \mathbf{R}^m$ and $\mathbf{A} \in \mathbf{R}^{m \times n}$ represents a linear transformation.

In the case of a scalar predictor variable $y \in \mathbf{R}$, the matrix $\mathbf{A}$ becomes a row vector $\vec{w}$ and the decision function from the data model without considering the noise component becomes $f(\vec{x}) = b + \vec{w} \cdot \vec{x}$ where $b \in \mathbf{R}$ and $\cdot$ represents the scalar product. To generate a solution for a concrete regression problem, data are provided in one or more sets $\{(\vec{x}_i, y_i),\ \vec{x}_i \in \mathbf{R}^n,\ y_i \in \mathbf{R},\ i = 1, \ldots, N\}$. In the case of a nonlinear transformation $f : \mathbf{R}^n \to \mathbf{R}$ the data model becomes much more complex and neural networks offer an efficient solution for that purpose, in particular RBF regression networks. RBF networks possess three layers: one input, one hidden, and one output layer. The hidden layer contains neurons realizing basis functions. The

Table 1
Examples of RBF nonlinearities

Function name	Function expression $\phi(r,c) =$ (c =constant)
Linear	r
Cubic	r^3
Thin plate spline	$r^2 \log r$
Gaussian	$\exp(-r^2/c^2)$
Multiquadric	$(r^2 + c^2)^{\pm 1/2}$

ouput layer contains one neuron for the approximation of functions $f : \mathbf{R}^n \to \mathbf{R}$. The approximation function g_m realized by a network architecture with m hidden units has the form in (1). The weights w_i, $i = 1,\dots,m$ are parameters weighting the connections between each of the hidden neurons and the output neuron.

The basis functions $\phi_i : \mathbf{R}^n \to \mathbf{R}$ realized in the individual hidden units are parameterized scalar functions, which are built using a given nonlinearity $\phi : \mathbf{R} \to \mathbf{R}$. To parameterize this function, the centers $\vec{c}_i$ and the widths σ_i for $i = 1,\dots,m$ are used, see (2). Examples of nonlinearities for RBF networks are summarized in Table 1, one of the most commonly used is the Gaussian nonlinearity. A more general form of the basis functions is given in (3). The basis functions ϕ_i, $i = 1,\dots,m$ are parameterized by the center $\vec{c}_i$ and the width matrix Σ_i, which in general has the form stated in (4). R_i is a rotation matrix that represents the orientation of a hyperellipsoid with respect to the given coordinate system in $\mathbf{R}^n$. The hyperellipsoid is the corresponding support region for a given basis function. Outside of the hyperellipsoid, the value of the function ϕ tends to disappear fast with the distance to this region. D_i is a diagonal matrix, whose elements are the square of the semiaxes σ_{ij}^2, $j = 1,\dots,n$ of the hyperellipsoid. A common choice is the use of hyperspherical support regions, i.e., $\forall j = 1,\dots,n$: $\sigma_{ij} = \sigma_i$, in which case it is sufficient to determine one scalar width σ_i per width matrix Σ_i, essentially simplifying the expression for the basis functions ϕ_i in (3) to the one in (2).

$$g_m(\vec{x}) = \sum_{i=1}^{m} w_i \cdot \phi_i(\vec{c}_i, \sigma_i, \vec{x}), \tag{1}$$

$$\phi_i(\vec{c}_i, \sigma_i, \vec{x}) = \phi(\|\vec{x} - \vec{c}_i\|, \sigma_i), \quad 1 \leqslant i \leqslant m, \tag{2}$$

$$\phi_i(\vec{x}) = \phi[(\vec{x} - \vec{c}_i)^{\mathrm{T}} \cdot \Sigma_i^{-1} \cdot (\vec{x} - \vec{c}_i)], \quad \vec{x} \in \mathbf{R}^n, \tag{3}$$

$$\Sigma_i = R_i^{\mathrm{T}} \cdot D_i \cdot R_i, \quad D_i = \mathrm{diag}[\sigma_{i1}^2, \dots, \sigma_{in}^2]. \tag{4}$$

3. Formulating the problem of the automatic design of RBF networks

From a computational learning theoretical perspective, i.e., following the probably approximately correct (PAC) learning from examples model originally stated in [57] for Boolean functions and taking into consideration decision theoretic generalizations to characterize learning from examples of real-valued functions, see e.g. [22], the basic learning problem is formalized in terms of six elements: the instance space X, the outcome space Y, the decision space A, the decision rule space $\mathscr{H}=\{h:X\to A\}$, a family $\mathscr{P}$ of joint probability distributions P on the sample space $Z=X\times Y$, and a family $\mathscr{L}$ of regret functions L.

The regret function $L:\mathscr{P}\times\mathscr{H}\to R^+$ is defined in terms of a chosen loss function $l:Y\times A\to R$ to measure how much we failed to generate the optimal decision rule. In typical regression problems a function $f:X\to Y$ needs to be approximated by $h\in\mathscr{H}$ on the basis of given data generated according to an unknown distribution P from $X\times Y$ and l is the quadratic loss function defined by $l(y,h(x))=(h(x)-y)^2$. An example of how to define the regret function L in terms of a chosen loss function l is shown in (5), where $0<\alpha<1$ is called the accuracy parameter and d_v $(v>0)$ is a metric to measure distance from optimality according to (6). $\mathbf{r}_{h,l}$ defines the expected loss of a given decision rule h according to (7) and $\mathbf{r}_l^*(P)$ denotes the infimum of $\mathbf{r}_{h,l}(P)$ over all $h\in\mathscr{H}$.

$$L(P,h)=L_{\alpha,v}(P,h)=\begin{cases}1 & \text{if } d_v(\mathbf{r}_{h,l}(P),\ \mathbf{r}_l^*(P))>\alpha,\\ 0 & \text{otherwise,}\end{cases} \tag{5}$$

$$d_v(r,s)=\frac{|r-s|}{v+r+s},\quad r,s\in R,\quad r,s\geqslant 0, \tag{6}$$

$$\mathbf{r}_{h,l}(P)=\mathbf{E}(l(y,h(x)))=\int_Z l(y,h(x))\,\mathrm{d}P(x,y). \tag{7}$$

The definition of a learning method follows. Given $Z=X\times Y$ and a sample $\vec{z}=((x_1,y_1),\ldots,(x_m,y_m))$ randomly obtained from Z^m according to an unknown distribution P^m, $P\in\mathscr{P}$, a learning method is defined in (8). The goal of learning is formally defined as minimizing the so-called big "L" risk in (9) of $\mathscr{A}$ under P for the sample size m. We say that a learning method $\mathscr{A}$ solves the basic learning problem if (10) holds, $0<\delta<1$ is called the confidence parameter. For the regret function example $L_{\alpha,v}$ given before, $R_{L,\mathscr{A},m}(P)\leqslant\delta$ translates into demanding according to the typical PAC approach that when m random training examples are generated according to P and given to the learning method $\mathscr{A}$, the error of the decision rule generated by the learning method $\mathscr{A}$ and measured with the regret function $L_{\alpha,v}$ has a distance greater than α from optimal using the metric d_v with probability at most δ. The smallest value of such m is called the sample complexity of the learning method $\mathscr{A}$ and is denoted $m(L,\delta)$.

$$\mathscr{A}:\ \bigcup_{m\geqslant 1} Z^m\to\mathscr{H}, \tag{8}$$

$$R_{L,\mathscr{A},m}(P)=\int_{\vec{z}\in Z^m} L(P,\mathscr{A}(\vec{z}))\,\mathrm{d}P^m(\vec{z}), \tag{9}$$

$\forall L\in\mathscr{L}$, $\forall\delta$, $\exists m=m(L,\delta)$ such that:

$$\forall P \in \mathscr{P}\colon R_{L,\mathscr{A},m}(P) \leqslant \delta. \tag{10}$$

The definition of the associated optimization problem follows, which is relevant for practical environments. For that purpose, the empirical risk is first defined. For a given decision rule h and a given random training sample $\vec{z} = (z_1, \ldots, z_m)$, $z_j = (x_j, y_j) \in Z$ drawn according to an unknown distribution P the empirical risk is defined in (11) and $\hat{\mathbf{r}}^*(\vec{z}) = \inf\{\hat{\mathbf{r}}_h(\vec{z})\colon h \in \mathscr{H}\}$. The associated optimization problem with the basic learning problem can then be stated: given the training sample $\vec{z}$, find a decision rule $\hat{h} \in \mathscr{H}$ such that $\hat{\mathbf{r}}_{\hat{h}}(\vec{z})$ is close to $\hat{\mathbf{r}}^*(\vec{z})$. In [22] a result shows that a learning method $\mathscr{A}$ which is a solution to the associated optimization problem, provides a solution to the basic learning problem, provided that the empirical risk estimates are uniformly convergent. In practice, the common situation is to solve the associated optimization problem, then only finite-sized training sets are available.

$$\hat{\mathbf{r}}_h(\vec{z}) = \frac{1}{m} \sum_{j=1}^{m} l(y_j, h(x_j)). \tag{11}$$

This general learning theoretical discussion summarized is the basis for the understanding of the learning-from-examples problem as a whole, in particular for the understanding of the complexity of learning, and applies to learning using RBF regression networks. Two kinds of complexity are considered when a learning-from-examples problem is being investigated from a learning-theoretical perspective: the sample complexity and the computational complexity. The sample complexity refers to the number of examples needed to generate a good approximation. The computational complexity refers to the computational effort required to generate the approximation from the examples given. A bound for the sample complexity of fixed, finite-sized RBF networks is given in (12), α and δ are parameters used to define the accuracy and the confidence in the learning model, respectively. v is used to parameterize a metric d_v to measure the distance from optimality. W and m are the number of parameters (weights) and the number of hidden neurons (basis functions), respectively. This bound translates into the following statement. The training set size and the number of parameters should be in a nearly linear relationship for a good approximation, at least in the worst case.

$$O\left(\frac{1}{\alpha^2 v}\left(W \log \frac{m}{\alpha v} + \log \frac{1}{\delta}\right)\right). \tag{12}$$

The relationship in (13) for the generalization error measured using the metric $L^2(P)$ has been introduced in [38]. $P(\vec{x}, y)$ is an unknown probability distribution defined on $\vec{X} \times Y$. The examples provided in the training set are sampled from $\vec{X} \times Y$ according to $P(\vec{x}, y)$. With probability greater than $1 - \delta$, (13) holds for a Gaussian RBF network with m hidden units, input dimension n, and a training set of size N. f_0 represents the regression function to be approximated by the function $\hat{f}_{m,N}$ realized by a finite RBF network with m hidden units trained using a given finite training set $\{(\vec{x}_j, y_j), j = 1, \ldots, N\}$. $\hat{f}_{m,N}$ represents a solution to the associated non-convex optimization problem given in (14). The bound for the generalization error can be decomposed into an approximation error bound $\epsilon(m)$ [16] and an estimation error bound $2w(N,m,\delta)$ [17] that correspond to the two terms on the right side of expression (13). This relationship shows how the number of parameters (hypothesis complexity), the number of examples (sample complexity), and the generalization error are related. In particular, due to this relationship it can be concluded that m needs to grow more slowly than N for guaranteed convergence, otherwise the estimation error $w(N,m,\delta)$ will diverge. For a fixed number of training data N there is an optimal number of centers $m=m^*(N)$ that minimizes the generalization error. For a low value of m the approximation and estimation error terms are high and low, respectively. For a high value of m the approximation and estimation error terms are low and high, respectively. $m^*(N)$ lies in between.

$$\|f_0-\hat{f}_{m,N}\|^2_{L^2(P)} \leqslant O\left(\frac{1}{m}\right)+O\left(\left[\frac{nm\ln(mN)-\ln\delta}{N}\right]^{1/2}\right), \tag{13}$$

$$\hat{f}_{m,N}=\arg\min_{w_i,\vec{z}_i,\sigma_i}\sum_{j=1}^{N}\left[y_j-\sum_{i=1}^{m}w_i\cdot\phi_i(\vec{z}_i,\sigma_i,\vec{x}_j)\right]^2. \tag{14}$$

In the specific area of RBF regression networks as specified in (1) and (2) and for a chosen nonlinearity ϕ the solution procedure to the automatic RBF network design problem is one that given a training sample $\vec{z}$ automatically determines the number of basis functions or hidden neurons m, the weights w_i, the centers $\vec{c}_i$ and the widths σ_i, $i=1,\ldots,m$ while optimizing the generalization capability of the network to be designed.

4. Evolving solutions to the automatic RBF network design problem

The problem of systematically and automatically designing artificial brains is far from being completed. It is an open and exciting research area and to great extent a new one. In this section, in the context of RBF regression network design, four approaches are summarized. One based partially on conventional pattern recognition techniques, i.e. k-means clustering. Then a constructive design procedure is presented. In addition, statistical learning theoretical and evolutionary approaches are discussed.

4.1. Clustering and basic weight determination

Given a set of points $\{\vec{x}_i\in\mathbf{R}^p,\ i=1,\ldots,n\}$ the goal of clustering is to determine a set of clusters C_j each of them represented by its prototype $\{\vec{c}_j\in\mathbf{R}^p,\ j=1,\ldots,m\}$

Table 2
Distance measures within clusters

Expression	Description
$d_j = \dfrac{\sum_{i=1}^{m_j} \sum_{k=1}^{m_j} \|\vec{x}_i - \vec{x}_k\|}{m_j \cdot (m_j - 1)}$	Average distance
$d_j = \dfrac{\sum_{i=1}^{m_j} \|\vec{x}_i - \vec{c}_j\|}{m_j}$	Centroid distance
$d_j = \dfrac{\sum_{i=1}^{m_j} \min_k \|\vec{x}_i - \vec{x}_k\|}{m_j}$	Nearest-neighbor distance

Table 3
Distance measures between clusters

Expression	Description
$d_{C_j,C_l} = \min_{\vec{x}_i \in C_j, \vec{x}_k \in C_l} \|\vec{x}_i - \vec{x}_k\|$	Single linkage distance
$d_{C_j,C_l} = \max_{\vec{x}_i \in C_j, \vec{x}_k \in C_l} \|\vec{x}_i - \vec{x}_k\|$	Complete linkage distance
$d_{C_j,C_l} = \dfrac{\sum_{i=1}^{m_j} \sum_{k=1}^{m_l} \|\vec{x}_i - \vec{x}_k\|}{m_j \cdot m_l}$	Average linkage distance
$d_{C_j,C_l} = \|\vec{c}_j - \vec{c}_l\|$	Centroid linkage distance

such that the distances between points of the same cluster are minimized and the distances between clusters are maximized. This corresponds to central clustering as opposed to pairwise clustering. In pairwise clustering pairwise distances are given $\{d_{i,j} \in \mathbf{R},\ i,j = 1,\ldots,n\}$. The goal of clustering stated needs to be further specified, e.g. in relation to the distance measures used. Tables 2 and 3 show commonly used distance measures within and between clusters. $\|\cdot\|$ is the Euclidian norm, m_j is the number of points belonging to cluster $j = 1,\ldots,m$, and the centroid $\vec{c}_j$ of a cluster is given in (15). The result of clustering is an element M of the set of assignment matrices $\mathscr{M}_{n,m}$ in (16), which is a Boolean representation of data partitionings. $M_{i,j} = 1$ and 0 indicate that the data point $\vec{x}_i$ belongs and does not belong to cluster C_j, respectively.

$$\vec{c}_j = \frac{\sum_{\vec{x}_i \in C_j} \vec{x}_i}{m_j}, \tag{15}$$

$$\mathscr{M}_{n,m} = \left\{ M \in \{0,1\}^{n \times m},\ \sum_{j=1}^{m} M_{i,j} = 1,\ i = 1,\ldots,n \right\}. \tag{16}$$

The criterion function used in k-means clustering, a very popular clustering technique, for a given data set $\{\vec{x}_t,\ t = 1,\ldots,n\}$ is defined in (17) where $\vec{c}_l$ are the centroid vectors of the clusters C_l, $l = 1,\ldots,m$. k-means clustering [35] has been used to determine the centers $\vec{z}_i$ of an RBF network from the data provided to solve the regression task, e.g. in [36].

$$E = \sum_{l=1}^{m} \sum_{\vec{x}_t \in C_l} \|\vec{x}_t - \vec{c}_l\|^2. \tag{17}$$

Once the centers and the widths are fixed a basic learning method for weight determination can be applied. This basic method is outlined in the sequel. For the RBF networks realizing g_m in (1), fixed values for $m, \vec{z}_i, \sigma_i, \; i = 1, \ldots, m$, and a given training data set $Tr = \{(\vec{x}_i, y_i), \; i = 1, \ldots, N\}$, the learning method needs to solve the optimization problem in (18) determining the weight vector $\vec{w} = (w_1, \ldots, w_m)^{\mathrm{T}}$. This leads to the solution of a linear equation system or more specifically, of a linear least-squares problem and to the maximization of the memorization capability for a fixed number of hidden units m. The actual learning goal, i.e., the optimization of the generalization capability is dependent on m (model complexity) and in practical settings is typically based on a given test data set $Te = \{(\vec{x}_k, y_k), \; k = 1, \ldots, M\}$. The solution of the least-squares problem is found using the non-iterative single value decomposition (SVD) in the basic weight determination procedure or other iterative numerical optimization techniques.

$$\min_{\vec{w}} e_{Tr}, \quad e_{Tr} = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - \sum_{j=1}^{m} w_j \cdot \phi_j(\vec{x}_i) \right)^2. \tag{18}$$

4.2. Constructive RBF regression network design procedure

The basic weight-determination procedure of the previous subsection and an algorithm for automatic width determination are used throughout the constructive RBF regression network-design procedure. The width-determination algorithm is described first. The algorithm works on a given set of centers $C = \{\vec{c}_1, \ldots, \vec{c}_m\}$ and determines automatically the corresponding width set $S = \{\sigma_1, \ldots, \sigma_m\}$. The individual widths are determined using (19), $\|\cdot\|$ is the Euclidian norm. The pair $(\vec{c}_i, \sigma_i)$ uniquely specifies the basis function $\phi_i(\vec{x})$.

$$\sigma_i = \min_{k=1,\ldots,m, \; k \neq i} \{\|\vec{c}_k - \vec{c}_i\|\}. \tag{19}$$

In Fig. 1, two examples show how the algorithm works. Fig. 1(a) shows a one-dimensional example. Five centers are given: $c_1, \ldots, c_5 \in \mathbf{R}$. To make things clear circles were drawn using the widths determined by the algorithm. Each circle has its center at one given RBF network center c_i and its radius equals the corresponding width σ_i determined by the algorithm. Fig. 1(b) shows a two-dimensional example. Seven centers were given: $\vec{c}_1, \ldots, \vec{c}_7 \in \mathbf{R}^2$. Their corresponding radii $\sigma_1, \ldots, \sigma_7 \in \mathbf{R}$ were determined by the algorithm. In this case, the drawn circles correspond geometrically to the hyperellipsoids (dimension $n = 2$) or support regions for the basis functions ϕ_i. In the one-dimensional case, the support regions are the intervals $[c_i - \sigma_i, \; c_i + \sigma_i]$. The circles were only drawn to visualize the width sizes. As the examples show, an overlapping of the support regions is possible. The centers may

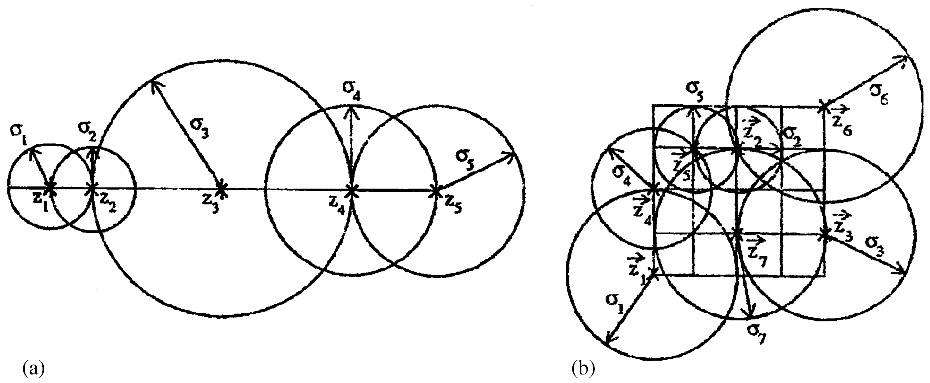

Fig. 1. Examples for width determination: (a) One-dimensional example; (b) two-dimensional example.

be located anywhere in $\mathbf{R}^n$. The grid of the two-dimensional example was drawn to better visualize the location of the given centers.

Using the algorithm for automatic width determination in the proposed constructive RBF regression network design procedure, an overlapping coverage of the space is generated by means of the given centers and the widths determined by the algorithm. This happens only for the areas of the space where training data are provided. The network approximation tends to zero with increasing distance from the generated overlapping coverage. If a better approximation is desired also for these distant areas, then more training data from these areas need to be provided, which is a reasonable condition.

The inputs to the constructive RBF regression network design procedure are two sets of data: a training data set $Tr = \{(\vec{x}_i, y_i),\ i = 1,\dots,N\}$, $\vec{x}_i \in \mathbf{R}^n, y_i \in \mathbf{R}$ and a test data set $Te = \{(\vec{x}_k, y_k),\ k = 1,\dots,M\}$, $\vec{x}_k \in \mathbf{R}^n, y_k \in \mathbf{R}$. Optionally, the training data can be reordered randomly at the beginning. The initial architecture consists of one hidden neuron, i.e. the initial size is $m := 1$ and the initial center is $\vec{c}_1 := \vec{x}_1$, $(\vec{x}_1, y_1) \in Tr$. The first width is determined using $\sigma_1 := \min_{i=2,\dots,N} \{\|\vec{x}_i - \vec{c}_1\|\}$, $(\vec{x}_i, y_i) \in Tr$. Using the basic weight-determination procedure and the training data set, the weight vector $\vec{w}$ is determined together with the approximation error for the training and test data set, i.e., the memorization and the generalization capability of the network, respectively. In particular, the minimum of the generalization error $e_{Te}^{\min}$ is initialized. The index set, I, points to training data whose component $\vec{x}$ can still be considered for inclusion as center of the hidden neuron to be incorporated into the RBF regression network architecture. Initially, $idx[i] := i + 1 \in I,\ i = 1,\dots,n_I := N - 1$.

The iterative, incremental growth of the architecture searches for an additional hidden neuron, which leads to an increase of the network's generalization capability. In each iteration, either a new hidden neuron is generated, i.e., the network generalization capability is increased, or the whole procedure is terminated, because no further generalization improvement could be achieved and therefore no new hidden neuron was added to the network architecture. The iteration search for additional hidden neurons, which takes place $\forall i$: $1 \leqslant i \leqslant n_I$, $idx[i] \in I$, $(\vec{x}_{idx[i]}, y_{idx[i]}) \in Tr$ is described in more detail in the sequel. The following initialization takes place. The minimum index $i^*_{m+1} := 1$ and the generalization error for $m := m+1 \rightarrow e^{m+1}_{Te} := e^{\min}_{Te}$. Tentatively, $\vec{c}_{m+1} := \vec{x}_{idx[i]}$ and using the width-determination algorithm σ_{m+1} is determined together with e_{Te}. If $e_{Te} < e^{m+1}_{Te}$ then $e^{m+1}_{Te} := e_{Te} \wedge i^*_{m+1} := i$. This process finds the best tentative hidden neuron, i.e., the one among the ones indexed by the elements in I which leads to the minimum generalization error. When $e^{m+1}_{Te} < e^{\min}_{Te}$ the tentative hidden neuron characterized by $(\vec{c}_{m+1}, \sigma_{m+1})$ is incorporated into the network architecture and the following updates take place: $e^{\min}_{Te} := e^{m+1}_{Te}$ and $I := I \backslash \{idx[i^*_{m+1}]\}$, $n_I := n_I - 1$. Otherwise, i.e., if $e^{m+1}_{Te} \geqslant e^{\min}_{Te}$ the evolution of the network architecture stops and no further generalization gain can be achieved.

In a line of study, learning methods for RBF networks have been systematically designed and developed. The basic learning method for weight determination was presented in [46]. The constructive RBF regression network design procedure was introduced in [47]. Both of these methods handle noise-free and noisy data. A new robust learning method for RBF networks that handles outliers in the data as well was introduced in [48]. The evolving RBF network architecture within the constructive procedure belongs to the family of approximations given in (20). The basis functions $\phi_i : \mathbf{R}^n \rightarrow \mathbf{R}$ realized in the hidden neurons can be determined according to (2). If the generalization error is given by the approximation error e_{Te} for the test data set Te, then the sequence in (21) for the generated architectures during the strategy to generate the final RBF network holds, i.e., a monotonic increase of the generalization capability of the RBF regression network is achieved.

$$U_N = \left\{ g_m \mid g_m(\vec{x}) = \sum_{i=1}^{m} w_i \cdot \phi_i(\vec{z}_i, \sigma_i, \vec{x}),\ m = 1, \ldots, N \right\}, \tag{20}$$

$$e^{(1)}_{Te} > e^{(2)}_{Te} > \cdots > e^{(m)}_{Te} \leqslant e^{(m+1)}_{Te}. \tag{21}$$

In terms of the computational complexity of the constructive RBF regression network design procedure, one way to express this complexity is through the number ΣN of networks examined until the final network architecture completes its design and training. This number can be determined using (23). An upper bound for ΣN can be found in terms of N when $m = N$. This bound has only theoretical character because overfitting takes place for $m = N$. Typically, m is much smaller than N. At the same time we should point out that the training of network architectures of different size has different complexity. Groups of architectures can be built that have the same number of p hidden neurons. For all the group members, training is equally complex. Training is carried out using the basic weight-determination procedure based on N training data and p hidden neurons, and consists essentially of the computation of the singular value decomposition of the matrix $C \in \mathbf{R}^{N \times p}$, whose computational complexity $\kappa_{N,p}$ is expressed in (24).

$$\begin{aligned}\Sigma N &= 1+(N-1)+(N-2)+\cdots+(N-m+1)+(N-m)\\ &= 1+Nm-(1+2+\cdots+(m-1)+m) \qquad (22)\\ &= 1+Nm-\frac{m(m+1)}{2}, \qquad (23)\end{aligned}$$

$$\kappa_{N,p}=O(4Np^2+8p^3). \qquad (24)$$

4.3. Support vector regression machines

Recently, kernel methods and support vector machines (SVMs) have become very popular for classification and regression, in particular when using RBF networks. Details on support vector algorithms and learning can be found in [58,53,8]. We present an RBF network solution with an associated SVM learning method. Additional, more specific regression-related material can be found e.g. in [11,59,52]. Theoretical foundations were reported in [55]. For regression the decision function is given in (25).

$$f(\vec{x})=b+\sum_{i=1}^{m} y_i(\alpha_i-\beta_i)K(\vec{x}_i,\vec{x}). \qquad (25)$$

For an ε-insensitive loss function:

$$L(x)=\begin{cases}0, & |x|<\varepsilon,\\ |x|, & \varepsilon\leqslant |x|,\end{cases} \qquad (26)$$

a quadratic optimization problem needs to be solved: the dual objective function to be minimized is given in (27) subject to the conditions in (28) $\forall i=1,\dots,m$.

$$\begin{aligned}W(\alpha,\beta)=&\sum_{i=1}^{m} y_i(\alpha_i-\beta_i)-\varepsilon\sum_{i=1}^{m}(\alpha_i+\beta_i)\cdots\\ &-\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}(\alpha_i-\beta_i)(\alpha_j-\beta_j)K(\vec{x}_i,\vec{x}_j), \qquad (27)\end{aligned}$$

$$0\leqslant\alpha_i\leqslant C,\quad 0\leqslant\beta_i\leqslant C,$$

$$\sum_{i=1}^{m}\alpha_i=\sum_{i=1}^{m}\beta_i. \qquad (28)$$

The bias b is typically determined by averaging individual values which are gained from the Karush–Kuhn–Tucker conditions leading to $b=y_i-\vec{w}\cdot\vec{x}_i\pm\varepsilon$, see e.g. [6], where,

$$\vec{w} = \sum_{i=1}^{m} y_i(\alpha_i^* - \beta_i^*)\vec{x}_i. \tag{29}$$

α_i^* and β_i^* are the optimal values previously determined. Similar quadratic optimization problems are generated when instead of the ε-insensitive loss function quadratic or robust loss functions [28,21] are utilized. A robust learning method for RBF regression networks was introduced in [48]. When the same statistical learning framework is used, the solution provided with the RBF regression network to the nonlinear regression problem shows a close connection to the solution to the linear regression task which follows. The linear regressor's decision function is given in (30). The optimization problem consists in the maximization of the functional given in (31) subject to the constraints in (28) as before, and the solution $\vec{w}$ is given in (32), whereas b is determined as in the case of the RBF network solution.

$$f(\vec{x}) = b + \vec{w} \cdot \vec{x}, \tag{30}$$

$$\begin{aligned} W(\alpha, \beta) = {} & \sum_{i=1}^{m} y_i(\alpha_i - \beta_i) - \varepsilon \sum_{i=1}^{m} (\alpha_i + \beta_i) \cdots \\ & -\frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} (\alpha_i - \beta_i)(\alpha_j - \beta_j)\vec{x}_i \cdot \vec{x}_j, \end{aligned} \tag{31}$$

$$\vec{w} = \sum_{i=1}^{m} (\alpha_i^* - \beta_i^*)\vec{x}_i. \tag{32}$$

4.4. Evolutionary approaches

The integration of neural and evolutionary systems has been an active area of research including coupled evolutionary and neural systems, evolutionary algorithm preprocessing of neural network input, genetic design of neural networks, and applications, among others. The combination of evolutionary algorithms and neural networks can also be subdivided into two broad categories: supportive combinations and collaborative combinations. The supportive combinations include using neural networks to assist evolutionary algorithms, using evolutionary algorithms to assist neural networks, and using evolutionary algorithms and neural networks independently on the same task. Collaborative combinations include evolutionary algorithms for neural network learning, evolutionary algorithms to specify neural network topology, and others. In relation to this subdivision we are interested in evolutionary algorithms to specify neural network topology within the collaborative combinations. Most of the work reported is for the application to other types of architecture though, most notably multilayer perceptrons (MLP), e.g. [31], sigma–pi networks, e.g. [63], or networks based on cellular automata, e.g. [9]. Or for designing Boolean neural networks, e.g. [19], as opposed to neural networks with real-valued outputs relevant to regression. Or for RBF networks for classification, e.g. [32], as opposed to RBF networks for regression applications. A few approaches have been presented for the design of RBF regression networks as referenced in the introductory section.

In [60,61] an approach using genetic algorithms was proposed to solve the RBF network-design problem within the context of time-series prediction. The main idea is to evolve centers and widths using a cooperative–competitive genetic algorithm. The genetic competition among individual RBF's is the basis for the selection of the set of centers. Competition is restricted within orthogonal niches determined using the singular value decomposition SVD. The outer loop of the algorithm constructs and evaluates successive generations $G_0, G_1, \ldots$. A fitness-based selection procedure is used to generate the population G_{k+1} from the population G_k. In contrast to other approaches that evolve populations of competing networks, in this algorithm, the entire population encodes only one network. Each population individual specifies a different basis function of the RBF network. In each generation G_k the algorithm state consists of a population of bit strings ψ_i, $i=1,\ldots,m$. m is the size of the population which is equal to the number of hidden neurons in the RBF network. The bit string ψ_i encodes the center $\vec{c}_i$ and the width σ_i of a basis function ϕ_i, cf. (1) and (2). If $\vec{c}_i$ and σ_i are encoded with l bits of precision, then each basis function ϕ_i can be encoded using a bit string of size $L=(n+1)l$, n is the dimension of the input vector $\vec{x}$. $\psi_i = b_1 b_2 \ldots b_L$, $b_h \in \{0,1\}$.

In a fixed-sized population the algorithm draws each slot in G_{k+1} by sampling from G_k with replacement, and with each encoded ϕ_i in G_k having the probability of selection proportional to its performance. The performance measure called fitness specifies the expected number of copies of ψ_i to occur in G_{k+1}. These copies fill the slots in G_{k+1} and are then modified using the genetic operators of recombination, creep, and mutation. The cooperative–competitive genetic selection of the algorithm is based on the following rationale. Suppose that we have the training data $Tr=\{(\vec{x}_j, f(\vec{x}_j)),\ j=1,\ldots,N\}$. The basis function ϕ_i is normalized to $\bar{\phi}_i$ according to (33). The vector called the normalized activation sequence $\hat{\phi}_i$ is defined according to (34). The inner product $\hat{\phi}_i \cdot \hat{\phi}_{i'}$ helps to define the degree to which the basis functions should be competing or cooperating according to the extremes shown in (35). When $\hat{\phi}_i \cdot \hat{\phi}_{i'}$ is near zero, then ϕ_i and $\phi_{i'}$ are contributing almost independently to the overall RBF network performance. Their relationship should be rather cooperative than competitive. When $\hat{\phi}_i \cdot \hat{\phi}_{i'}$ is near one, then ϕ_i and $\phi_{i'}$ are attempting to make almost the same contribution. They should compete in the sense that selecting one basis function to be in the next generation should exclude the selection of the other. This is described as ϕ_i and $\phi_{i'}$ sharing a niche.

$$\bar{\phi}_i(\vec{x}) = \frac{\phi_i(\vec{x})}{\sum_{j=1}^{N} \phi_i^2(\vec{x}_j)}, \quad i=1,\ldots,m, \tag{33}$$

$$\hat{\phi}_i(\vec{x}) = [\bar{\phi}_i(\vec{x}_1), \bar{\phi}_i(\vec{x}_2), \ldots, \bar{\phi}_i(\vec{x}_N)], \quad \|\hat{\phi}_i\| = 1, \tag{34}$$

$$\hat{\phi}_i \cdot \hat{\phi}_{i'} = \begin{cases} 1 & \text{(pure competition, nice sharing)}, \\ 0 & \text{(pure cooperation, independent niches)}. \end{cases} \tag{35}$$

After examining several prospective fitness measures for the basis functions ϕ_i, the one stated in (36), where $E(\cdot)$ denotes mean value over all $\hat{\phi}_{i'}$, $i' = 1, \ldots, m$, was selected. That fitness measure was selected because it fulfilled the cooperative–competitive requirements in the rationale described above better than the others examined. In the experiments reported, a value of $\beta = 1.5$ was used for the fitness measure within the cooperative–competitive selection procedure. After applying the genetic operators to the selected strings, the strings in the generation G_{k+1} are decoded into the centers and widths of the RBF network and this is trained using the LMS algorithm to save computational time. This step involves the solution of a linear least-squares problem. To improve precision of the training solution, the SVD method is applied after the execution of the genetic algorithm.

$$\frac{|w_i|^\beta}{E(|w_{i'}|^\beta)}, \quad 1 < \beta < 2. \tag{36}$$

To get a feeling about other directions in which evolutionary approaches help to design RBF networks for regression another procedure is briefly reviewed. In [12] an evolutionary strategy based on a random search algorithm is used to determine only the widths σ_i, $i = 1, \ldots, m$ of the RBF network. Heuristic procedures determine the centers and weights of the network. The widths are determined using a two-step learning strategy composed of a local optimization of the width of each added neuron, called local tuning, and a global optimization of all the network widths, called fine tuning. Based on an initial value σ_i at time t a population of individuals is generated using (37) excluding zero as a possible result. $N(0, \Sigma)$ is a random number generated using a Gaussian distribution with mean zero and variance Σ. $\sigma_i(t)$ is replaced by $\sigma_i(t+1)$ only if the overall network approximation error decreases. Through the adjustment of Σ a non-local search is enabled to escape from local minima in the σ_i parameter space. Based on an initial value Σ at time t updated values are generated according to (38), ΔE is the change in the value of the network approximation error, $c_1 \to 1^-$, and $c_2 \to 1^+$.

$$\sigma_i(t+1) = \sigma_i(t)[1 + N(0, \Sigma)], \tag{37}$$

$$\Sigma(t+1) = \begin{cases} c_1 \Sigma(t) & \text{if } \Delta E > 0, \\ c_2 \Sigma(t) & \text{if } \Delta E < 0, \\ \Sigma(t) & \text{if } \Delta E = 0. \end{cases} \tag{38}$$

5. Visualization example

To visualize the key components of the automatic design of RBF regression networks, the approximation of a real-valued function of a two-dimensional real vector is described. The partitioning of the input space is as follows. $N_{Tr} = N_x N_y$

and $N_{Te} = (N_x - 1)(N_y - 1)$ elements of the training and test data sets are generated, respectively. N_x and N_y are the number of equidistant points of the partitions of the x- and y-coordinates of the definition square $[a_x, b_x] \times [a_y, b_y]$. The constant distances between partition points are given in (39). The components of the training partition are specified in (40) and (41). The components of the test partition are specified in (42) and (43).

$$d_x = \frac{b_x - a_x}{N_x - 1}, \quad d_y = \frac{b_y - a_y}{N_y - 1}, \tag{39}$$

$$x_i^{Tr} = a_x + i d_x, \quad i = 0, \ldots, N_x - 1, \tag{40}$$

$$y_j^{Tr} = a_y + j d_y, \quad j = 0, \ldots, N_y - 1, \tag{41}$$

$$x_i^{Te} = a_x + (i + 0.5) d_x, \quad i = 0, \ldots, N_x - 2, \tag{42}$$

$$y_j^{Te} = a_y + (j + 0.5) d_y, \quad j = 0, \ldots, N_y - 2. \tag{43}$$

Using these one-dimensional partitions, a two-dimensional partititon of the definition square $[a_x, b_x] \times [a_y, b_y]$ is generated. The x- and y-coordinates of the elements of the training data set $(x_k^{Tr}, y_k^{Tr}, z_k^{Tr})$, $k = 0, \ldots, N_{Tr} - 1$, $z_k^{Tr} = f(x_k^{Tr}, y_k^{Tr})$ are determined according to $x_k^{Tr} = x_i^{Tr}$, $y_k^{Tr} = y_j^{Tr}$ for $i = 0, \ldots, N_x - 1$, $j = 0, \ldots, N_y - 1$ and $k = N_x i + j$. In an analogous way the x- and y-coordinates of the elements of the test data set $(x_l^{Te}, y_l^{Te}, z_l^{Te})$, $l = 0, \ldots, N_{Te} - 1$, $z_l^{Te} = f(x_l^{Te}, y_l^{Te})$ are determined according to $x_l^{Te} = x_i^{Te}$, $y_l^{Te} = y_j^{Te}$ for $i = 0, \ldots, N_x - 2$, $j = 0, \ldots, N_y - 2$ and $l = (N_x - 1)i + j$. The function specified in (44) is used to generate the training and test data with $N_x = N_y = 11$, $d_x = d_y = 0.1$, $N_{Tr} = 121$, $N_{Te} = 100$.

$$f\colon [0.0, +1.0] \times [-0.5, +0.5], \quad f(x, y) = \cos(\pi x) \sin(\pi \sqrt{|y|^3}). \tag{44}$$

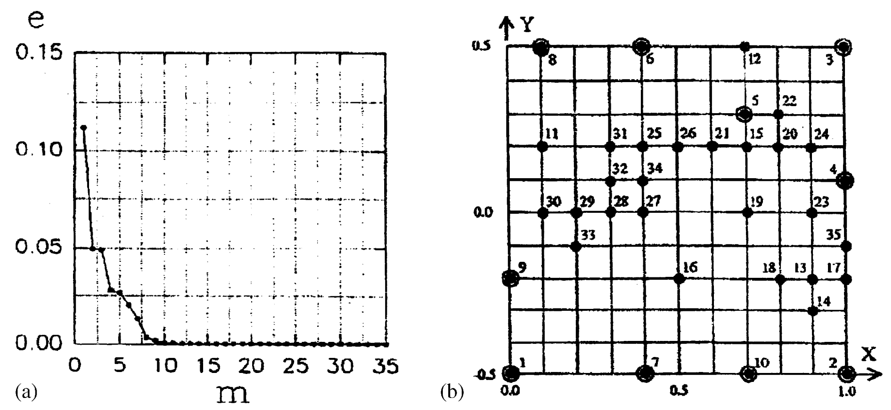

Fig. 2. Generalization and centers for the evaluation example: (a) learning curve; (b) centers.

Table 4
Centers and widths for the evaluation example

i	1	2	3	4	5	6	7	8	9	10	11	12
z_{ix}	0.00	1.00	1.00	1.00	0.70	0.40	0.40	0.10	0.00	0.70	0.10	0.70
z_{iy}	−0.50	−0.50	0.50	0.10	0.30	0.50	−0.50	0.50	−0.20	−0.50	0.20	0.50
σ_i	0.30	0.22	0.28	0.14	0.10	0.30	0.30	0.30	0.22	0.28	0.20	0.20
i	13	14	15	16	17	18	19	20	21	22	23	24
z_{ix}	0.90	0.90	0.70	0.50	1.00	0.80	0.70	0.80	0.60	0.80	0.90	0.90
z_{iy}	−0.20	−0.30	0.20	−0.20	−0.20	−0.20	0.00	0.20	0.20	0.30	0.00	0.20
σ_i	0.10	0.10	0.10	0.22	0.10	0.10	0.20	0.10	0.10	0.10	0.14	0.10
i	25	26	27	28	29	30	31	32	33	34	35	
z_{ix}	0.40	0.50	0.40	0.30	0.20	0.10	0.30	0.30	0.20	0.40	1.00	
z_{iy}	0.20	0.20	0.00	0.00	0.00	0.00	0.20	0.10	−0.10	0.10	−0.10	
σ_i	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	

The results for this evaluation example are discussed in the sequel. Fig. 2(a) shows the monotonic increase of the generalization capability or, equivalently, monotonic minimization of the generalization error of the RBF network being constructed dependent on the number m of hidden neurons. The optimal RBF network was found to have $m = 35$ hidden neurons. In Fig. 2(a) for $m > 1$ and for each (m, e) drawn, $(N - m)$ RBF networks, all with m hidden neurons, are designed and trained, but only the minimal value of e is considered. Fig. 2(b) shows the centers automatically generated for the optimal RBF network in the two-dimensional input space. The coordinates (c_{ix}, c_{iy}) of the centers $\vec{c}_i$ and the widths σ_i for the optimal RBF network are summarized in Table 4. In Fig. 2(b), the center indices i of Table 4 are superimposed on the graph. Intermediate approximations generated during the growth of the final RBF network are shown in Fig. 3. The function $f(x, y) = \cos(\pi x)\sin(\pi\sqrt{|y|^3})$ on its definition square $[0.0, +1.0] \times [-0.5, +0.5]$ is shown in Fig. 3(a). To draw the graph the training data $(x_k^{Tr}, y_k^{Tr}, z_k^{Tr})$, $k = 0, \ldots, N_{Tr} - 1$, $z_k^{Tr} = f(x_k^{Tr}, y_k^{Tr})$ was used. The approximations shown in Fig. 3(b)–(d) are the ones realized by the growing RBF network with $m = 5, 10$ and 35 (optimal number of) hidden neurons. The following values were drawn: $(x_k^{Tr}, y_k^{Tr}, z_k^{Tr})$, $k = 0, \ldots, N_{Tr} - 1$, $z_k^{Tr} = g_m(x_k^{Tr}, y_k^{Tr})$. The approximations for $m = 10$ and 35 are very similar. The generalization error difference for both RBF networks is minimal as can also be verified in Fig. 2(a). In practical environments, a threshold could be introduced and used in such a way that when the generalization error decrease is under the given threshold, the growth of the RBF network terminates leading to savings in the computational efforts. In Fig. 2(b), the first 10 centers generated automatically already cover the different areas of the definition square. The incorporation of further centers led to a better coverage, but at the same time led only to a minimal increase of the generalization capability of the resulting RBF network.

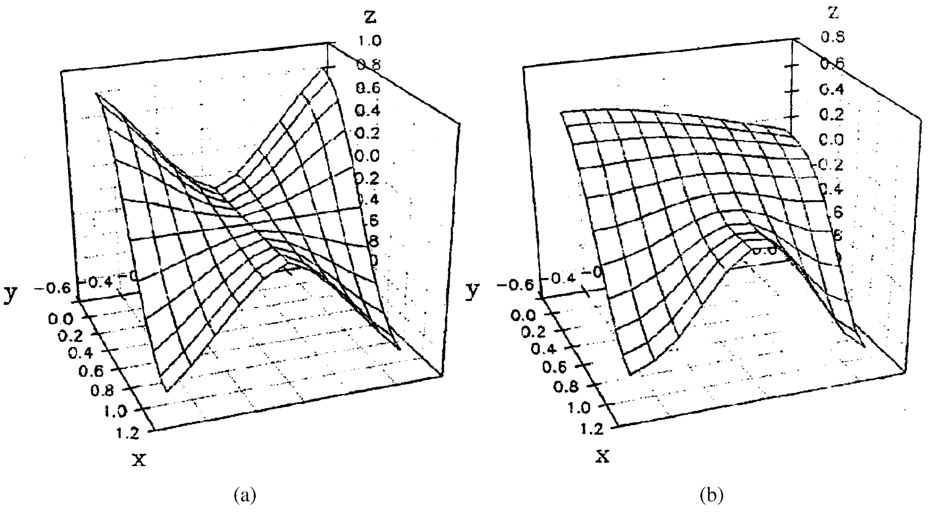
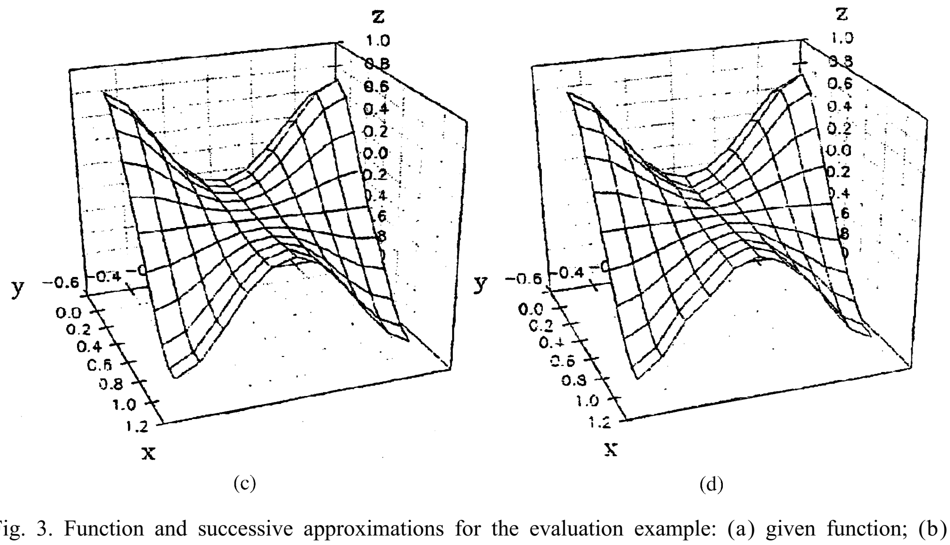

Fig. 3. Function and successive approximations for the evaluation example: (a) given function; (b) network approximation, $m = 5$; (c) network approximation, $m = 10$; (d) network approximation, $m = 35$.

6. Discussion from a search perspective

One further evaluation example serves as the basis for the discussion from the perspective of searching for a solution to the automatic RBF regression network design. To generate the noisy data, the function $f(x) = \sqrt{x}$ is defined on the interval $[0, 1]$. The input space, more precisely the function definition region, is partitioned regularly. In the case of a one-dimensional function $f : [a, b] \to R$ when N elements of the training data set Tr are to be generated, the following partition is used: $x_i = a + id$, $i = 0, \ldots, N-1$, where $d = (b-a)/(N-1)$. This partition includes the interval

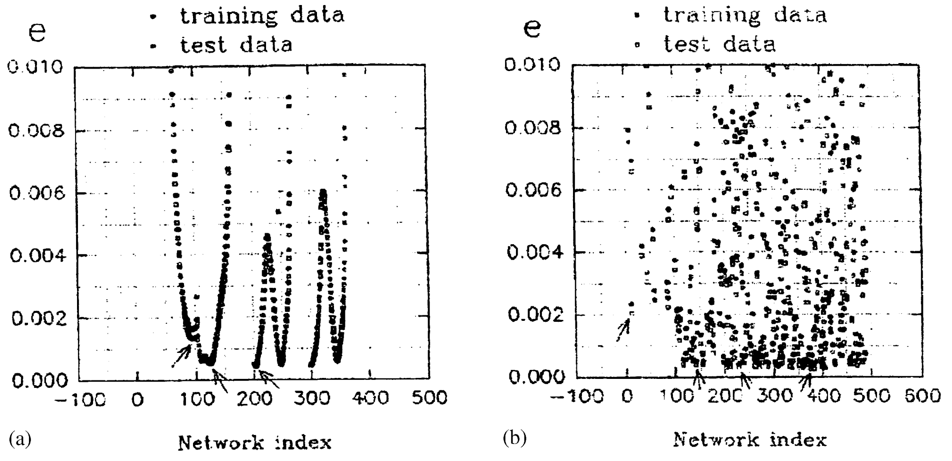

Fig. 4. Automatic generalization optimization for the second evaluation example: (a) without reordering; (b) with random reordering.

ends, $x_0 = a$ and $x_{N-1} = b$. The input components of the test data set are generated according to $x_j = a + (j + 0.5)d$, $j = 0, \ldots, N-2$, so that a space of $0.5d$ between training data inputs and test data inputs is accomodated. The noise-free data are modified to generate the noisy data. The input component remains the same while the output component is modified according to $y_i = f(x_i) + \epsilon_i$, $i = 1, \ldots, N$ and $y_j = f(x_j) + \epsilon_j$, $j = 1, \ldots, M$ for the training data set Tr and the test data set Te, respectively, and for functions of the type $f : R \to R$. Gaussian noise ϵ with mean $\mu = 0$ and standard deviation σ_n is used. Correspondingly, $\epsilon_{i,j} = \sigma_n n_{i,j}$, where $n_{i,j} \sim N(0,1)$, i.e., $n_{i,j}$ is distributed according to the standard normal distribution. For this evaluation example $N = 101$ elements of the training data set and $M = N - 1$ elements of the test data set are generated. The standard deviation of the noise is $\sigma_n = 0.05$.

Fig. 4 visualizes the automatic, monotonic generalization optimization while systematically and automatically designing and training the final RBF network architecture. The x-coordinate of the graphs represents the RBF network index. The y-value represents the approximation error for the training and the test data set, i.e., the memorization and the generalization error, respectively. The training data are taken as given in Fig. 4(a) while in Fig. 4(b) the training data were reordered randomly. For both cases, the test data are taken as given, since the order of the elements does not influence the growth of the final RBF network. The optimal RBF network when the training data are not reordered has $m = 4$ hidden neurons. According to (23) a total of 395 RBF networks were designed and trained. The RBF networks from the index sets $\{1\}$, $\{2, \ldots, 101\}$, $\{102, \ldots, 200\}$, $\{201, \ldots, 298\}$, and $\{299, \ldots, 395\}$ have 1, 2, 3, 4, and 5 hidden neurons, respectively. These sets have 1, 100, 99, 98, and 97 elements corresponding to the same number of RBF networks, respectively. For determining the number of RBF networks that have the same number of hidden neurons see (22) .

Table 5
Minimizing the generalization error, training data without reordering

m	1	2	3	4
p	1	94	125	202
e_{Te}	0.49893218	0.00132397	0.00051568	0.00043095

Table 6
Centers and widths for the second evaluation example, training data without reordering

i	1	2	3	4
z_i	0.00	0.93	0.24	0.02
σ_i	0.02	0.69	0.22	0.02

Table 7
Minimizing the generalization error, training data with random reordering

m	1	2	3	4	5
p	1	11	141	229	373
e_{Te}	0.48702455	0.00203784	0.00037184	0.00028910	0.00025237

The first center is taken as $c_1 = x_1$ from the first element of the training data set. The corresponding width σ_1 is determined using the special width determination algorithm for the first width embedded within the constructive RBF regression network design procedure. Then, iteratively one hidden neuron is selected at a time and added to the RBF network if it leads to a generalization increase. In this evaluation example, this happened after adding a second, a third, and a fourth hidden neuron. A fifth hidden neuron was also tested, but did not bring an improvement in the RBF network generalization capability. Table 5 shows the number m of hidden neurons of the RBF network, the index p of the RBF network that provided a generalization increase, and the new improved generalization error e_{Te}. The values of (p, e_{Te}) for $m = 2, 3, 4$ are shown in Fig. 4(a) with an arrow. The centers and widths for the resulting RBF network based on the training data without reordering are summarized in Table 6.

The optimal RBF network when the training data are reordered randomly has $m = 5$ hidden neurons. RBF networks (491) were designed and trained. The automatic, monotonic generalization optimization was performed in a similar way as for the case when the training data are not reordered until $m = 4$. Then a fifth hidden neuron was added into the RBF network and a sixth hidden neuron was tested, but did not provide for a generalization increase. The corresponding index sets were $\{299, 395\}$ and $\{396, 491\}$, respectively. Table 7 shows the number m of hidden neurons of the RBF network, the index p of the RBF network that provided a generalization increase, and the new improved generalization error e_{Te}. The values of (p, e_{Te}) for $m = 2, \ldots, 5$ are shown in Fig. 4(b) with an arrow. Table 8

Table 8
Centers and widths for the second evaluation example, training data with random reordering

i	1	2	3	4	5
z_i	0.48	1.00	0.22	0.03	0.69
σ_i	0.21	0.31	0.19	0.19	0.21

summarizes the centers and widths for the resulting RBF network based on the training data with random reordering.

The generalization of the RBF network could be increased in this case when the training data was randomly reordered, cf. Table 5, $e_{Te} = 0.00043095$ for $m = 4$ and Table 7, $e_{Te} = 0.00025237$ for $m = 5$. This increase due to random reordering is of course not always possible. In order to become independent of the choice of the first center, N different choices according to $z_1 = x_i$, $i = 1, \ldots, N$ could be carried out and the corresponding generated RBF networks compared to determine the optimal of all of them. With a linear increase of factor N in the computation effort, this could be achieved. By using parallel processing, nearly no increase in the computational time could be achieved if N processors were available and each of them would generate the RBF network corresponding to a different choice of the first center. In many practical situations, even though the generalization error is increased marginally, the final RBF network approximation looks very similar to the one generated by only one choice of the first center.

The dependence on the choice of the first center is easy to see. After a first coverage of the input space, more specifically of the definition interval of the function, by the first center, further centers need to be added to increasingly cover the input space. If a different first center were chosen, then a different set of complementary centers would be needed and this would also lead to a different set of widths determined by the algorithm for automatic width determination. This can be observed comparing Table 6 with Table 8. This example stresses the key ideas behind searching for the best solution. At the local level, there is competition among the prospective new neurons to be incorporated into the network architecture one at a time or per iteration, on the basis of improving the network generalization capability. At the global level, after choosing a different first center, the search for the best solution is carried out on the basis of the same criterium, i.e., the improvement of the evolving network generalization capability.

7. Conclusions

Different approaches that play an important role in the area of automatic design of artificial brains were reviewed in the context of the automatic RBF regression network design. Currently, traditional techniques of pattern recognition, e.g., clustering, constructive procedures based on principles of computational and statistical learning theory, and evolutionary approaches are evolving to form the foundations for further research in this fascinating discipline. The perspective of searching for a solution to the problem at different levels from the originating posture of the optimization problem associated to the learning problem from a computational learning theoretical perspective to the solution of subproblems makes the use of evolutionary approaches a natural choice and in many cases a competitive choice. On the other hand, after posing the optimization problem, the incorporation of requirements specific to the underlying learning problem and goal, i.e., maximization of the network generalization, can guide the search process very effectively. Examples and theoretical analysis of the procedures were used to visualize some of the key ingredients of the automatic design process.

References

[1] T. Bäck, U. Hammel, H.-P. Schwefel, Evolutionary computation: comments on the history and current state, IEEE Trans. Evol. Comput. 1 (1) (1997) 3–17.

[2] J.C. Bezdek, J. Keller, R. Krisnapuram, N.R. Pal, Fuzzy Models and Algorithms for Pattern Recognition and Image Processing, Kluwer Academic Publishers, Dordrecht, 1999.

[3] S.A. Billings, G.L. Zheng, Radial basis function network configuration using genetic algorithms, Neural Networks 8 (6) (1995) 877–890.

[4] C.M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, Oxford, 1999.

[5] D.S. Broomhead, D. Lowe, Multivariable functional interpolation and adaptive networks, Complex Systems 2 (1998) 321–355.

[6] C. Campbell, An introduction to kernel methods, in: R.J. Howlett, L.C. Jain (Eds.), Radial Basis Function Networks: Design and Applications, Physica-Verlag, Wurzbury, 2000, pp. 155–192.

[7] S. Chen, C.F.N. Cowan, P.M. Grant, Orthogonal least squares learning algorithm for radial basis function networks, IEEE Trans. Neural Networks 2 (1991) 302–309.

[8] N. Cristianini, J. Shawe-Taylor, An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods, Cambridge University Press, Cambridge, 2000.

[9] H. de Garis, Implementation and performance-scaling issues concerning the genetic programming of a cellular automata based artificial BRAIN, in: Proceedings of the International Conference on Evolutionary, Orlando, FL, 1994, pp. 1714–1720.

[10] H. de Garis, D. Fogel, M. Conrad, X. Yao, Special issue on evolutionary neural systems. Neurocomputing, in press, this issue.

[11] H. Drucker, C.J.C. Burges, L. Kaufman, A. Smola, V. Vapnik, Support vector regression machines, in: M. Mozer, M. Jordan, T. Petsche (Eds.), Advances in Neural Information Processing Systems 9, The MIT Press, Cambridge, MA, 1997, pp. 155–161.

[12] A. Esposito, M. Marinaro, D. Oricchio, S. Scarpetta, Approximation of continuous and discontinuous mappings by a growing neural RBF-based algorithm, Neural Networks 13 (6) (2000) 651–665.

[13] D.B. Fogel, Evolutionary Computation: Towards a New Philosophy of Machine Intelligence, IEEE Press, New York, 1995.

[14] M. Friedman, A. Kandel, Introduction to Pattern Recognition—Statistical, Structural, Neural, and Fuzzy Logic Approaches—. World Scientific Publishing Co., Singapore, 1999.

[15] F. Girosi, Some extensions of radial basis functions and their applications in artificial intelligence, Comput. Math. Appl. 24 (12) (1992) 61–80.

[16] F. Girosi, Regularization theory, radial basis functions and networks, in: V. Cherkassky, J. Friedman, H. Wechsler (Eds.), From Statistics to Neural Networks, Theory and Pattern Recognition Applications, Springer, Berlin, 1994, pp. 166–187.

[17] F. Girosi, G. Anzellotti, Rates of convergence for radial basis functions and neural networks, in: R. Mammone (Ed.), Artificial Neural Networks for Speech and Vision, Chapman & Hall, London, 1993, pp. 97–113.

[18] D.E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, Reading, MA, 1989.
[19] F. Gruau, Genetic synthesis of Boolean neural networks with a cell rewriting developmental process, in: Proceedings of COGANN-92, International Workshop on Combinations of Genetic Algorithms and Neural Networks, Baltimore, MD, 1992, pp. 55–74.
[20] I. Guyon, P.S.P. Wang (Eds.), Advances in Pattern Recognition Systems using Neural Network Technologies, World Scientific, Singapore, 1994.
[21] F.R. Hampel, E.M. Rochetti, P.J. Rousseeuw, W.A. Stahel, Robust Statistics, Wiley, New York, 1986.
[22] D. Haussler, Decision theoretic generalizations of the PAC model for neural net and other learning applications, Inf. Comput. 100 (1992) 78–150.
[23] S. Haykin, Neural Networks: A Comprehensive Foundation, Macmillan, London, 1994.
[24] R. Hecht-Nielsen, Neurocomputing, Addison-Wesley, Reading, MA, 1990.
[25] J. Hertz, A. Krogh, R. Palmer, Introduction to the Theory of Neural Computation, Addison-Wesley, Reading, MA, 1991.
[26] J.H. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, 1975.
[27] R.J. Howlett, L.C. Jain (Eds.), Radial Basis Function Networks: Design and Applications, Physica-Verlag, Wurzbury, 2000.
[28] P.J. Huber, Robust Statistics, Wiley, New York, 1981.
[29] P.A. Jokinen, A nonlinear network model for continuous learning, Neurocomputing 3 (4) (1991) 157–176.
[30] J. Koza, Genetic Programming, The MIT Press, Cambridge, MA, 1992.
[31] J. Koza, J. Rice, Genetic generation of both the weights and architecture for a neural network, in: Proceedings IJCNN-91, Seattle, WA, 1991, pp. 397–404.
[32] L. Kuncheva, Initializing of an RBF network by a genetic algorithm, Neurocomputing 14 (3) (1997) 273–288.
[33] B.K. Lavine, Pattern Recognition Analysis via Genetic Algorithms and Multivariate Statistical Methods, CRC Press, Boca Raton, FL, 2000.
[34] C.G. Looney, Pattern Recognition Using Neural Networks: Theory and Algorithms for Engineers and Scientists, Oxford University Press, Oxford, 1997.
[35] J. MacQueen, Some methods for classification and analysis of multivariate observations, in: Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, 1967, pp. 281–297.
[36] J. Moody, C.J. Darken, Fast learning in networks of locally-tuned processing units, Neural Comput. 1 (2) (1989) 281–294.
[37] A. Nigrin, Neural Networks for Pattern Recognition, The MIT Press, Cambridge, MA, 1993.
[38] P. Niyogi, F. Girosi, On the relationship between generalization error, hypothesis complexity, and sample complexity for radial basis functions, Neural Comput. 8 (1996) 819–842.
[39] S.K. Pal, P.P. Wang (Eds.), Genetic Algorithms for Pattern Recognition, CRC Press, Boca Raton, FL, 1996.
[40] S.K. Pal, S. Mitra, Neuro-Fuzzy Pattern Recognition: Methods in Soft Computing, Wiley, New York, 1999.
[41] Y.-H. Pao, Adaptive Pattern Recognition and Neural Networks, Addison-Wesley, Reading, MA, 1989.
[42] J. Park, I.W. Sandberg, Approximation and radial-basis-function networks, Neural Comput. 5 (1993) 305–316.
[43] J. Platt, A resource-allocation network for function interpolation, Neural Comput. 3 (1991) 213–225.
[44] T. Poggio, F. Girosi, Networks for approximation and learning, Proc. IEEE 78 (9) (1990) 1481–1497.
[45] B.D. Ripley, Pattern Recognition and Neural Networks, Cambridge University Press, Cambridge, 1996.

[46] V.D. Sánchez A., On the Design of a Class of Neural Networks, J. Network Comput. Appl. 19 (1996) 111–118.
[47] V.D. Sánchez A., Advances towards the automatic design of RBF networks, Int. J. Knowledge-Based Intell. Eng. Systems 1 (3) (1997) 168–174.
[48] V.D. Sánchez A., New Robust Learning Method, Int. J. Smart Eng. System Des. 1 (1998) 223–233.
[49] V.D. Sánchez A., Special Issue on RBF Networks, Neurocomputing 19 (1998) 1–3 and 20 (1998) 1–3.
[50] V.D. Sánchez A., Neural Network Based Pattern Recognition, in: S.K. Pal and A. Pal (Eds.), Pattern Recognition from Classical to Modern Approaches, World Scientific Publishing Co., Singapore, 2001, in press.
[51] R.J. Schalkoff, Pattern Recognition: Statistical, Structural, and Neural Approaches, Wiley, New York, 1992.
[52] B. Schoelkopf, P. Bartlett, A.J. Smola, R. Williamson, Shrinking the tube: a new support vector regression algorithm, in: M.S. Kearns, S.A. Solla, D.A. Cohn (Eds.), Advances in Neural Information Processing Systems 11, The MIT Press, Cambridge, MA, 1999, pp. 330–336.
[53] B. Schoelkopf, C.J.C. Burges, A.J. Smola (Eds.), Advances in Kernel Methods—Support Vector Learning—. The MIT Press, Cambridge, MA, 1999.
[54] H.-P. Schwefel, Evolution and Optimum Seeking, Wiley, New York, 1995.
[55] J. Shawe-Taylor, S. Ben-David, P. Koiran, R. Schapire, Special Issue on Theoretical Analysis of Real-Valued Function Classes, Neurocomputing 29 (1999) 1–3.
[56] N. Sundararajan, P. Saratchandran, L.Y. Wei, Radial Basis Function Neural Networks with Sequential Learning, World Scientific Publishing Co., Singapore, 1999.
[57] L.G. Valiant, A theory of the learnable, Commun. ACM 27 (11) (1984) 1134–1142.
[58] V. Vapnik, The Nature of Statistical Learning, Springer, Berlin, 1995.
[59] V. Vapnik, S.E. Golowich, A. Smola, Support vector method for function approximation, regression estimation, and signal processing, in: M. Mozer, M. Jordan, T. Petsche (Eds.), Advances in Neural Information Processing Systems 9, The MIT Press, Cambridge, MA, 1997, pp. 281–287.
[60] B.A. Whitehead, Genetic evolution of radial basis function coverage using orthogonal niches, IEEE Trans. Neural Networks 7 (6) (1996) 1525–1528.
[61] B.A. Whitehead, T. Choate, Cooperative-competitive genetic evolution of radial basis function centers and widths for time series prediction, IEEE Trans. Neural Networks 7 (4) (1996) 869–880.
[62] X. Yao, Evolving Artificial Neural Networks, Proc. IEEE 87 (1999) 1423–1447.
[63] B. Zhang, H. Mühlenbein, Synthesis of sigma-pi neural networks by the breeder genetic programming, in: Proceedings of the International Conference on Evolutionary, Orlando, FL, 1994, pp. 381–323.
[64] J. Zurada, Introduction to Artificial Neural Systems, PWS Publishing, Boston, MA, 1992.

V. David Sanchez A is a Fellow of the IEEE with the following citation: "For leadership in neural and parallel computation, and pioneering contributions to autonomous space robots" and a member of the American Association for the Advancement of Science. Sanchez founded in 1989 the journal "Neurocomputing" (vol.36-41 to be published in 2001). For more information please visit http://www.elsevier.com/locate/neucom. Since the journal foundation Sanchez has served as the Editor-in-Chief.

Recently, Sanchez helped NASA JPL in Pasadena, California in the framework of the most advanced spacecraft for deep-space exploration ever built, X2000/MDS for the Integrated First Delivery Project IFDP to Jupiter-Europa and Pluto. On the industrial side, Sanchez has been the Chief Technology Officer (CTO) of Thuris Corporation, a world leading company in Newport Beach, California, committed to the application of brain research and advanced computational methods. He has also acted in advisory capacity to the embedded systems, telecom, and datacom industries as well as to the laser-based communications and cable-TV industry building a new generation of optics, RF, digital hardware and software hybrid.

Previously he headed the systems engineering activities to design and develop System-On-Chip (SoC) with applications in the multimedia and telecommunications markets, was involved in the development of one of the most complex communications SoC ever built, highly complex real-time avionics systems including signal processing algorithms and neurochips. Sanchez has held a honorary computer science professorship position at the University of Miami, a senior research scientist position at the German Aerospace Research Establishment, and industry R&D positions among other at Hewlett Packard, Siemens AG, Conexant Broadcom, and WindRiver Systems.

Sanchez was the leader of the robot vision experiment for the STS 55 flight of the spaceshuttle Columbia. For that purpose he designed and developed the image understanding algorithms and a real-time architecture for the task of grasping floating objects under u-gravity. The architecture was a part of the telerobotics ground station and was built using parallel and signal processors with three industry companies. Sanchez was the invited expert for camera system design for the mars mission proposal MARSNET from the European Space Agency (ESA). Sanchez delivered the original design of a domestic satellite communications system launched with the Ariane 4 working in cooperation with Daimler Aerospace (former MBB and Dornier).

Sanchez has served as Associate Editor of other journals and has numerous conference proceedings, journal, and book chapter publications in the areas of parallel computing, neural networks, image processing, computer vision, signal processing, and robotics. Sanchez has given plenary/keynote talks and has served as program committee member at several international conferences. Sanchez has also given invited presentations at NASA JPL (Jet Propulsion Laboratory) in Pasadena, California, NASA ARC (Ames Research Center) in Moffett Field, California, the Ohio Aerospace Institute (OAI), and the main ESA laboratory ESTEC in Noordwijk, the Netherlands. More recently, Sanchez won a U.S. nation-wide award to design microspacecraft for a series of mars missions (2003,05,07) delivering it to NASA JPL in 1999.