A New Clustering Technique for Function Approximation

Jesús González, Ignacio Rojas, Héctor Pomares, Julio Ortega, *Member, IEEE*, and Alberto Prieto, *Member, IEEE*

***Abstract*—To date, clustering techniques have always been oriented to solve classification and pattern recognition problems. However, some authors have applied them unchanged to construct initial models for function approximators. Nevertheless, classification and function approximation problems present quite different objectives. Therefore it is necessary to design new clustering algorithms specialized in the problem of function approximation. This paper presents a new clustering technique, specially designed for function approximation problems, which improves the performance of the approximator system obtained, compared with other models derived from traditional classification oriented clustering algorithms and input–output clustering techniques.**

***Index Terms*—Clustering techniques, function approximation, model initialization.**

I. Introduction

CLUSTERING techniques were originally conceived by Aristotle and Theophrastos in the fourth century B.C. and in the 18th century by Linnaeus [23], but it was not until 1939 when one of the first comprehensive foundations of these methods was published [45]. These techniques have traditionally been applied to classification problems [14], where the task to solve is how to organize observed data into meaningful structures. More recently, these kinds of algorithms have been applied to new paradigms such as fuzzy systems (FS) [6] and artificial neural networks (ANNs) [20].

In 1969, Ruspini [40] and in 1973, Dunn [9], proposed new techniques of clustering to perform a fuzzy partition of the input space. This work was generalized by Bezdek in 1981, introducing fuzzy clustering [3]. Some variations of these algorithms have appeared in [6]. Such fuzzy clustering algorithms were used to obtain the knowledge base for fuzzy classifiers trained from input–output examples in [1] and [30] and to obtain neuro-fuzzy systems in [1], [37], and [41].

Training algorithms for radial basis function neural networks (RBFNNs) also use a clustering step to initialize the RBF centers. In [20] these techniques are used to construct initial neural networks as part of a growing algorithm which iteratively fits an RBFNN to a set of example data by splitting the RBFs with higher classification errors. In [22], a training algorithm based on the minimum description length (MDL) principle is proposed and in [47], some statistical analysis is applied to the training examples in order to describe them as a whole and to facilitate a global learning algorithm. Although these algorithms are quite different, all of them use clustering techniques to create the initial model.

All of these approaches have proved successful in classification problems. Nevertheless, in recent papers some authors have attempted to solve function approximation problems by means of clustering techniques without making significant changes to them [26], [44]. There are, however, some particular concepts of function approximation and classification problems which should be taken into account:

1) In classification problems, the output variable takes values in a finite label set which is defined *a priori*, while in function approximation problems the output variable can take any of the infinite values within an interval of real numbers. In other words, the output variable is discrete for classification and continuous for function approximation problems.
2) In a function approximation problem, output values different from the optimum ones may be accepted if they are "sufficiently close" to them. These variations might only produce a worthless approximation error. This behavior is not desirable for a classifier. In classification problems the output labels may not be related by any distance operator and an output response different from the correct one may be unacceptable.
3) Clustering techniques (originally proposed for classification) do not take into account the interpolation properties of the approximator system, since this is not necessary in a classification problem.

Therefore, new clustering algorithms specially fitted to the problem of function approximation, which take these differences into account, would improve the performance of the approximator.

This paper describes a new clustering technique specially designed for function approximation problems (CFA). This new technique analyzes the output variability of the target function during the clustering process and augments the number of prototypes in those input zones where the target function is more variable, increasing the variance explained by the approximator. This change in the behavior of the clustering algorithm improves the performance of the approximator system obtained, compared with other models derived from traditional classification oriented clustering algorithms.

The organization of this paper is as follows: Section II presents a brief review of some traditional clustering algorithms. Section III describes the proposed function approximation oriented clustering algorithm in detail. Some comparative results are shown in Section IV and final conclusions are discussed in Section V.

Manuscript received December 4, 2000; revised June 11, 2001. This work has been supported in part by the Spanish Ministerio de Ciencia y Tecnología under Projects TIC2000-1348 and DPI2001–3219.

The authors are with the Department of Computer Architecture and Computer Technology, University of Granada, Campus de Fuentenueva, E–18071 Granada, Spain.

Publisher Item Identifier S 1045-9227(02)00345-4.

II. Clustering Algorithms

Traditional clustering algorithms attempt to split a given set of vectors $X = \{\boldsymbol{x}^i\colon i = 1, \ldots, n\}$ into an *a priori* known number c of subgroups or clusters, thus producing a Voronoi partition $P = \{\boldsymbol{p}^1, \ldots, \boldsymbol{p}^c\}$ [11] of the training set X. These clustering algorithms can be divided into two conceptually different families: input clustering and input–output clustering.

A. Input Clustering

This kind of algorithm allocates the prototypes according to an analysis of the input training sample vectors, completely ignoring the information about the dependent output variable. Important examples of this family are the hard c-means, the fuzzy c-means, and the ELBG algorithms, described below.

1) HCM (Hard c-Means) Algorithm: This algorithm was originally proposed by Duda and Hart in [8]. It assigns each one of the training vectors $\boldsymbol{x}^i$ to the cluster whose prototype $\boldsymbol{p}^j$ is its closest neighbor. After all the training vectors are assigned, the new prototypes are recalculated as the centroids of the training vectors assigned to each cluster of the new partition. The certainty of the assignment of the training example $\boldsymbol{x}^i$ to the cluster defined by the jth prototype is measured by the membership function $\mu_j(\boldsymbol{x}^i)$, defined as

$$\mu_j(\boldsymbol{x}^i) = \begin{cases} 1 & \text{if } \|\boldsymbol{x}^i - \boldsymbol{p}^j\|^2 < \|\boldsymbol{x}^i - \boldsymbol{p}^l\|^2, \quad \forall l \neq j \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

and the centroid of each cluster is calculated by

$$\boldsymbol{p}^j = \frac{\sum_{i=1}^{n} \mu_j\left(\boldsymbol{x}^i\right) \boldsymbol{x}^i}{\sum_{i=1}^{n} \mu_j\left(\boldsymbol{x}^i\right)} \tag{2}$$

To stop the algorithm, the following measure of the distortion of the actual partition is used:

$$d = \sum_{j=1}^{c} d_j \tag{3}$$

where d_j represents the local distortion produced by the jth cluster

$$d_j = \sum_{i=1}^{n} \mu_j\left(\boldsymbol{x}^i\right) \|\boldsymbol{x}^i - \boldsymbol{p}^j\|^2 \tag{4}$$

Thus, the algorithm stops when the change in the distortion is below a given value ϵ.

2) FCM (Fuzzy c-Means) Algorithm: This algorithm [3] considers each cluster as a fuzzy set [46]. Making the partition in this way implies that a vector may be assigned to some clusters with different degrees of certainty. The new membership function, which takes values in the interval [0,1] is defined as

$$\mu_j(\boldsymbol{x}^i) = \left(\sum_{l=1}^{c} \left(\frac{\|\boldsymbol{x}^i - \boldsymbol{p}^j\|^2}{\|\boldsymbol{x}^i - \boldsymbol{p}^l\|^2} \right)^{1/r-1} \right)^{-1} \tag{5}$$

where $r \in (1, \infty)$ determines the fuzziness of the partition produced by the algorithm. If $r \to 1_+$, then the resulting partition asymptotically approaches a hard or crisp partition. On the other hand, the partition becomes a maximally fuzzy partition if $r \to \infty$.

The equation used to obtain the centroids of each cluster must be adapted to the new membership function, becoming

$$\boldsymbol{p}^j = \frac{\sum_{i=1}^{n} \left[\mu_j(\boldsymbol{x}^i)\right]^r \boldsymbol{x}^i}{\sum_{i=1}^{n} \left[\mu_j(\boldsymbol{x}^i)\right]^r}. \tag{6}$$

Criterion (3) is also used to stop this algorithm, but in this case d_j is obtained with the expression

$$d_j = \sum_{i=1}^{n} \left[\mu_j(\boldsymbol{x}^i)\right]^r \|\boldsymbol{x}^i - \boldsymbol{p}^j\|^2. \tag{7}$$

3) ELBG (Enhanced LBG) Algorithm: The algorithms described above perform a local search of the nearest suboptimum quantizer to the initial configuration. Often, this is far from being an acceptable solution. The way in which these algorithms make the prototype adjustment every iteration only permits the prototypes to “move” through contiguous regions. This implies that a bad initialization could lead to the impossibility of finding a good partition. To overcome this drawback, Russo and Patanè proposed the ELBG algorithm [42], an enhanced version of the HCM algorithm. The objective of this algorithm is to obtain a final quantizer where each cluster makes an equal contribution to the total distortion [10]. To achieve it, a prototype migration step capable of detecting less useful clusters and migrating them to regions with a higher density of input vectors is incorporated. This modification allows the algorithm to escape from local minima and to obtain a prototype allocation independent of the initial configuration.

B. Input–Output Clustering

As the design objective in a function approximation problem is to construct the relationships between the input–output examples, it is necessary to incorporate the output variable into the clustering algorithm. This process can be performed in different ways: in this paper we have analyzed two of them, the alternating cluster estimation (ACE) [39] and the conditional fuzzy clustering algorithm [31], [32].

1) ACE Algorithm: Traditionally, membership functions and prototype locations are restricted to particular shapes and positions determined by the updating of equations derived from the objective function for a clustering model. Nevertheless, the final user might be interested in the use of a certain type of membership function which could be better fitted to the problem in question. In [39], an ACE is proposed; this uses an alternating iteration architecture, but membership and prototype functions are selected directly by the user.

The way ACE incorporates the available outputs into the training set is by constructing a new training set $Z = \{\boldsymbol{z}^i\colon i = 1, \ldots, n\} \subset \Re^{n_i + n_o}$, where each data vector $\boldsymbol{z}^i$ is a concatenation of an input vector $\boldsymbol{x}^i \in \Re^{n_i}$ with its corresponding output vector $\boldsymbol{y}^i \in \Re^{n_o}$. Once this new training set has been generated, ACE can be used to implement well-known clustering algorithms such as HCM and FCM, or to design new clustering techniques better fitted to a particular problem. For example, in [39] an instance of ACE is proposed called *dancing cones* (DC), based on the use of hyper-conic membership functions shown in (8) at the bottom of the page.

Once the final configuration is obtained, the projections of these hyper-cones are isosceles triangles that can be used to form a fuzzy model [24] with IF-THEN rules.

2) CFC (Conditional Fuzzy Clustering) Algorithm: The main objective of this algorithm [31], [32] is to develop clusters while preserving the homogeneity of the clustered patterns. This means that samples belonging to the same cluster must meet a similarity criterion in the input and also in the output space.

The CFC algorithm is based mainly on the FCM algorithm, but introduces the concept of context-sensitivity, which requires the output variable of a cluster to satisfy a particular condition. This condition or context (termed $\boldsymbol{\mathcal{A}}$) can be treated as a fuzzy set, with $\boldsymbol{\mathcal{A}}$ being defined via the corresponding membership function. The new μ_j proposed in [32] is defined as

$$\mu_j(\boldsymbol{x}^i) = \frac{g_i}{\sum_{l=1}^{c} \left(\frac{\|\boldsymbol{x}^i - \boldsymbol{p}^j\|^2}{\|\boldsymbol{x}^i - \boldsymbol{p}^l\|^2} \right)^{1/r-1}} \tag{9}$$

where g_i describes a level of involvement of $\boldsymbol{x}^i$ in the constructed cluster P_j. The clustering problem can be reformulated as one of making groups of input data, taking into account that their associated output must be $\boldsymbol{\mathcal{A}}$, with the restriction $\boldsymbol{\mathcal{A}}$ being expressed as a fuzzy set. Therefore, the constraint g_i maintains the requirement that if the ith element (pattern) of the data set is considered not to belong to the context $\boldsymbol{\mathcal{A}}$ and thus its membership degree is null ($\boldsymbol{\mathcal{A}}(y^i) = 0$), then this element is not considered in the calculation in the prototype of the cluster [32]. The computation of the prototypes is performed as before, using (6).

An important consequence of the use of this method is that the computational complexity is reduced by splitting the original problem into a series of context-driven clustering problems, which are usually represented as trapezoidal fuzzy sets designed by an expert human operator. This feature makes it difficult to compare this method with other approaches and means it cannot be automated.

III. CFA: A New Clustering Technique for Function Approximation

Although the above clustering algorithms have been widely used to make the initial models of function approximators, they were initially designed for classification problems. The output space in both problems is quite different, being infinite and continuous for the former but finite and discrete for classifiers. Another important difference is the final objective of the model, which is the interpolation of a target function f for new input vectors in function approximation and the assignment of unknown examples to a set of *a priori* defined set of labels in classification.

The objective of a function approximation oriented clustering algorithm is to increase the density of prototypes in the input areas where the target function presents a more variable response, rather than just in the zones where there are more input examples. The former approach is more adequate for function approximation problems because it helps to minimize the approximation error by means of reducing the output variance not explained by the model, while the latter one, which corresponds to the goal of most of previous clustering techniques, does not make much sense when the problem is to build a function approximator. As all clustering algorithms, CFA reveals the structure of training data in the input space, but it also preserves the homogeneity in the output responses of data belonging to the same cluster. To carry out this task, CFA incorporates a set $O = \{o^1, \ldots, o^c\}$ representing an estimation of the output response of each cluster. The value of each o^j is calculated as a weighted average of the output responses of the training data belonging to cluster P_j, as will be explained in Section III-B. The objective function to be minimized is

$$d = \frac{\sum_{j=1}^{c} \sum_{\boldsymbol{x}^i \in P_j} \|\boldsymbol{x}^i - \boldsymbol{p}^j\|^2 \omega_{ij}}{\sum_{j=1}^{c} \sum_{\boldsymbol{x}^i \in P_j} \omega_{ij}} \tag{10}$$

where ω_{ij} weights the influence of each training example $\boldsymbol{x}^i$ in the final position of the jth prototype. This index informs about the existing controversy between the expected output for the prototype $\boldsymbol{p}^j$ and the output of the training example $\boldsymbol{x}^i$. The greater the distance between the expected output of $\boldsymbol{x}^i$ and the estimated output of the cluster P_j it belongs to, the greater the influence of $\boldsymbol{x}^i$ in the final result. Mathematically, ω_{ij} is defined as

$$\omega_{ij} = \frac{|f(\boldsymbol{x}^i) - o^j|}{\max_{l=1}^{n} \{f(\boldsymbol{x}^l)\} - \min_{l=1}^{n} \{f(\boldsymbol{x}^l)\}} + \mu_{\min} \tag{11}$$

with $\mu_{\min} > 0$. The first term of the sum calculates a normalized distance (in the range [0,1]) between $f(\boldsymbol{x}^i)$ and o^j and the second term is a minimum contribution threshold (when there is no controversy ω_{ij} is equal to $\mu_{\min}$). As $\mu_{\min}$ decreases, CFA forces the prototypes to concentrate on input zones where the output variability is greater. This action is justified because it preserves the homogeneity in the output space of the points $\boldsymbol{x}^i$ belonging to each cluster P_j by means of minimizing the distance of $f(\boldsymbol{x}^i)$ with respect to o^j. On the contrary, if $\mu_{\min}$ increases, CFA pays more attention to the input space, acquiring the behavior of HCM for a sufficiently large value of $\mu_{\min}$. This parameter is studied in detail in Section IV-A.

Once c and ϵ have been fixed, the basic organization of the proposed algorithm consists, as shown in Fig. 1, of the iteration of three main steps: the partition of the training examples, the updating of the prototypes and their estimated outputs and the migration of prototypes from clusters with lower distortions to clusters with bigger distortions.

$$\mu_j(\boldsymbol{z}^i) = \begin{cases} 1 - \left(\frac{\|\boldsymbol{z}^i - \boldsymbol{p}^j\|}{r_j} \right)^{\alpha}, & \text{for } \|\boldsymbol{z}^i - \boldsymbol{p}^j\| \le r_j, \quad \alpha > 0. \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

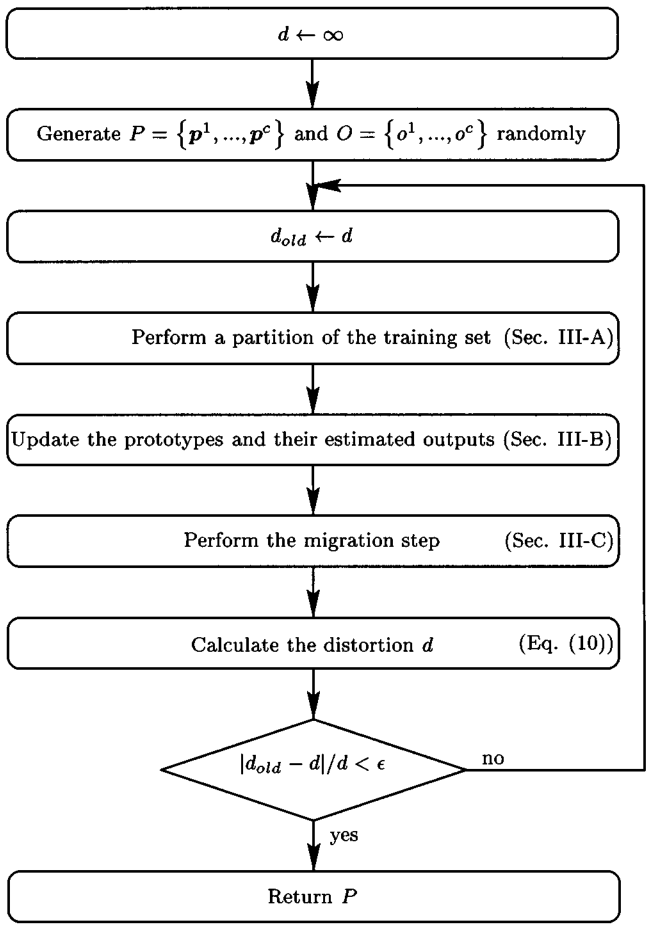

$d \leftarrow \infty$
Generate $P = \{p^1, ..., p^c\}$ and $O = \{o^1, ..., o^c\}$ randomly
$d_{old} \leftarrow d$
Perform a partition of the training set (Sec. III-A)
Update the prototypes and their estimated outputs (Sec. III-B)
Perform the migration step (Sec. III-C)
Calculate the distortion d (Eq. (10))
$|d_{old} - d|/d < \epsilon$
no
yes
Return P

Fig. 1. General description of CFA.

A. Partition of the Training Data

The partition of training data is performed as in HCM, using (1). This step produces a Voronoi partition of the training data

$$X = P_1 \cup \cdots \cup P_c \tag{12}$$

where

$$P_j = \{\boldsymbol{x}^i \in X \colon \mu_j(\boldsymbol{x}^i) = 1\} \tag{13}$$

B. Updating of the Prototypes and Their Estimated Outputs

After partition, the prototypes and their estimated outputs must be updated. The updating is carried out by an iterative process that updates $\boldsymbol{p}^j$ as the weighted mean of the training data belonging to each cluster and o^j as the weighted mean of their output responses until convergence is met. The expressions used to update $\boldsymbol{p}^j$ and o^j are

$$\boldsymbol{p}^j = \frac{\sum_{\boldsymbol{x}^i \in P_j} \boldsymbol{x}^i \omega_{ij}}{\sum_{\boldsymbol{x}^i \in P_j} \omega_{ij}} \tag{14}$$

$$o^j = \frac{\sum_{\boldsymbol{x}^i \in P_j} f(\boldsymbol{x}^i) \omega_{ij}}{\sum_{\boldsymbol{x}^i \in P_j} \omega_{ij}} \tag{15}$$

and the algorithm to perform this step is as follows:

```
$d' \leftarrow \infty$
repeat
    $d'_{\text{old}} \leftarrow d'$
    Update $\boldsymbol{p}^j$ using (14),    $j = 1, \ldots, c$
    Update $o^j$ using (15),    $j = 1, \ldots, c$
    Update $\omega_{ij}$ using (11),
$i = 1, \ldots, n,\quad j = 1, \ldots, c$
    Calculate $d'$ using (10)
until $|d'_{\text{old}} - d'|/d' < \epsilon$
```

The updating step has to be an iterative process because $\boldsymbol{p}^j$ and o^j are interdependent. For the updating of $\boldsymbol{p}^j$ the last value of o^j is used and so the updating must iterate until an equilibrium is reached and $\boldsymbol{p}^j$ converges to the right position for each cluster P_j. This state is reached when the updating distortion d' does not vary significantly in two consecutive iterations.

C. Migration of Prototypes

CFA also incorporates a random migration step to avoid local minima. Randomization has been widely used to break the curse of dimensionality [43], that is, the exponential growth of the search space as a function of the dimensionality of the problem [2]. There exist well-known techniques based on the application of random changes to explore the search space, like simulated annealing (SA) [21], genetic algorithms (GAs) [16] and tabu search (TS) [12], [13]. This kind of technique does not guarantee the global optimum, but can be used to obtain a sufficiently good solution in a reasonable time. Moreover, it is usually possible to improve the quality of the solution found if more processing time can be used.

In this case, randomization is incorporated in CFA by a migration step which moves prototypes allocated in input zones where the target function is stable to zones where the output variability is higher. This migration step is necessary because the iteration of the partition and updating steps only moves prototypes locally, thus converging to the nearest local optimum from the initial configuration. But the objective of the proposed algorithm is also to detect those input zones where, due to the target function variability, it is necessary to increase the density of prototypes. To detect such zones, the migration step is based on the utility index of a prototype, defined as

$$u_j = \frac{d_j}{\bar{d}} \tag{16}$$

where d_j is the contribution of the jth cluster to the total distortion

$$d_j = \frac{\sum_{\boldsymbol{x}^i \in P_j} \|\boldsymbol{x}^i - \boldsymbol{p}^j\|^2 \omega_{ij}}{\sum_{j=1}^{c} \sum_{\boldsymbol{x}^i \in P_j} \omega_{ij}} \tag{17}$$

and $\bar{d}$ is the mean distortion of the current configuration

$$\bar{d} = \frac{d}{c}. \tag{18}$$

In an optimal vector quantization, each cluster makes an equal contribution to the total distortion [10]. This means that the objective is to find a configuration in which all prototypes have

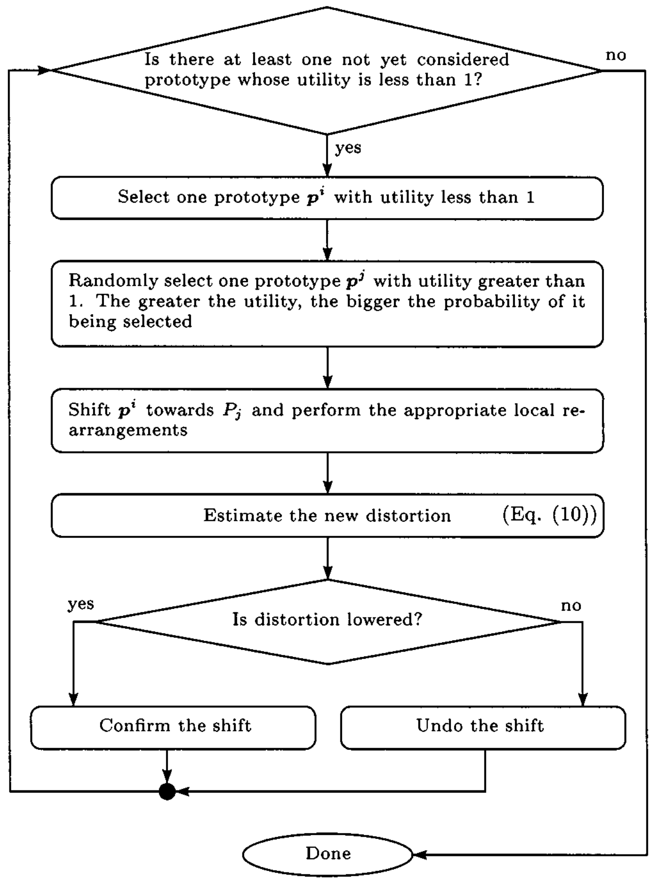
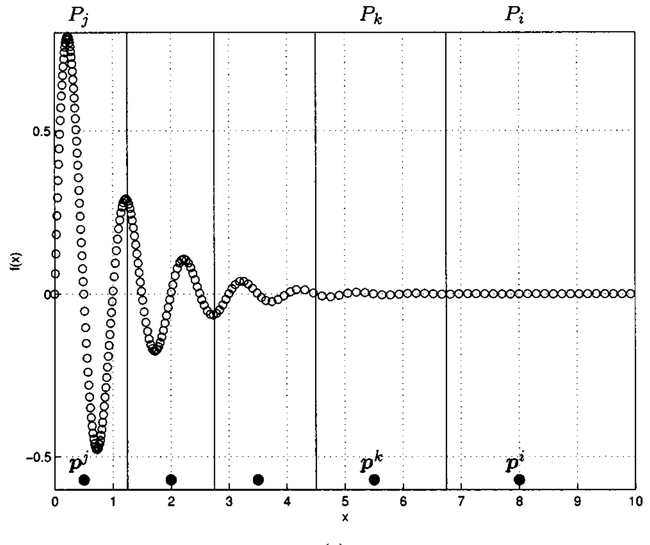

Is there at least one not yet considered prototype whose utility is less than 1?
no
yes
Select one prototype p^i with utility less than 1
Randomly select one prototype p^j with utility greater than 1. The greater the utility, the bigger the probability of it being selected
Shift p^i towards P_j and perform the appropriate local rearrangements
Estimate the new distortion (Eq. (10))
Is distortion lowered?
yes
no
Confirm the shift
Undo the shift
Done

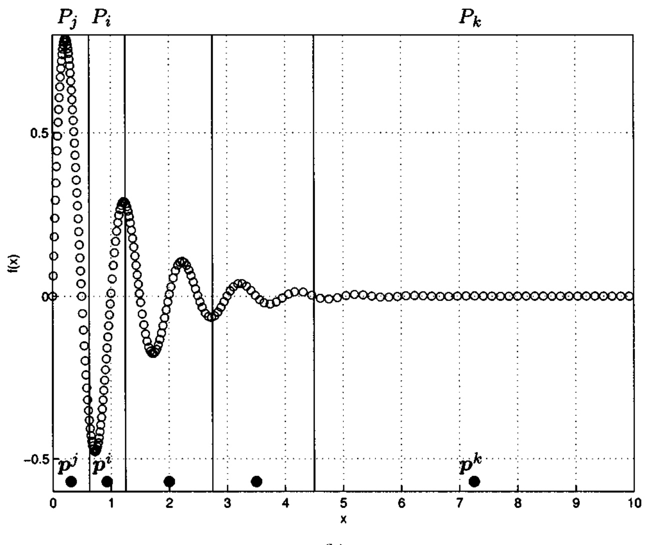
Fig. 2. Migration of prototypes.

utility index equal to one. Thus, the migration procedure (see Fig. 2) shifts prototypes $\boldsymbol{p}^i$ with utility index less than one toward clusters P_j with utility index greater than one. Once $\boldsymbol{p}^i$ has been selected to be migrated, the destination cluster P_j is chosen randomly among all the clusters with utility index greater than one and clusters with greater distortions have more likelihood of being chosen. These shifts make the algorithm independent of the initial configuration and provide it with a mechanism to escape from local optima.

As shown in Fig. 3, the shifting of $\boldsymbol{p}^i$ toward P_j affects three clusters, P_i, P_j, and P_k, the latter being the closest cluster to P_i. Prototypes $\boldsymbol{p}^i$ and $\boldsymbol{p}^i$ are allocated in the principal diagonal of the hyper-box containing the cluster P_j. This diagonal is divided into three parts, with the side segments being equal to half of the central one and the two prototypes are initialized at the extremes of the central segment. After this movement, it is necessary to perform some local rearrangements to the affected clusters P_i, P_j and P_k. P_j is split into two new clusters P_i' and P_j' by running a local HCM algorithm with $c = 2$ with the input vectors in P_j and the prototypes $\boldsymbol{p}^i$ and $\boldsymbol{p}^j$. The examples belonging to cluster P_i, now without a prototype, are assigned to cluster P_k, thus obtaining the new cluster P_k'. Once the three new clusters P_i', P_j' and P_k' have been created, the updating step (Section III-B) is executed to obtain their prototypes and estimated outputs. After finishing these local rearrangements, the migration of the prototype is only accepted if the total distortion is lowered, otherwise it is rejected.



(a)



(b)

Fig. 3. Migration of $\boldsymbol{p}^i$ toward cluster P_j. (a) Situation before the migration and (b) prototype position and clusters after the local rearrangements.

From the convergence point of view, CFA is a weighted version of HCM where each training vector is given a weight depending on an output variability criterion. CFA reaches the state of convergence when the movement of the prototypes is insignificant, as does HCM. The objective for both techniques is the same: to reach a configuration in which each prototype makes an equal contribution to the total distortion. The difference is that the distortion equation used for HCM produces a distribution of the prototypes according to the density of examples in the input space and the new distortion equation designed for CFA concentrates more prototypes in those input regions where the output response is more variable. The latter strategy provides a better fit to function approximation problems and so the posterior application of a local minimization algorithm to the model such as the conjugate gradient, Newton–Rapshon, or Levenberg–Marquardt methods [7], [17], [34] is assumed to achieve a better approximation error, as will be shown in the next section.

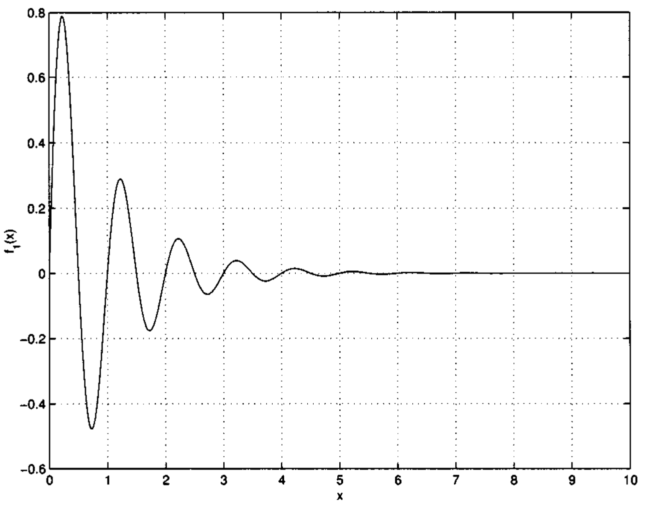


Fig. 4. Target function f_1.

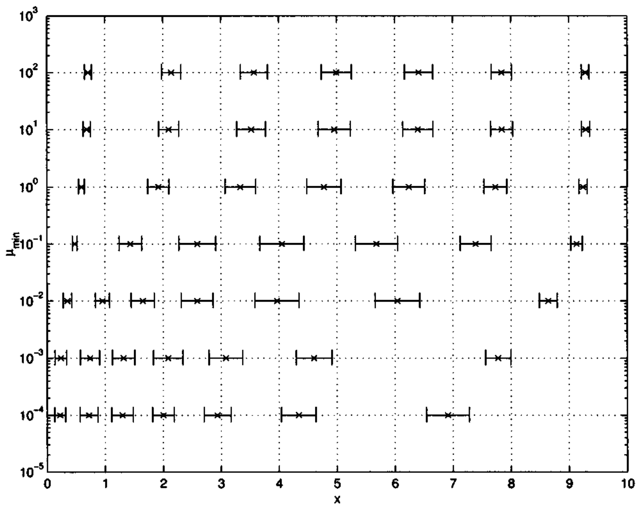


Fig. 5. Effect of $\mu_{\min}$ in the allocation and deviations of prototypes.

IV. Results

As an example, the CFA algorithm was applied to initialize the RBF centers of an RBFNN. The functioning of this model can be summarized by the following equation:

$$F(\boldsymbol{x}) = \sum_{j=1}^{c} \omega_j \phi_j(\boldsymbol{x}) \tag{19}$$

where ϕ_j is an RBF and ω_j is its associated weight. Basis functions are implemented using Gaussian functions

$$\phi_j(\boldsymbol{x}) = \exp\left(\frac{-\|\boldsymbol{p}^j - \boldsymbol{x}\|^2}{\sigma_j^2}\right). \tag{20}$$

This kind of initialization will allocate more RBFs in those input areas where the target function presents a higher output variability. Once the location of the RBFs is obtained, the following step is to fix a width for each one, in order to obtain some overlapping between them and to produce a model with a continuous and smooth output. Although the use of the same width for all the RBFs has proved sufficient to obtain a universal approximator [28], [29], a different width for each basis function improves the model performance [27], [36]. Thus, the RBF width values have been fixed using the closest RBF heuristic [26].

After establishing all the parameters concerning the RBFs, the RBFNN becomes a linear model and its weights can be calculated optimally using pseudo-inverse methods [26], [35] like orthogonal least squares (OLS) [4], [5] or singular value decomposition (SVD) [19].

The application of these three steps (getting the location of the RBFs, fixing their widths and calculating the net weights) produces an initial model that can be fine tuned using a minimization algorithm to adjust its nonlinear parameters. This last step reduces the approximation error of the model iteratively until the nearest local minimum to the initial configuration is reached [15], [33]. The algorithm chosen in this paper to adjust the parameters of the model is the Levenberg–Marquardt minimization algorithm [7], [25] because it combines information of first and second derivatives of the error function to make the convergence process faster.

TABLE I
Mean and Standard Deviation of the Approximation Error and the Time (in Seconds) Required to Approximate f_1 for Each Value of $\mu_{\min}$

$\mu_{\min}$	NRMSE (dev)	Time (dev)
10^2	0.964 (0.004)	0.50 (0.20)
10^1	0.963 (0.004)	0.53 (0.25)
10^0	0.956 (0.004)	0.59 (0.20)
10^{-1}	0.942 (0.009)	1.31 (0.43)
10^{-2}	0.846 (0.092)	8.8 (3.8)
10^{-3}	0.666 (0.138)	56 (39)
10^{-4}	0.647 (0.128)	359 (255)

A. Effect of $\mu_{\min}$

The parameter $\mu_{\min}$ controls how CFA deals with the output variability in the training data. CFA can converge to a great variety of prototype allocations depending on the value of this parameter. A large value for $\mu_{\min}$ implies that CFA pays more attention to input space than to output variability and as $\mu_{\min}$ is lowered, the variability of the target function has a greater influence on the algorithm behavior. To gain an insight into the role of $\mu_{\min}$ in the algorithm functioning, an experiment was designed in which $\mu_{\min}$ takes the values $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^0, 10^1, 10^2\}$ and for each one of these values, we ran CFA 25 times to approximate the target function

$$f_1(x) = \frac{\sin(2\pi x)}{e^x}, \quad x \in [0, 10] \tag{21}$$

with seven prototypes and a training set of 1000 samples of f_1 generated by evaluating inputs taken uniformly from the interval [0,10]. Fig. 5 shows a graph of the mean positions and deviations of the prototypes for each different value of $\mu_{\min}$. This graph reveals that CFA begins to move prototypes to the input zones where f_1 is more variable when $\mu_{\min}$ is less than one. This effect is due to the design of (11), which assigns a minimum weight of $\mu_{\min}$ for examples in input regions where the target function

TABLE II
MEAN AND STANDARD DEVIATION OF THE APPROXIMATION ERROR JUST AFTER THE INITIALIZATION PROCEDURE USING RBFNNS FROM FOUR TO TEN PROTOTYPES TO APPROXIMATE f_1

c	NRMSE (dev)			
	HCM	FCM	ELBG	CFA
4	0.984 (0.003)	0.990 (7E-5)	0.984 (0.005)	0.952 (0.001)
5	0.979 (0.004)	0.988 (9E-5)	0.979 (0.001)	0.928 (0.011)
6	0.972 (0.005)	0.985 (2E-4)	0.970 (0.007)	0.813 (0.128)
7	0.964 (0.007)	0.983 (4E-4)	0.962 (0.002)	0.781 (0.052)
8	0.961 (0.006)	0.980 (3E-4)	0.960 (0.011)	0.537 (0.007)
9	0.947 (0.013)	0.978 (0.001)	0.954 (0.004)	0.514 (0.070)
10	0.940 (0.015)	0.976 (0.001)	0.945 (0.006)	0.509 (0.113)

TABLE III
MEAN AND STANDARD DEVIATION OF THE APPROXIMATION ERROR AFTER THE LOCAL MINIMIZATION PROCEDURE USING RBFNNS FROM FOUR TO TEN PROTOTYPES TO APPROXIMATE f_1

c	NRMSE (dev)			
	HCM	FCM	ELBG	CFA
4	0.851 (0.016)	0.835 (0.205)	0.211 (0.118)	0.176 (0.051)
5	0.818 (0.034)	0.812 (0.341)	0.202 (0.132)	0.081 (0.028)
6	0.759 (0.263)	0.783 (0.317)	0.152 (0.122)	0.090 (0.008)
7	0.344 (0.281)	0.248 (0.115)	0.111 (0.072)	0.081 (0.016)
8	0.227 (0.368)	0.150 (0.162)	0.093 (0.064)	0.053 (0.028)
9	0.252 (0.386)	0.300 (0.160)	0.073 (0.057)	0.056 (0.027)
10	0.087 (0.100)	0.285 (0.334)	0.064 (0.039)	0.047 (0.015)

is constant and a maximum weight of $1+\mu_{\min}$ for examples in more variable zones.

Table I shows the mean and standard deviation of the approximation error of the models obtained by CFA and of the computation time needed to reach convergence. The approximation error of each model is measured using the normalized root mean squared error (NRMSE)

$$\mathrm{NRMSE} = \sqrt{\frac{\sum_{i=1}^{n}\left(f(\boldsymbol{x}^i) - F(\boldsymbol{x}^i)\right)^2}{\sum_{i=1}^{n}\left(f(\boldsymbol{x}^i) - \bar{f}\right)^2}} \tag{22}$$

where $F(\boldsymbol{x}^i)$ is the function approximator output of the input vector $\boldsymbol{x}^i$ and $\bar{f}$ is the mean output of all the input vectors.

The smaller the value of $\mu_{\min}$, the bigger the contribution of the output fluctuations of f_1 to the final prototype allocation. Nevertheless, as $\mu_{\min}$ decreases, the time required by the algorithm increases dramatically while the approximation error of the model only suffers small changes. From Table I it can be concluded that values of $\mu_{\min}$ below 0.0001 do not significantly improve the approximation error, taking into account the computation time needed by CFA to reach convergence.

B. Comparison With Traditional Input Clustering Techniques

This experiment has been designed to show the importance of the output variability of the target function in the initialization of the approximator. Here, the CFA algorithm is compared with the traditional input clustering algorithms to make an initial model for the target function f_1 defined in (21).

Fig. 4 shows that function f_1 produces a very variable output when x is close to zero, becoming more stable as x is increased and achieving a "constant" value of zero when $x \geq 6$. A good initialization procedure should allocate more RBFs in the input interval where the output of f_1 is more variable than in the interval where f_1 takes the constant value of zero, thus increasing the variance explained by the model and reducing the unexplained variance or approximation error.

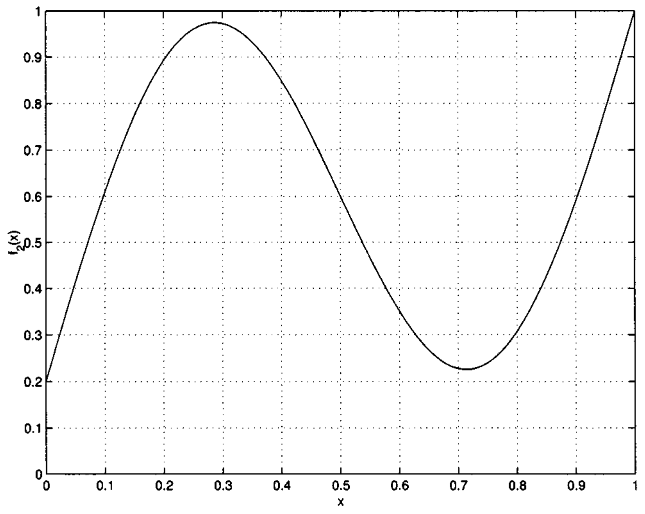


Fig. 6. Function f_2.

To test the effects caused by the CFA algorithm on the initialization of the prototypes, a training set of 1000 samples of f_1 was generated by evaluating inputs taken uniformly from the interval [0,10]. The clustering algorithms HCM, FCM with $r=2$, ELBG and CFA with $\mu_{\min}=0.001$ were applied to this training set several times. Our algorithm, CFA, obtained better approximation errors than traditional clustering algorithms for this function. Table II shows that as the number of prototypes increases, CFA is able to allocate them in the right positions to reduce the unexplained output variability of the target function.

TABLE IV
MEAN AND STANDARD DEVIATION OF THE APPROXIMATION ERROR JUST AFTER THE INITIALIZATION PROCEDURE TO INITIALIZE MODELS USING FROM FOUR TO TEN PROTOTYPES TO APPROXIMATE f_2 WITH THE ORIGINAL TRAINING DATA

c	NRMSE (dev)			
	Original training data		Perturbed training data	
	ACE (DC)	CFA	ACE (DC)	CFA
4	1.125 (0.045)	0.380 (0.035)	1.002 (0.023)	0.422 (0.062)
5	1.179 (0.020)	0.327 (0.078)	1.159 (0.026)	0.345 (0.052)
6	0.625 (0.018)	0.309 (0.089)	1.270 (0.120)	0.296 (0.087)
7	0.633 (0.090)	0.181 (0.051)	0.839 (0.357)	0.207 (0.025)
8	0.523 (0.072)	0.167 (0.048)	0.646 (0.219)	0.189 (0.043)
9	0.714 (0.295)	0.161 (0.039)	0.633 (0.321)	0.166 (0.036)
10	0.526 (0.096)	0.098 (0.024)	0.468 (0.071)	0.164 (0.027)

After obtaining the initial models for each clustering algorithm, the Levenberg–Marquardt method [7], [25] was applied to reduce their approximation error. Table III shows average NRMSEs and standard deviations for the executions of each clustering technique for RBFNNs with four to ten RBFs after applying the minimization algorithm. HCM and FCM algorithms obtain very variable results. This undesirable effect is produced because they perform a local search from an initial random configuration. ELBG and CFA avoid this kind of behavior by the incorporation of a prototype migration step that allows them to search globally for the best prototype allocation, but CFA also takes into account the target function variability to supervise the clustering algorithm, thus producing a better starting configuration for the minimization process.

It is apparent that even for unidimensional functions there exist a large number of local minima and that these are more and more numerous as the number of prototypes grows. This can be appreciated if we observe the deviations from the mean NRMSE in Table III. The initialization performed by traditional clustering techniques produces very variable configurations, so that the minimization procedure gets trapped in local minima and the standard deviations of the error become larger. Nevertheless, the small deviations from the mean NRMSE obtained by CFA show that the proposed algorithm converges toward the same initial configuration, thus obtaining very similar results after the minimization procedure.

C. CFA versus ACE

This example compares the performance of the CFA algorithm with another input–output clustering technique, the dancing cones (DC) instance of ACE proposed in [39]. First, we consider the approximation of the following function, also used in [39]:

$$f_2(x) = 0.2 + 0.8(x + 0.7\sin(2\pi x)), \qquad x \in [0, 1] \quad (23)$$

from 21 equidistant input–output training examples belonging to the interval [0, 1]. Fig. 6 shows a graph of the target function.

As in [39], we performed two experiments, the first one with the original training set and the second one with a modified training set to which a random 5% white noise was added. For each of these two experiments, the training data were approximated with models with four to ten prototypes (rules for DC or RBFs for CFA) and for each combination of training set and number of prototypes, the clustering algorithms were run several times. All the models obtained were tested with a set of 100 test points in the interval [0,1]. Table IV shows the mean and standard deviation of the test NRMSEs obtained.

TABLE V
MEAN AND STANDARD DEVIATION OF THE APPROXIMATION ERROR OBTAINED BY ACE (DC) TO INITIALIZE MODELS USING FUZZY SYSTEMS WITH 19 TO 27 RULES TO APPROXIMATE FUNCTION f_1

c	NRMSE (dev)
19	1.001 (0.011)
20	0.925 (0.008)
21	0.855 (0.010)
22	0.789 (0.008)
23	0.719 (0.008)
24	0.653 (0.017)
25	0.604 (0.025)
26	0.549 (0.027)
27	0.512 (0.011)

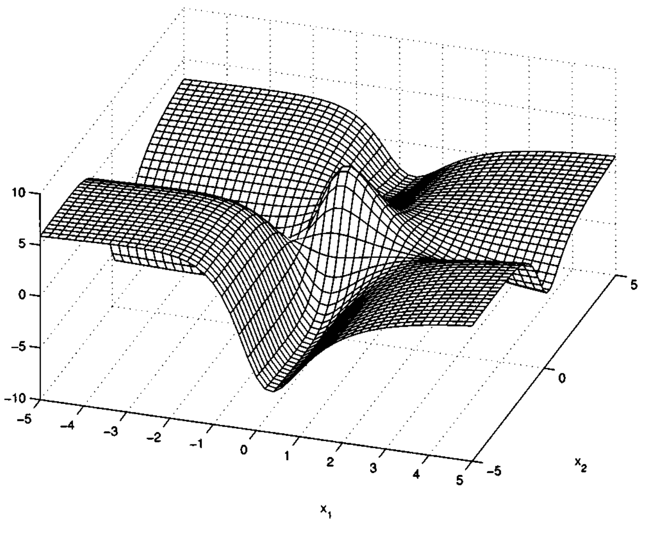


Fig. 7. Function f_3.

For the fuzzy models obtained with DC, the product was used as the T-norm, the sum as the T-conorm and the center of gravity (COG) as the defuzzification mechanism. This combination of operators has proved a good choice to solve function approximation problems elsewhere [18], [38].

Although CFA only performs changes in the prototypes, differing from DC, which also models the consequents of the fuzzy rules, the results obtained show that the proposed algorithm is

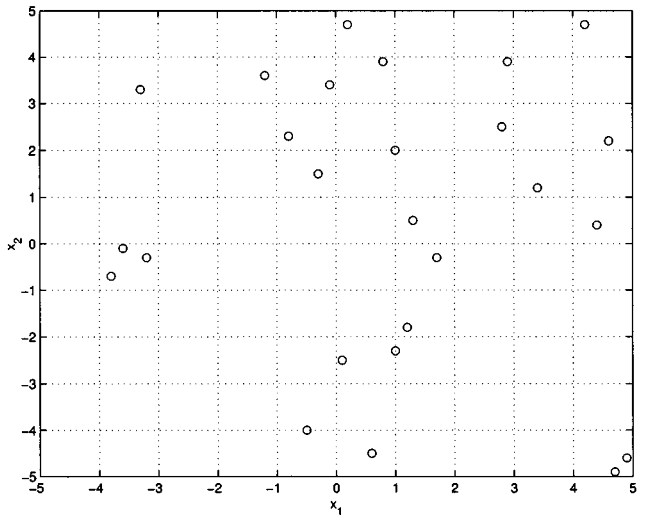


Fig. 8. Limited training set used to approximate f_3.

better fitted for function approximation problems. CFA converges to better models and obtains similar solutions in different executions of the algorithm. This fact is directly derived from the standard deviations of the NRMSE obtained by the two algorithms (see Table IV).

The second part of this section compares DC and CFA in the approximation of function f_1 (21). Comparing Table II and Table V, it can be seen that DC needs a considerably larger number of prototypes to obtain similar solutions to those obtained by CFA. This experiment shows again that CFA is able to fit target functions with a minimum number of prototypes.

D. CFA versus CFC

This last example compares the performance of the CFA algorithm with the CFC algorithm to obtain the initial RBF centers for an RBFNN. As in [32], a normalized RBFNN

$$F(\boldsymbol{x}) = \frac{\sum_{j=1}^{c} \omega_j \phi_j(\boldsymbol{x})}{\sum_{j=1}^{c} \phi_j(\boldsymbol{x})} \tag{24}$$

was used to approximate the two-input data produced by the target function

$$f_3(x_1, x_2) = \frac{(x_1-2)(2x_1+1)}{1+x_1^2} \cdot \frac{(x_2-2)(2x_2+1)}{1+x_2^2}, \quad x_1, x_2 \in [-5, 5] \tag{25}$$

as shown in Fig. 7.

The above function was approximated using two different training sets, a limited training set of 26 examples similar to that used in [32], shown in Fig. 8 and detailed in Table VI and a complete training set of 441 examples obtained from a grid of 21×21 points equidistributed in the input interval defined for f_3.

For CFA, both training sets were learned with RBFNNs with four to ten RBFs and for CFC, we used the same output context proposed in [32] and described by the following three trapezoidal linguistic labels:

$$\mathcal{A}_1(y) = \mathcal{T}(y; -8, -7, -3, -0.6) \tag{26}$$

TABLE VI
LIMITED TRAINING SET USED TO APPROXIMATE f_3

x_1	x_2	y
-3.8	-0.7	1.797
-3.6	-0.1	-4.137
-3.3	3.3	2.074
-3.2	-0.3	-2.109
-1.2	3.6	1.726
-0.8	2.3	0.274
-0.5	-4.0	0.000
-0.3	1.5	0.519
-0.1	3.4	-1.446
0.1	-2.5	-5.605
0.2	4.7	-2.947
0.6	-4.5	-5.542
0.8	3.9	-1.962
1.0	-2.3	-3.692
1.0	2.0	0.000
1.2	-1.8	-2.598
1.3	0.5	2.248
1.7	-0.3	0.286
2.8	2.5	0.247
2.9	3.9	0.671
3.4	1.2	-0.970
4.2	4.7	1.349
4.4	0.4	-2.868
4.6	2.2	0.221
4.7	-4.9	2.953
4.9	-4.6	3.058

TABLE VII
MEAN AND STANDARD DEVIATION OF THE APPROXIMATION ERROR JUST AFTER THE INITIALIZATION PROCEDURE TO APPROXIMATE f_3

Alg.	c	NRMSE (dev)	
		Limited data	Complete data
CFA	4	0.798 (0.017)	0.926 (0.008)
	5	0.797 (0.024)	0.898 (0.019)
	6	0.779 (0.007)	0.812 (0.044)
	7	0.752 (0.021)	0.758 (0.053)
	8	0.672 (0.032)	0.713 (0.016)
	9	0.593 (0.032)	0.708 (0.041)
	10	0.575 (0.016)	0.705 (0.035)
CFC	6	0.756 (0.017)	0.838 (0.073)
	9	0.709 (0.054)	0.812 (0.100)
	12	0.614 (0.021)	0.823 (0.056)

$$\mathcal{A}_2(y) = \mathcal{T}(y; -1, -0.4, 0.4, 1) \tag{27}$$

$$\mathcal{A}_3(y) = \mathcal{T}(y; 0.6, 3, 7, 8). \tag{28}$$

Each context was learned with two, three, and four prototypes, thus generating RBFNNs of six, nine, and 12 RBFs, respectively. As in the above examples, for each combination of network structure and training set, each clustering technique, CFA with $\mu_{\min} = 0.001$ and CFC with $r = 2$, was run several times to obtain several initial configurations.

Table VII shows average NRMSEs and standard deviations of the approximation error obtained. These results reveal that CFA also supervises the clustering and groups data according to their output similarity, but with one important difference with respect to CFC, that the contexts in the output space are obtained directly from the training data and so there is no need for an expert designer to define them *a priori*. This automatization means the algorithm obtains better results as it adjusts the output contexts dynamically.

V. Conclusion

When we have a set of input–output data from the observation of a system to be identified, determining the structure of the function approximator becomes an important issue.

This paper presents a new clustering algorithm specially designed for function approximation problems. It is shown that the proposed approach is specially useful in the approximation of functions with a high output variability, improving the performance obtained with traditional clustering algorithms that are oriented toward classification problems and which ignore this information when initializing the model.

The minimization methods currently used to fine tune nonlinear models perform a local search of the closest minimum to the starting configuration and so it is desirable for the initial configuration to be as close as possible to the global minimum. Incorporating some information about the target function into the clustering algorithm used to obtain the initial model has been proved to obtain better results than if it is constructed in an unsupervised way.

The proposed algorithm also improves the initialization performed by other input–output clustering techniques. This is directly derived from its objective function, which increases the density of prototypes in those input areas where the target function presents a more variable response, thus minimizing the variance of the output response of the training examples belonging to the same cluster.

It is emphasized that the proposed clustering method does not need any prior assumption about the structure of the data and that it is able to initialize function approximators from noise data.

Acknowledgment

The authors would like to thank the anonymous reviewers for their constructive comments and suggestions.

References

[1] A. Baraldi and P. Blonda, “A survey of fuzzy clustering algorithms for pattern recognition—Part II,” *IEEE Trans. Syst., Man, Cybern.*, pt. Part B, vol. 29, no. 6, pp. 786–801, December 1999.
[2] R. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton Univ. Press, 1957.
[3] J. C. Bezdek, *Pattern Recognition With Fuzzy Objective Function Algorithms* Plenum, New York, 1981.
[4] S. Chen, S. A. Billings, and W. Luo, “Orthogonal least squares methods and their application to nonlinear system identification,” *Int. J. Contr.*, vol. 50, no. 5, pp. 1873–1896, 1989.
[5] S. Chen, C. F. N. Cowan, and P. M. Grant, “Orthogonal least squares learning algorithm for radial basis function networks,” *IEEE Trans. Neural Networks*, vol. 2, pp. 302–309, 1991.
[6] M. Delgado, A. Skarmeta, and F. Martin, “Using fuzzy clustering in a descriptive fuzzy modeling approach,” in *Proc. 6th Int. Conf. IPMU’96*, vol. 1, Granada, Spain, 1996, pp. 563–568.
[7] J. E. Dennis and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Englewood Cliffs, NJ: Prentice-Hall, 1983.
[8] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.
[9] J. C. Dunn, “A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters,” *J. Cybern.*, vol. 3, pp. 32–57, 1973.
[10] A. Gersho, “Asymptotically optimal block quantization,” *IEEE Trans. Inform. Theory*, vol. 25, pp. 373–380, 1979.
[11] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Boston, MA: Kluwer, 1992.
[12] F. Glover, “Tabu search—Part I,” *ORSA J. Comput.*, vol. 1, pp. 190–206, 1989.
[13] ——, “Tabu search—Part II,” *ORSA J. Comput.*, vol. 2, pp. 4–32, 1990.
[14] J. A. Hartigan, *Clustering Algorithms*. New York: Wiley, 1975.
[15] S. Haykin, *Neural Networks, A comprehensive foundation*, 2nd ed. Upper Saddle River, NJ: Prentice-Hall, 1999.
[16] J. J. Holland, *Adaption in Natural and Artificial Systems*. Ann Arbor, MI: Univ. Michigan Press, 1975.
[17] *The State of the Art in Numerical Analysis*, D. A. H. Jacobs, Ed., Academic, London, U.K., 1977.
[18] J. S. R. Jang, C. T. Sun, and E. Mizutani, *Neuro-Fuzzy and Soft Computing*. Upper Saddle River, NJ: Prentice-Hall, 1997.
[19] P. P. Kanjilal and D. N. Banerjee, “On the application of orthogonal transformation for the design and analysis of feedforward networks,” *IEEE Trans. Neural Networks*, vol. 6, pp. 1061–1070, 1995.
[20] N. B. Karayiannis and G. W. Mi, “Growing radial basis neural networks: Merging supervised and unsupervised learning with network growth techniques,” *IEEE Trans. Neural Networks*, vol. 8, pp. 1492–1506, Nov. 1997.
[21] S. Kirpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by simulated annealing,” *Science*, vol. 220, pp. 671–680, May 1983.
[22] A. Leonardis and H. Bischof, “An efficient MDL-based construction of RBF networks,” *Neural Networks*, vol. 11, pp. 963–973, 1998.
[23] C. Linnaeus, *Clavis Classium in Systemate Phytologorum in Bibliotheca Botanica*. Amsterdam, The Netherlands: Biblioteca Botanica, 1736.
[24] E. H. Mamdani and S. Assilian, “An experiment in linguistic synthesis with a fuzzy logic controller,” *Int. J. Man-Machine Studies*, vol. 7, no. 1, pp. 1–13, 1975.
[25] D. W. Marquardt, “An algorithm for least-squares estimation of nonlinear inequalities,” *SIAM J. Appl. Math.*, vol. 11, pp. 431–441, 1963.
[26] J. Moody and C. J. Darken, “Fast learning in networks of locally-tuned processing units,” *Neural Comput.*, vol. 1, no. 2, pp. 281–294, 1989.
[27] M. T. Musavi, W. Ahmed, K. H. Chan, K. B. Faris, and D. M. Hummels, “On the training of radial basis function classifiers,” *Neural Networks*, vol. 5, no. 4, pp. 595–603, 1992.
[28] J. Park and I. W. Sandberg, “Universal approximation using radial-basis-function networks,” *Neural Comput.*, vol. 3, pp. 257–546, 1991.
[29] ——, “Approximation and radial-basis-function networks,” *Neural Comput.*, vol. 5, pp. 305–316, 1993.
[30] W. Pedrycz, *Fuzzy Contr. Fuzzy Syst.*. New York: Wiley, 1993.
[31] ——, “Conditional fuzzy C-means,” *Pattern Recognition Lett.*, vol. 17, pp. 625–632, 1996.
[32] ——, “Conditional fuzzy clustering in the design of radial basis function neural networks,” *IEEE Trans. Neural Networks*, vol. 9, pp. 601–612, 1998.
[33] T. Poggio and F. Girosi, “A Theory of Networks for Approximation and Learning,” MIT Artificial Intelligence Laboratory, Cambridge, MA, Tech. Rep. AI-1140, 1989.
[34] E. Polak, *Computational Methods in Optimization*. New York: Academic, 1971.
[35] H. Pomares, I. Rojas, J. Ortega, J. González, and A. Prieto, “A systematic approach to a self-generating fuzzy rule-table for function approximation,” *IEEE Trans. Syst., Man, Cybern. B*, vol. 30, pp. 431–447, June 2000.
[36] I. Rojas, H. Pomares, J. González, J. L. Bernier, E. Ros, F. J. Pelayo, and A. Prieto, “Analysis of the functional block involved in the design of radial basis function networks,” *Neural Processing Lett.*, vol. 12, no. 1, pp. 1–17, Aug. 2000.
[37] I. Rojas, H. Pomares, J. Ortega, and A. Prieto, “Self-organized fuzzy system generation from training examples,” *IEEE Trans. Fuzzy Syst.*, vol. 8, pp. 23–36, Feb. 2000.
[38] I. Rojas, O. Valenzuela, M. Anguita, and A. Prieto, “Analysis of the operators involved in the definition of the implication functions and in the fuzzy inference process,” *Int. J. Approximate Reasoning*, vol. 19, pp. 367–389, 1998.
[39] T. A. Runkler and J. C. Bezdek, “Alternating cluster estimation: A new tool for clustering and function approximation,” *IEEE Trans. Fuzzy Syst.*, vol. 7, pp. 377–393, Aug. 1999.

[40] E. Ruspini, “A new approach to clustering,” *Inf. Contr.*, vol. 15, pp. 22–32, 1969.

[41] M. Russo, “FuGeNeSys—A fuzzy genetic neural system for fuzzy modeling,” *IEEE Trans. Fuzzy Syst.*, vol. 6, pp. 373–388, Aug. 1998.

[42] M. Russo and G. Patanè, “Improving the LBG Algorithm,” in *Lecture Notes in Computer Science*. New York: Springer-Verlag, 1999, vol. 1606, pp. 621–630.

[43] J. Rust, “Using randomization to break the curse of dimensionality,” *Econometrica*, vol. 65, no. 3, pp. 487–516, 1997.

[44] E. L. Sutanto, J. D. Masson, and K. Warwick, “Mean-tracking clustering algorithm for radial basis function center selection,” *Int. J. Contr.*, vol. 67, no. 6, pp. 961–977, 1997.

[45] R. C. Tryon, *Cluster Analysis*. Ann Arbor, MI: Edward Brothers, 1939.

[46] L. A. Zadeh, “Fuzzy sets,” *Inform. Contr.*, vol. 8, pp. 338–353, 1965.

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

Jesús González was born in 1974. He received the M.Sc. degree in computer science in 1997 and the Ph.D. degree in 2001, and from the University of Granada, Spain. He is currently an Assistant Professor at the same University.

His current areas of research interest include function approximation using radial basis function neural networks, fuzzy systems, and evolutionary computation.

Ignacio Rojas received the M.Sc. degree in physics and electronics in 1992 and the Ph.D. degree in 1996, both from the University of Granada, Spain.

He was at the University of Dortmund, Germany, as Invited Researcher from 1993 to 1995. In 1998, he was Visiting Researcher at the BISC Group, University of California, Berkeley. He is currently an Associate Professor in the Department of Computer Architecture and Computer Technology at the University of Granada, Spain. His research interests include hybrid system and combination of fuzzy logic, genetic algorithms and neural networks and financial forecasting.

Héctor Pomares was born in 1972. He received the M.Sc. degree in electronic engineering in 1995, the M.Sc. degree in physics in 1997, and the Ph.D. degree in 2000, all from the University of Granada, Granada, Spain.

He is currently an Associate Professor in the Department of Computer Architecture and Computer Technology at the same university. His current areas of research interest include function approximation and on-line control using adaptive and self-organizing fuzzy systems.

Julio Ortega (A’88–M’98) received the B.Sc. degree in electronic physics in 1985, the M.Sc. degree in electronics in 1986, and the Ph.D. degree in 1990, all form the University of Granada, Granada, Spain.

He was at the Open University, U.K. and at the Department of Electronics, University of Dortmund, Germany, as Invited Researcher. Currently, he is an Associate Professor in the Department of Computer Architecture and Computer Technology at the University of Granada. His research interests include parallel processing and parallel computer architectures, artificial neural networks and evolutionary computation. He has led research projects in the area of parallel algorithms and architectures for combinatorial optimization problems.

Dr. Ortega’s Ph.D. dissertation received the Ph.D. Award of the University of Granada.

Alberto Prieto (M’74) received the B.Sc. degree in electronic physics in 1968 from the Complutense University, Madrid, Spain, and the Ph.D. degree from the University of Granada, Granada, Spain, in 1976.

From 1969 to 1970, he was at the “Centro de Investigaciones Técnicas de Guipuzcoa” and the “E.T.S.I. Industriales” of San Sebastián, Spain. From 1971 to 1984, he was Director of the Computer Center and from 1985 to 1990 Dean of the Computer Science and Technology studies at the University of Granada, Spain, where he is currently Full Professor and Director of the Department of Computer Architecture and Computer Technology. He has stayed as Visitor Researcher in different foreign centers as the University of Rennes, France (1995, Prof. Boulaye), the Open University, U.K., (1987 Prof. S. L. Hurst), the Institute National Polytechnique of Grenoble, France (1991, Profs. J. Herault and C. Jutten) and the University College of London, London, U.K., (1991–1992, Prof. P. Treleaven). His research interests include intelligent systems.

Prof. Prieto received the Award of Ph.D. dissertations and the Citema Foundation National Award. He was the Organization Chairman of the International Workshop on Artificial and Neural Networks (IWANN’91) (Granada, Spain, September 17–19, 1991) and the Seventh International Conference in Microelectronics for Neural, Fuzzy and Bio-inspired Systems (MicroNeuro’99), Granada, Spain, April 7–9, 1999. He is nominated member of the IFIP WG 10.6 (Neural Computer Systems) and Chairman of the Spanish RIG of the IEEE Neural Networks Council.