Journal of Intelligent & Fuzzy Systems 11 (2001) 147–161
IOS Press

Structure and parameter learning of neuro-fuzzy systems: A methodology and a comparative study

Rui Pedro Paiva and António Dourado*
CISUC – Centro de Informática e Sistemas da Universidade de Coimbra, Department of Informatics Engineering, Pólo II of the University of Coimbra, P 3030 Coimbra, Portugal

Abstract. A methodology and experimental comparison of neuro-fuzzy structures, namely linguistic and zero and first-order Takagi-Sugeno, are developed. The implementation of the model is conducted through the training of a neuro-fuzzy network, i.e., a neural net architecture capable of representing a fuzzy system. In the first phase, the structure of the model is obtained by subtractive clustering, which allows the extraction of a set of relevant rules based on a set of representative input-output data samples. Membership functions with two-sided Gaussian functions are proposed and discussed. In the second phase, the model parameters are tuned via the training of a neural network. Furthermore, different fuzzy operators are compared, as well as regular and two-sided Gaussian functions.

Keywords: Neuro-fuzzy systems, structure learning, parameter learning, clustering

1. Introduction

The construction of fuzzy systems for process modeling and prediction or, in general, for feature extraction from data is presently carried out in several ways. From the set of possible strategies, performing their training through neural networks seems to be the most promising one. This strategy leads to so-called neuro-fuzzy systems. The development of neuro-fuzzy systems is actually a subject of great activity. In fact, since the pioneer works of Zadeh [16] and Mamdani [10] many progresses have been made regarding the development of fuzzy relations based on experiences of skilled operators or from a set of observed input-output data [5]. With the development of neural networks and of its training algorithms, their computational potentialities have been introduced into fuzzy learning operations leading to neuro-fuzzy systems. Many structures have been proposed and some were largely disseminated, among which Jang's ANFIS structure [6] is probably the most famous. Several recent developments have improved the possibilities of neuro-fuzzy systems. For example Azeem, Hanmandlu and Ahmad [1] proposed the GANFIS structure, a generalization of Jang's work. Hong and Harris [4] researched for new types of basis function for the case of n-dimensional input spaces in order to overcome the curse of dimensionality problem when n is large. Zhang and Morris [17] proposed a type of recurrent neuro-fuzzy networks to build long-range prediction models for nonlinear processes easier to interpret than strictly black-box models. Pal and Pal [12] propose a connectionist implementation of the compositional rule of inference and an architecture for it that automatically finds an "optimal" relation representing a set of fuzzy rules. Kim and Kasabov [7] propose an HyFIS- Hybrid Neural Fuzzy Inference System for building and optimizing fuzzy models providing linguistic meaning; they use a two-phase methodology: a rule generation phase from data and a rule tuning phase using error backpropagation.

In this work, it is presented a methodology carried out in two main phases: in the first one, structure learn-

*Correspondent author. E-mail: dourado@dei.uc.pt; URL: http://www.dei.uc.pt/~dourado.

ing is performed, i.e., a set of fuzzy rules is obtained by subtractive clustering; in the second one, the model parameters (the membership function parameters of the fuzzy system) are tuned.

Based on the generic methodology referred, an analysis is made regarding some important issues in fuzzy modeling, e.g., type of rules (Takagi – Sugeno [15] or linguistic), type of operators and membership functions. In terms of membership functions, this study is restricted to simple and two-sided Gaussian functions.

The paper is organized as follows. In Section 2 the main issues of fuzzy identification are introduced. In Section 3 subtractive clustering, used for structure learning, is presented. Then, the parameter learning strategies used are described in Section 4. The methodologies are applied to the Mackey-Glass time series, in Section 5. Finally, some conclusions are drawn in Section 6.

2. Fuzzy modelling and identification

Dynamical system identification deals with the implementation of models using experimental data. Thus, when a model is developed based on the theory of system identification, its parameters are tuned according to some criteria, aiming to obtain a final representation adequate for the modeling purposes. In this sense, fuzzy identification is presented as a particular case of system identification, in which the model is categorized as a fuzzy system.

Thus, without loss of generality, let us assume a single-input single-output (SISO) model, with one input, u, and one output, y, from where N data samples are collected Eq. (1):

$$Z^N = \{u(1), y(1)], [u(2), y(2)], \ldots, [u(N), y(N)]\} \tag{1}$$

Using data collected from the system, the goal is to obtain a fuzzy model, represented by a set of rules of type R_i Eq. (2):

$$\begin{aligned} R_i:\ & \text{If } y(t-1) \text{ and } y(t-2) \text{ is } A_{2i} \text{ and } \ldots \\ & \text{is } A_{1i} \text{ and } u(t-d) \\ & \text{is } B_{1i} \text{ and } u(t-d-1) \text{ is } B_{2i} \text{ and } \ldots \\ & \text{then } \hat{y}(t) \text{ is } C_{1i} \end{aligned} \tag{2}$$

where d represents the system time delay and A_{ji}, B_{ji} and C_{ji} denote linguistic terms associated to each input and output. Those terms are defined by their respective membership functions $\mu_{A_{ji}}, \mu_{B_{ji}}, \mu_{C_{ji}}$. The previous structure is called a FARX structure (Fuzzy Auto Regressive with eXogenous inputs), as a generalization of the well-known ARX structure. Thus, the selection of a set of rules of type Eq. (2), as well as the definition of the fuzzy sets A_{ji}, B_{ji} and C_{ji}, constitute project issues specific to fuzzy systems.

3. Structure learning

In order to obtain a set of g fuzzy conditional rules, capable of representing the system under study, clustering algorithms are particularly suited, since they permit a scatter partitioning of the input-output data space, which results in finding only the relevant rules. Comparing to grid-based partitioning methods, clustering algorithms have the advantage of avoiding the rule base explosion, i.e., the curse of dimensionality. Some authors use grid-based partitioning methods, combined with network pruning. Based on the authors' previous work [13], it is their opinion that the results are not as good as the ones resulting from clustering techniques: curse of dimensionality problems are avoided; the pruning of the network can lead to wrong deletion of nodes if the network is not optimized; optimization of a large dimension network is very time consuming; after deletion of nodes, the network should be re-optimized.

In this paper, Chiu's subtractive clustering is applied [2]. This scheme possesses some interesting advantages, especially in a neuro-fuzzy identification context. In fact, subtractive clustering is an efficient algorithm and does not require any optimization, being a good choice for the initialization of the neuro-fuzzy network. Fuzzy c-means and other optimization-based clustering algorithms would lead to a performance diminishing because they perform an unnecessary optimization phase prior to network training. Also, progressive clustering and compatible cluster merging algorithms are computationally expensive and need metrics for validation of individual clusters [3]. Therefore, despite their potential, they are too complex for a simple initialization of a fuzzy neural network.

Chiu's algorithm belongs to the class of potential function methods, being, more precisely, a variation of the mountain method (see [3]). In this class of algorithms, a set of points are defined as possible group centers, each of them being interpreted as an energy source. In subtractive clustering the center candidates are the data samples themselves. In this way, the main limitation of the mountain method is overtaken. In fact, there, the candidates are defined in a grid, leading to curse of dimensionality problems.

So, let Z^N Eq. (1) be a set of N data samples, $z_1, z_2, \ldots, z_N$, defined in a $m+n$ space, where m denotes the number of inputs and n the number of outputs. In order to make the range of values in each dimension identical, the data samples are normalized, so that they are limited by a hypercube.

As it was referred, it is admitted that each of the samples defines a possible cluster center. Therefore, the potential associated to z_i is Eq. (3):

$$P_i(z_i, Z^N) = \sum_{j=1}^{N} e^{-\alpha \|z_i - z_j\|^2}, \qquad \alpha = \frac{4}{r_a^2},\ i = 1, 2, \ldots, N \tag{3}$$

where $r_a > 0$ is *radii*, a constant which defines the neighborhood radius of each point. Thus, points z_j located out of the radius of z_i will have a reduced influence in its potential. On the other hand, the effect of points close to z_i will grow with their proximity. In this way, points with a dense neighborhood will have higher associated potentials.

After computing the potential for each point, the one with the highest potential is selected as the first cluster center.

Next, the potential of all the remaining points is reduced. Defining z_1^* as the first group center and denoting its potential as P_1^*, the potential of the remaining points is reduced as in Eq. (4):

$$P_i \leftarrow P_i - P_1^* e^{-\beta \|z_i - z_1^*\|^2},\ \beta = \frac{4}{r_b^2} \tag{4}$$

where the constant $r_b > 0$ defines the neighborhood radius with sensible reductions in its potential.

In this way, points close to the center selected will have their potentials reduced in a more significant manner, and so the probability of being chosen as centers diminishes. This procedure has the advantage of avoiding the concentration of identical clusters in denser zones. Therefore, the r_b value is selected in order be slightly higher than r_a, so as to avoid closely spaced clusters. Typically, $r_b = 1.5 r_a$.

After performing potential reduction for all the candidates, the one with the highest potential is selected as the second cluster, after what the potential of the remaining points is again reduced. Generically, after determining the rth group, the potential is reduced as Eq. (5):

$$P_i \leftarrow P_i - P_r^* e^{-\beta \|z_i - z_r^*\|^2} \tag{5}$$

The procedure of center selection and potential reduction is repeated until the following stopping criterion (Algorithm 1) is reached.

Algorithm 1. *Stopping criterion for subtractive clustering.*

```
If $P_k^* > \varepsilon^{\text{up}} P_1^*$
  Accept $z_k^*$ as the next cluster center and continue
Otherwise,
  If $P_k^* < \varepsilon^{\text{down}} P_1^*$
    Reject $z_k^*$ and finish the algorithm.
  Otherwise
    Let $d_{\min}$ be the shortest distance between $z_k^*$ and
    all the centers already found
    If $d_{\min}/r_a + P_k^*/P_1^* \geqslant 1$
      Accept $z_k^*$ as the next cluster center and continue
    Otherwise
      Reject $z_k^*$ and assign it the potential 0.0.
      Select the point with higher potential as new $z_k^*$.
      Repeat the test.
    End If
  End If
End If
```

In Algorithm 1, ε^{up} specifies a threshold above which the point is selected as a center with no doubts and $\varepsilon^{\text{down}}$ specifies the threshold below which the point is definitely rejected. The third case is where the point is characterized by a good trade-off between having a sufficiently high potential and being distant enough from the clusters determined before. Typically, $\varepsilon^{\text{up}} = 0.5$ and $\varepsilon^{\text{down}} = 0.15$.

As it can be understood from the description of the algorithm, the number of clusters to obtain is not prespecified. However, it is important to note that the parameter *radii* is directly related to the number of clusters found. Thus, a small radius will lead to a high number of rules, which, if excessive, may result in overfitting problems. On the other hand, a bigger radius will lead to a smaller number of clusters, which may originate underfitting, and so, models with reduced representation accuracy. Therefore, in practice it is necessary to test several values for *radii* and select the most adequate according to the results obtained. However, despite the fact that some *radii* values should be tested, this parameter gives an initial hint on the number of clusters necessary [13]. This constitutes an important advantage over optimization-based and other classes of clustering algorithms, when little information is known regarding the best number of clusters. Another advantage of subtractive clustering is that the algorithm is noise robust, since outliers do not significantly influence the choice of centers, due to their low potential.

After applying subtractive clustering, each of the clusters obtained will constitute a prototype for a particular behavior of the system under analysis. So, each cluster can be used to define a fuzzy rule, able to describe the behavior of the system in some region of the input-output space. Typically, g fuzzy conditional rules of type Eq. (6) are obtained:

$$\begin{aligned}&\text{Rule } r:\\&\text{IF } (X_1 \text{ is } LX1^{(r)}) \text{ AND } (X_2 \text{ is } LX2^{(r)})\\&\quad\text{AND } \ldots \text{ AND } (X_m \text{ is } LXm^{(r)})\\&\text{THEN } (Y_1 \text{ is } LY1^{(r)}) \text{ AND } (Y_2 \text{ is } LY2^{(r)})\\&\quad\text{AND } \ldots \text{ AND } (Y_n \text{ is } LY_n^{(r)})\end{aligned} \tag{6}$$

where each of the linguistic terms in the antecedent, $LXj^{(r)}$, has an associated membership function defined as follows Eq. (7):

$$\begin{aligned}&\mu_{LXj^{(r)}}(x_j) = e^{-\alpha(x_j - x_{rj}^*)^2},\\&r = 1, 2, \ldots, g;\ j = 1, 2, \ldots, m\end{aligned} \tag{7}$$

Here, x_j denotes a numeric value related to the jth input dimension and x_{rj}^* is the jth coordinate in the m-dimensional vector x_r^*. Equation (7) results from the computation of the potential associated to each point in the data space. Clearly, expression Eq. (6) is a consequence of using linguistic models, i.e., models in which the consequents are fuzzy sets. Such consequents result naturally from the application of subtractive clustering and are obtained as follows Eq. (8):

$$\begin{aligned}&\mu_{LYo^{(r)}}(y_o) = e^{-\alpha(y_o - y_{ro}^*)^2},\\&\qquad\qquad o = 1, 2, \ldots, n\end{aligned} \tag{8}$$

where y_o denotes a numeric value regarding the oth output dimension and y_{ro}^* is the jth coordinate in the n-dimensional vector y_r^*.

Obtaining an initial structure for Takagi-Sugeno models, in which the terms in the consequents are typically zero and first-order linear functions, is performed similarly. However, since the consequents are not fuzzy sets, the initialization procedure just described applies only to the antecedents. In fact, based on the linear characteristics of the consequents, their values can be easily obtained by means of linear optimization techniques.

Comparing Eqs (7), (8) and the general equation for Gaussian functions, it becomes clear that the membership functions considered belong to the type referred. Thus, regarding the standard deviation of each function, expression Eq. (9) is obtained:

$$\sigma_{rj} = \frac{r_a}{\sqrt{8}} \tag{9}$$

Finally, after the parameterization of the Gaussian membership functions, the data used, normalized previously, are restored to their initial values. In the same way, the function parameters are adjusted to the domains defined for each dimension.

4. Parameters learning

After determining a fuzzy model structure, the model parameters, i.e., the centers and standard deviations of the Gaussian membership functions, should be tuned. Therefore, it is necessary to select the type of model to use. In linguistic models, the conditional rules belong to type Eq. (6). Regarding zero-order Takagi-Sugeno models, with constant consequents, the rules are of type Eq. (10). As for first order Takagi-Sugeno models, the rules are of type Eq. (11), where $f_{or}(x)$ is defined as in Eq. (12).

$$\begin{aligned}&\text{Rule } r:\\&\text{IF } (X_1 \text{ is } LX1^{(r)}) \text{ AND } (X_2 \text{ is } LX2^{(r)})\\&\text{AND } \ldots \text{ AND } (X_m \text{ is } LXm^{(r)})\\&\text{THEN } (\hat{y}_1 = b_{1r}) \text{ AND } (\hat{y}_2 = b_{2r})\\&\text{AND } \ldots \text{ AND } (\hat{y}_n = b_{nr})\end{aligned} \tag{10}$$

$$\begin{aligned}&\text{Rule } r:\\&\text{IF } (X_1 \text{ is } LX1^{(r)}) \text{ AND } (X_2 \text{ is } LX2^{(r)})\\&\text{AND } \ldots \text{ AND } (X_m \text{ is } LXm^{(r)})\\&\text{THEN } (\hat{y}_1 = f_{1r}) \text{ AND } [\hat{y}_2 = f_{2r}(x)]\\&\text{AND } \ldots \text{ AND } [\hat{y}_n = f_{nr}(x)]\end{aligned} \tag{11}$$

$$\begin{aligned}f_{or}(x) &= b_{or0} + b_{or1}x_1 + b_{or2}x_2 + b_{orm}x_n\\&\quad + b_{orj} \in \mathcal{R},\ j = 1, 2, \ldots, m;\\&\quad o = 1, 2, \ldots, n;\ r = 1, 2 \ldots, g\end{aligned} \tag{12}$$

As it can be seen, first order Takagi-Sugeno models are characterized by having more flexible consequents. Thus, these structures can be seen as smooth shifters between local linear models, which is an advantage comparing to the interpolative properties of both linguistic and zero-order models.

The parameters of each of the referred fuzzy structures are adjusted by means of a fuzzy neural network, i.e., a neural network able to represent the functions of a fuzzy system, namely, fuzzification, fuzzy implication and defuzzification.

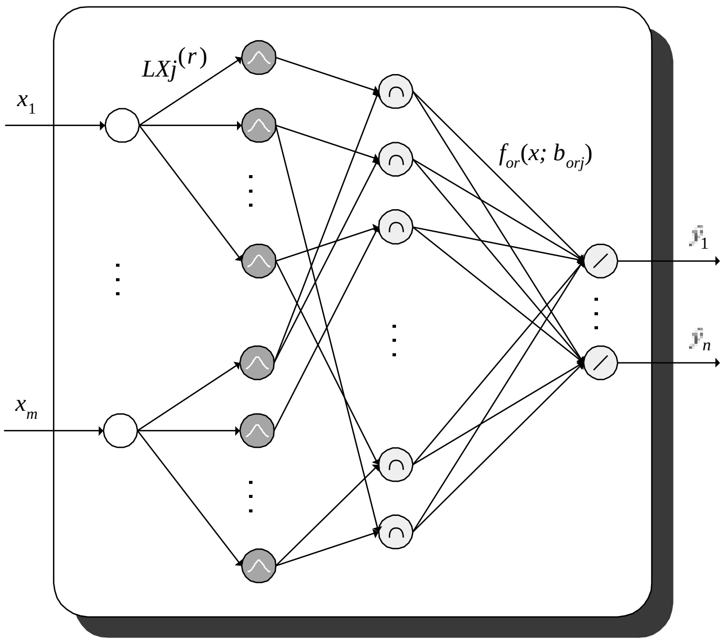

Fig. 1. Neuro-fuzzy network: Takagi-Sugeno type consequents.

4.1. Neuro-fuzzy arquitectures

Basically, the nets presented in the following paragraphs are composed by an input layer, preceding a fuzzification layer and then a rule layer. After the initial layers, the next layer is, in the case of Takagi-Sugeno models, the final linear output layer. As for fuzzy consequents, the fourth layer is the union layer, used to integrate rules with the same consequents, and the last one is the output layer, responsible for defuzzification.

In order to make the next expressions more readable, the notation used is presented beforehand:

- m: number of network inputs
- n: number of network outputs
- g: number of fuzzy rules (groups)
- $a_i^{(p2)}$: activation of neuron i in layer 2, regarding training pattern p (i denotes an input term: "input");
- $a_r^{(p3)}$: activation of neuron r in layer 3, regarding pattern p (r denotes "rule");
- $a_s^{(p4)}$: activation of neuron s in layer 4, regarding pattern p (s denotes "S-norm");
- $a_o^{(p5)} = \hat{y}_o^{(p)}$: activation of neuron o in layer 5, i.e., output, regarding pattern p (o denotes "output");
- $y_o^{(p)}$: desired activation for neuron o in layer 5, i.e., for the network output, regarding pattern p; this is an output sample.

As for Takagi-Sugeno models, the output takes place in the fourth layer, resulting:

- $a_o^{(p4)} = \hat{y}_o^{(p)}$: activation of neuron o in layer 5, i.e., output, regarding pattern p (o denotes "output");
- $y_o^{(p)}$: desired activation for neuron o in layer 5, i.e., for the network output, regarding pattern p.

Takagi-Sugeno models

From the described previously, Takagi-Sugeno structures are represented by the architecture in Fig. 1. Naturally, the network presented serves both first and zero-order models, in which the consequents will be either first order functions or constants, respectively.

In this structure, the input layer simply receives data from the external environment and passes them to the next layer.

In the second layer, the fuzzification layer, each of the cells corresponds to a membership function associated to each of the inputs. Defining conventional Gaussian functions, the output of each neuron in this layer is given by Eq. (13):

$$a_i^{(p2)} = e^{-\frac{\left(x_j^{(p)} - c_{ij}\right)^2}{2\sigma_{ij}^2}} \tag{13}$$

where c_{ij} and σ_{ij} represent, respectively, the center and standard deviation of the ith membership function related to the jth input. Such parameters constitute the weights of layer one to layer two links ($LXj^{(r)}$ in

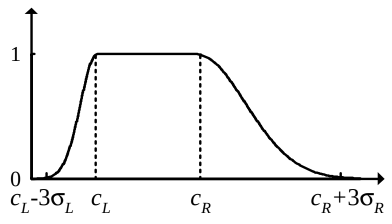

Fig. 2. Two-sided Gaussian function.

Fig. 1). In the same expression, $x_j^{(p)}$ denotes the pth pattern associated do input j.

Alternatively, it is possible to define two-sided Gaussian functions, which are characterized by their possibility of being asymmetric and containing a plateau, as a generalization of conventional functions (Fig. 2). Therefore, the possibility of obtaining better results can be formulated, due to the increased flexibility of the generalized functions.

In the case where two-sided Gaussians are used, the activation of each of the neurons in this layer is given by Eq. (14).

$$a_i^{(p2)} = \begin{cases} e^{-\frac{\left(x_j^{(p)} - c_{ijL}\right)^2}{2\sigma_{ijL}^2}}, \; x_j^{(p)} < c_{ijL} \\ 1, \; c_{ijL} \leqslant x_j^{(p)} \leqslant c_{ijR} \\ e^{-\frac{\left(x_j^{(p)} - c_{ijR}\right)^2}{2\sigma_{ijR}^2}}, \; x_j^{(p)} < c_{ijR} \end{cases} \tag{14}$$

Here, c_{ijL} and σ_{ijL} represent, respectively, the center and standard deviation of the left component of the ith membership function related to the jth input. For the right component, the index R is used. Such parameters constitute the weights of layer one to layer two links ($LXj^{(r)}$ in Fig. 1). In the same expression, $x_j^{(p)}$ denotes the pth pattern associated to input j.

As for the neurons in the rule layer, their function consists of performing the antecedent conjunction of each rule, by means of some T-norm, e.g., product Eq. (15) or minimum Eq. (16). The first one is classified as an algebraic operator and the second one is a truncation operator.

$$a_r^{(p3)} = T - \text{norm}_{i=1}^{na_r}\left(a_i^{(p2)}\right) = \prod_{i=1}^{na_r} a_i^{(p2)} \tag{15}$$

$$\begin{aligned} a_r^{(p3)} &= T - \text{norm}_{i=1}^{na_r}\left(a_i^{(p2)}\right) \\ &= \min_{i=1}^{na_r}\left(a_i^{(p2)}\right) \end{aligned} \tag{16}$$

In the previous expressions, na_r stands for the number of inputs in the antecedent of rule r.

Regarding the output layer, its task consists of computing numeric outputs based on the level of activation of each rule. As referred previously, in zero-order models the weights in this layer denote the rule consequents, defined by constants. Therefore, each output neuron is activated as in Eq. (17). In the implementation of first order Takagi-Sugeno models, the net defines a fuzzy system with rules of type Eq. (11). In this way, the task of the output neurons is very similar to the ones in zero-order models, being defined as in Eq. (18).

$$\hat{y}_o^{(p)} = a_o^{(p4)} = \frac{\sum_{r=1}^{g} a_r^{(p3)} \cdot b_{0r}}{\sum_{r=1}^{g} a_r^{(p3)}} \tag{17}$$

$$\begin{aligned} \hat{y}_o^{(p)} &= a_o^{(p4)} \\ &= \frac{\sum_{r=1}^{g} a_r^{(p3)} \cdot \left(\sum_{j=1}^{m} b_{orj} x_j^{(p)} + b_{or0}\right)}{\sum_{r=1}^{g} a_r^{(p3)}} \end{aligned} \tag{18}$$

Fuzzy consequents models

As a basis for dealing with fuzzy consequents, Lin defines in his NFCN architecture [8] a fuzzy neural net composed by five layers, as in Fig. 3. However, the original structure is adapted in the present work, so as to allow different operators and membership functions.

Comparatively to the network depicted in Fig. 1, the first three layers perform exactly the same tasks. Obviously, the difference resides in the layers following those.

Thus, the fourth layer, called the union layer, is responsible for integrating the rules with the same consequents, via some S-norm, e.g., bounded sum Eq. (19) or maximum Eq. (20). The first one is classified as an algebraic operator and the second one is a truncation operator. There, nr_s stands for the number of rules which have neuron s as consequent.

$$\begin{aligned} a_s^{(p4)} &= S - \text{norm}_{r=1}^{nr_s}\left(a_r^{(p3)}\right) \\ &= \min\left(1, \sum_{r=1}^{nr_s} a_r^{(p3)}\right) \end{aligned} \tag{19}$$

$$\begin{aligned} a_s^{(p4)} &= S - \text{norm}_{r=1}^{nr_s}\left(a_r^{(p3)}\right) \\ &= \max_{r=1}^{nr_s}(a_r^{(p3)}) \end{aligned} \tag{20}$$

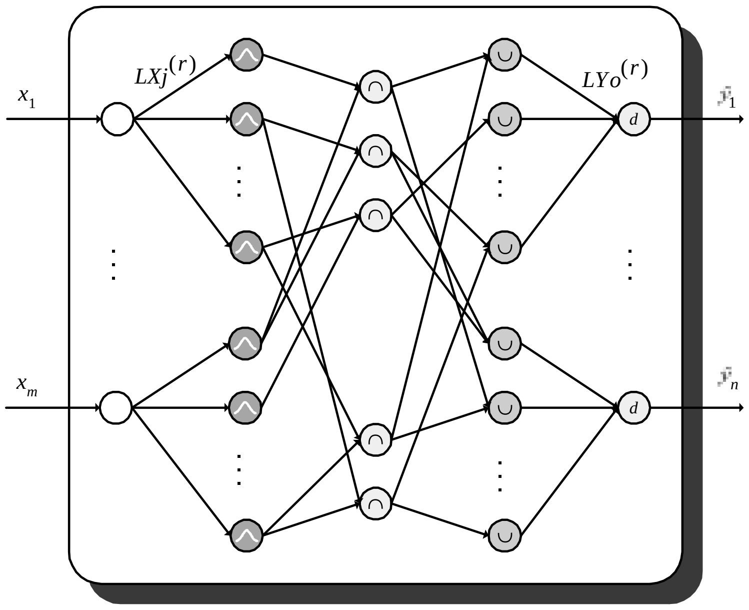

Fig. 3. Neuro-fuzzy network: fuzzy consequents.

As for the output layer, or defuzzification layer (d, in Fig. 3), layer four to layer five links ($LYo^{(r)}$ in the same figure) define the parameters of the membership functions associated to the output linguistic terms. Thus, based on these membership functions and on the activation of each rule, its neurons should implement a defuzzification method suited to the types of fuzzy consequents, as the one presented in [8] Eq. (21):

$$\hat{y}_o^{(p)} = a_o^{(p5)} = \frac{\sum_{s=1}^{|T(Y_o)|} c_{os}\sigma_{os}a_s^{(p4)}}{\sum_{s=1}^{|T(Y_o)|} \sigma_{os}a_s^{(p4)}} \tag{21}$$

In Eq. (21), c_{os} and σ_{os} represent the center and standard deviation of the sth membership function related to output o. In the case where two-sided Gaussians are used, Eq. (22) results, as is defined in [14].

$$\hat{y}_o^{(p)} = a_o^{(p5)} \tag{22}$$
$$= \frac{\sum_{s=1}^{|T(Y_o)|} \frac{1}{2}(c_{osL}\sigma_{osL} + c_{osR}\sigma_{osR})a_s^{(p4)}}{\sum_{s=1}^{|T(Y_o)|} \frac{1}{2}(\sigma_{osL} + \sigma_{osR})a_s^{(p4)}}$$

In the previous expressions, $|T(Y_o)|$ stands for the number of membership functions associated to each linguistic output variable Y_o. The main idea of the defuzzification method proposed is to weight the activation of each rule, not only by the centers, right and left, but also by their standard deviations. Clearly, expression Eq. (22) is equivalent to Eq. (21) in the cases where one deals with regular Gaussian functions.

Based on the function performed by each neuron, the linguistic networks are trained in batch mode, via the well-known backpropagation algorithm. Regarding Takagi-Sugeno models, several alternatives are applicable. In fact, the training can be also conducted through backpropagation. However, as a consequence of the linearity in the output layer, it is possible to apply the least square estimator in matrix form. This strategy has the advantage of leading to a significant reduction of the number of epochs required. However, the time necessary for each epoch will be greater.

4.2. Training methodologies

Based on the function performed by each neuron, the network is trained in batch mode through backpropagation. The training of the fuzzy neural network starts by defining a criterion for error minimization. The sum squared error (SSE) is used. In this way, the total network error E Eq. (23), is defined as the sum of squared errors $E^{(p)}$ Eq. (24), computed for each training pattern p.

$$E = \sum_{p=1}^{N} E^{(p)} \tag{23}$$

$$E^{(p)} = \frac{1}{2}\sum_{o=1}^{no}(y_o^{(p)} - \hat{y}_o^{(p)})^2 \tag{24}$$

where, $\hat{y}_o^{(p)}$ stands for the pth network output pattern for the oth output variable, $y_o^{(p)}$ represents the corresponding real output sample and *no* denotes the number of network outputs.

In this way, every pattern is subjected to a forward pass, where signals flow from the input layer until the output layer, where the error for that specific pattern is calculated.

Next, starting from the output layer, a backward pass takes place, in which the network parameters are adjusted towards error minimization. The minimization procedure is conducted iteratively via the gradient descent method, as follows Eq. (25):

$$\Delta w = -\gamma \frac{\partial E^{(p)}}{\partial w} \tag{25}$$

where w denotes, generically, any adjustable network parameter, or weight, and γ represents the network learning rate. Typically, the error derivative relative to the weight is calculated by the chain rule as follows Eq. (26):

$$\frac{\partial E^{(p)}}{\partial w} = \frac{\partial E^{(p)}}{\partial a^{(p)}} \cdot \frac{\partial a^{(p)}}{\partial w} \tag{26}$$

In the previous expression, $a^{(p)}$ stands for the activation of any neuron in the network. In the output layer, $a^{(p)}$ is equivalent to some network output, $\hat{y}_o^{(p)}$.

As for the delta signals that need to be backpropagated, their value for an output neuron is computed as in Eq. (27). For a neuron in layer L_i, its delta value is computed via the chain rule, based on the delta signal of the following layer, L_{i+1} Eq. (28). In Eq. (28), nL_{i+1} denotes the number of neurons in layer L_{i+1}.

$$\delta^{(p)} = -\frac{\partial E^{(p)}}{\partial a^{(p)}} \tag{27}$$

$$\begin{aligned}\delta^{(pL_i)} &= -\frac{\partial E^{(p)}}{\partial a^{(pL_i)}}\\ &= \sum_{h=1}^{nL_{i+1}}\left(-\frac{\partial E^{(p)}}{\partial a_h^{(pL_{i+1})}} \cdot \frac{\partial a_h^{(pL_{i+1})}}{\partial a^{(pL_i)}}\right)\\ &= \sum_{h=1}^{nL_{i+1}}\left(\delta_h^{(pL_i+1)} \cdot \frac{\partial a_h^{(pL_{i+1}}}{\partial a^{(pL_i)}}\right)\end{aligned} \tag{28}$$

Fuzzy consequent models

Regarding the five-layered linguistic network (Fig. 3), the network training equations were generalized from [8], in order to allow the integration of two-sided Gaussian functions, as well as algebraic and truncation operators for conjunction and disjunction.

In this way, in regular Gaussians, the centers associated to the output layer are tuned via Eqs (29) e (30):

$$\delta_o^{(p5)} = y_o^{(p)} - \hat{y}_o^{(p)} \tag{29}$$

$$\frac{\partial E_o^{(p)}}{\partial c_{os}} = -\delta_o^{(p5)} \frac{\sigma_{os} a_s^{(p4)}}{\sum_{k=1}^{|T(Y_0)|} \sigma_{ok} a_k^{(p4)}} \tag{30}$$

In case two-sided Gaussians are used, Eq. (30) is replaced by Eq. (31):

$$\frac{\partial E_o^{(p)}}{\partial c_{osL}} = -\delta_o^{(p5)} \frac{\sigma_{osL} a_s^{(p4)}}{\sum_{k=1}^{|T(Y_o)|} (\sigma_{okL} + \sigma_{okR}) a_k^{(p4)}} \tag{31}$$

The previous equation is related to the left side of the Gaussian function (which will be used throughout this paper). As for the right sided, the expressions are exactly the same, except for the subscript L, which is substituted by R.

Regarding the standard deviations, their tuning is performed as in Eq. (32) for regular Gaussians and Eq. (33) for two-sided Gaussians:

$$\frac{\partial E_o^{(p)}}{\partial \sigma_{os}} = -\delta_o^{(p5)} \cdot \frac{c_{os} a_s^{(p4)} \sum_{k=1}^{|T(Y_o)|} \sigma_{ok} a_k^{(p4)} - a_s^{(p4)} \sum_{k=1}^{|T(Y_o)|} c_{ok} \sigma_{ok} a_k^{(p4)}}{\left[\sum_{k=1}^{|T(Y_0)|} \sigma_{ok} a_k^{(p4)}\right]^2} \tag{32}$$

$$\frac{\partial E_o^{(p)}}{\partial \sigma_{osL}} = -\delta_o^{(p5)} \cdot \frac{\left[c_{osL} a_s^{(p4)} \sum_{k=1}^{|T(Y_o)|} (\sigma_{okL} + \sigma_{okR}) a_k^{(p4)} - a_s^{(p4)} \sum_{k=1}^{|T(Y_o)|} (c_{okL}\sigma_{okL} + c_{okR}\sigma_{okR}) a_k^{(p4)}\right]}{\left[\sum_{k=1}^{|T(Y_o)|} (\sigma_{okL} + \sigma_{okR}) a_k^{(p4)}\right]^2} \tag{33}$$

In the fourth layer there are no parameters to adjust. However, the delta signal must be calculated, in order to backpropagate it to the inner layers. This signal is computed based on the same signal in the following layer, as stated before, resulting in Eqs (34) and (35) for regular and two-sided Gaussians, respectively.

$$\delta_s^{(p4)} = \sum_{o=1}^{n} \delta_o^{(p5)} \cdot \frac{c_{os}\sigma_{os} \sum_{k=1}^{|T(Y_o)|} \sigma_{ok} a_k^{(p4)} - \sigma_{os} \sum_{k=1}^{|T(Y_o)|} c_{ok}\sigma_{ok} a_k^{(p4)}}{\left[\sum_{k=1}^{|T(Y_0)|} \sigma_{ok} a_k^{(p4)}\right]^2} \tag{34}$$

$$\delta_s^{(p4)} = \sum_{o=1}^{n} \delta_o^{(p5)} \cdot \left[\frac{(c_{osL}\sigma_{osL} + c_{osR}\sigma_{osR}) \sum_{k=1}^{|T(Y_o)|} (\sigma_{okL} + \sigma_{okR}) a_k^{(p4)}}{\left[\sum_{k=1}^{|T(Y0)|} (\sigma_{okL} + \sigma_{okR}) a_k^{(p4)}\right]^2} - \frac{(\sigma_{osL} + \sigma_{osR}) \sum_{k=1}^{|T(Y_o)|} (c_{okL}\sigma_{okL} + c_{okR}\sigma_{okR}) a_k^{(p4)}}{\left[\sum_{k=1}^{|T(Y0)|} (\sigma_{okL} + \sigma_{okR}) a_k^{(p4)}\right]^2} \right] \tag{35}$$

As for the third layer, once again there are no parameters to adjust. So, the only task to perform is to calculate the delta signals. Generically, it comes Eq. (36):

$$\delta_r^{(p3)} = \sum_{o=1}^{no_r} \delta_o^{(p4)} \frac{\partial a_o^{(p4)}}{\partial a_r^{(p3)}} \tag{36}$$

where no_r stands for the number of consequents defined for rule r.

In the original version [8], the disjunction is performed by means of bounded-sum, which is not continuously differentiable. In order to overcome that problem, simple sum is used, resulting Eq. (37):

$$\frac{\partial a_o^{(p4)}}{\partial a_r^{(p3)}} = 1 \tag{37}$$

In case the S-norm is performed by maximum, some care must be taken while calculating the derivative. In this case, it is necessary to save the index associated to the neuron that originated the maximum. Thus, the derivative regarding that element (the "winner") will be 1, being 0 for the "loosers" Eq. (38):

$$\frac{\partial a_o^{(p4)}}{\partial a_r^{(p3)}} = \begin{cases} 1, \ a_o^{(p4)} = a_r^{(p3)} \\ 0, \ a_o^{(p4)} \neq a_r^{(p3)} \end{cases} \tag{38}$$

In the second layer, there are again parameters to adjust. Their tuning is conducted based on Eqs (39) e (40), for the centers, and Eqs (41) e (42), for the widths:

$$\frac{\partial E_o^{(p)}}{\partial c_{ij}} = \left(\sum_{r=1}^{nr_i} \delta_r^{(p3)} \frac{\partial a_r^{(p3)}}{\partial a_i^{(p2)}}\right) \frac{\partial a_i^{(p2)}}{\partial c_{ij}} \tag{39}$$

$$\frac{\partial a_i^{(2)}}{\partial c_{ijL}} = \frac{x_j - c_{ijL}}{\sigma_{ijL}^2} e^{-\frac{(x_j - c_{ijL})^2}{2\sigma_{ijL}^2}} \tag{40}$$

$$\frac{\partial E_o^{(p)}}{\partial \sigma_{ij}} = \left(\sum_{r=1}^{nr_i} \delta_r^{(p3)} \frac{\partial a_r^{(p3)}}{\partial a_i^{(p2)}}\right) \frac{\partial a_i^{(p2)}}{\partial \sigma_{ij}} \tag{41}$$

$$\frac{\partial a_i^{(2)}}{\partial \sigma_{ijL}} = \frac{(x_j - c_{ijL})^2}{\sigma_{ijL}^2} e^{-\frac{(x_j - c_{ijL})^2}{2\sigma_{ijL}^2}} \tag{42}$$

where nr_i represents the number of rules that have neuron i as antecedent and j stands for the input variable associated with the ith membership function.

In this case, centers are tuned in exactly the same manner, both for regular and two-sided Gaussians. The only difference is that two-sided Gaussians have two centers to adapt.

As in layer 3, it is necessary to choose an operator to implement the T-norm. In the original version [8], minimum is used Eq. (43):

$$\frac{\partial a_r^{(p3)}}{\partial a_i^{(p2)}} = \begin{cases} 1, \ a_r^{(p3)} = a_r^{(p2)} \\ 0, \ a_r^{(p3)} \neq a_i^{(p2)} \end{cases} \tag{43}$$

In the previous equation, the same artifice used in Eq. (38) was performed, since those operators are not continuous. Alternatively, product, which is continuously differentiable, can be used. In this case, it comes Eq. (44):

$$\frac{\partial a_r^{(p3)}}{\partial a_i^{(p2)}} = \prod_{k=1}^{na_r} a_k^{(p2)}, \ k \neq i \tag{44}$$

Usually, minimum and maximum operators, referred in this section, are called truncation operators, whereas sum and product are called algebraic operators. Algebraic operators have some advantages. Namely, they lead to smoother output surfaces and permit the direct application of gradient descent without the artifices used with truncation operators.

Tuning the parameters without any constraints, can lead to inconsistent membership functions. In fact, it makes no sense to have neither negative standard deviations nor two-sided Gaussian functions with right and left centers exchanged. Therefore, after adjusting the parameters, the integrity of the membership functions must be verified and guaranteed. Thus, in case centers in two-sided Gaussians get exchanged, it was decided to attribute them their mean value. As for standard deviations, in case they become negative, they are given a "small" value, which is quantified as 1% of the domain amplitude. Formally, it results Eq. (45).

$$\begin{aligned} &c_L > c_R \Rightarrow \begin{cases} c_L^{\text{new}} = \frac{c_L + c_R}{2} \\ c_R^{\text{new}} = \frac{c_L + c_R}{2} \end{cases} \\ &\sigma < 0 \Rightarrow \sigma = \frac{X_{\max} - X_{\min}}{100} \end{aligned} \tag{45}$$

where the domain interval is $[X_{\min}, X_{\max}]$. However, it is important to note that, with the constraints imposed, the true gradient is not followed any longer. Instead, an approximation is performed.

Takagi-Sugeno models

Regarding Takagi-Sugeno fuzzy models, zero or first-order, several alternatives are applicable.

In a first alternative, the network can be trained via backpropagation. In this way, the only differences comparatively to fuzzy consequents models regard parameter tuning in the linear output layer and calculating the delta signals necessary to the rule layer.

However, since the output layer is linear, the least squares estimator can be applied in matrix form. In this way, matrix Eq. (46) is obtained:

$$Y^T = \Phi^T \cdot B \tag{46}$$

In Eq. (46), B denotes a $(m+1) \cdot g \times n$ matrix with parameters to identify, defined as follows Eq. (47):

$$B = \begin{bmatrix} b_{110} & b_{210} & & & b_{n10} \\ b_{111} & b_{211} & & & b_{n11} \\ \vdots & \vdots & \vdots & \vdots & \\ b_{11m} & b_{21m} & & & b_{n1m} \\ & & & & \\ \vdots & \vdots \ldots & b_{orj} & \ldots & \vdots \\ & & & & \\ b_{1g0} & b_{2g0} & & & b_{ng0} \\ b_{1g1} & b_{2g1} & & & b_{2g1} \\ \vdots & \vdots & \vdots & \vdots & \\ b_{1gm} & b_{2gm} & & & b_{ngm} \end{bmatrix} \tag{47}$$

In Eq. (47), each column in matrix B represents one model output. Each of those columns has $(m+1) \cdot g$ parameters associated. The same matrix can be divided in g sets of $m+1$ lines (separated by dashed lines). Each of those sets contains the parameters defined for the consequents of each rule. Thus, the total number of parameters for Takagi-Sugeno models will be $(m+1) \cdot g \cdot n$. In zero-order models, each rule will have a unique parameter, b_{or0}, and so matrix B will be $g \times n$.

Still in Eq. (46), Y represents an $n \times N$ matrix containing real output data, and Φ is a $(m+1) \cdot g \times N$ matrix, defined in Eq. (48), where $x_i^{(p)}$ is the pth pattern received by input i. For zero-order models, vector $X^{(p)}$ is reduced to $X^{(p)} = 1$.

$$\Phi = \begin{bmatrix} \frac{a_1^{(13)}}{\sum_{r=1}^{g} a_r^{(13)}} X^{(1)} & \frac{a_1^{(23)}}{\sum_{r=1}^{g} a_r^{(23)}} X^{(2)} & & \frac{a_1^{(N3)}}{\sum_{r=1}^{g} a_r^{(N3)}} X^{(N)} \\ \frac{a_2^{(13)}}{\sum_{r=1}^{g} a_r^{(13)}} X^{(1)} & \frac{a_2^{(23)}}{\sum_{r=1}^{g} a_r^{(23)}} X^{(2)} & & \frac{a_2^{(N3)}}{\sum_{r=1}^{g} a_r^{(N3)}} X^{(N)} \\ \vdots & \vdots & \ldots & \vdots \\ \frac{a_g^{(13)}}{\sum_{r=1}^{g} a_r^{(13)}} X^{(1)} & \frac{a_g^{(23)}}{\sum_{r=1}^{g} a_r^{(23)}} X^{(2)} & & \frac{a_g^{(N3)}}{\sum_{r=1}^{g} a_r^{(N3)}} X^{(N)} \end{bmatrix}$$

$$X^{(p)} = [1\ x_1^{(p)}\ x_2^{(p)}\ \ldots\ x_m^{(p)}]^T, \quad p = 1, 2, \ldots, N \tag{48}$$

Typically, Eq. (46) can be solved in a single step, as follows Eq. (49):

$$B = (\Phi^T \Phi)^{-1} \Phi^T Y. \tag{49}$$

This requires matrix $\Phi^T \Phi$ to be positive definite. By its construction, that matrix is always positive semi-definite. However, in case it is singular, that requirement is not followed any longer. In this case, an infinite number of solutions will result. Therefore, data collection must be carried out very carefully, so that the samples obtained are informative enough, in order to avoid the problem referred.

In Eq. (49), calculating the inverse can be computationally heavy. Therefore, the recursive version of the least square estimator (RLS) is preferred Eq. (50):

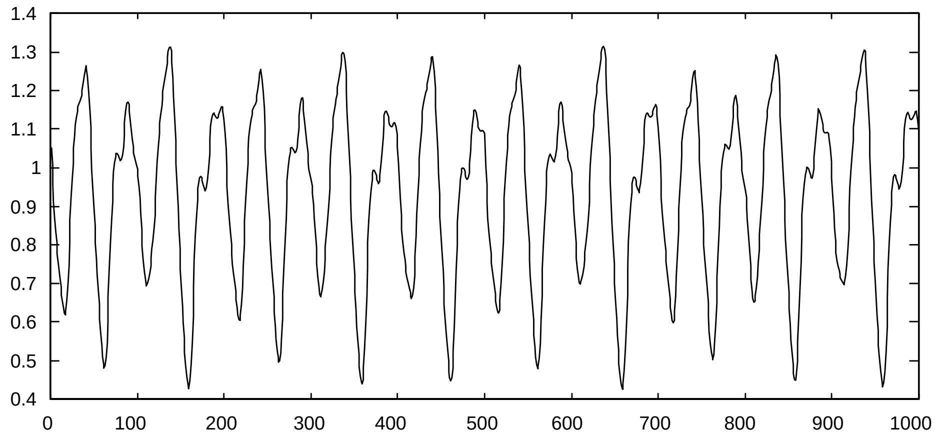

Fig. 4. Chaotic time series: identification data.

Table 2
Chaotic series: training results

	Method	Type of Gaussians	Nr. Param.	Fuzzy. Oper.	Nr. Epochs.	Time p/ Ep.	RMSE	
							Train	Test
1	FC	2-sided	180	Algebraic	2000	0.27s	0.0070	0.0076
2	"	"	"	Truncation	"	0.24s	0.0111	0.0121
3	"	Regular	90	Algebraic	"	0.26s	0.0066	0.0071
4	CC	2-sided	153	Algebraic	1500	0.54s	0.0047	0.0050
5	"	"	"	Truncation	"	0.52s	0.0097	0.0108
6	"	Regular	81	Algebraic	"	0.53s	0.0050	0.0052
7	FOC	2-sided	189	Algebraic	300	4.1s	0.0025	0.0030
8	"	"	"	Truncation	"	4.3s	0.0038	0.0043
9	"	Regular	147	Algebraic	"	4.0s	0.0030	0.0033

$$\begin{aligned} B(p) &= B(p-1) \\ &\quad + P(p)\Phi^{(p)}[Y^{(p)} - \hat{Y}^{(p)}]^T \\ P(p) &= P(p-1) \\ &\quad - \frac{P(p-1)\Phi^{(p)}\Phi^{(p)^T}P(p-1)}{1+\Phi^{(p)T}P(p-1)\Phi^{(p)}}, \\ &\quad p = 1,2,\ldots,N \end{aligned} \tag{50}$$

Here, $\Phi^(p)$ stands for the pth column in matrix Φ and $Y^{(p)}$ e $\hat{Y}^{(p)}$ are $n \times 1$ vectors that denote, respectively, real and model output.

The method described has the advantage of guaranteeing the optimum solution, for fixed parameters in the second layer. Therefore, it is usual to use it in a hybrid scheme [6]. In this scheme, the network performs a forward pass, until the rule layer. At this point, matrix Φ is determined. Next, the optimal parameters for matrix B are obtained through RLS and the modeling error is calculated. In the second phase, the network performs backpropagation, and the second layer parameters are tuned as described previously, based on the delta signal relative to the rule layer.

This scheme, which is used in this work, allows a significant reduction of the number of training epochs. However, it is important to note that the computing time for each epoch is notoriously higher than the pure application of backpropagation to the training of the network.

In order to reduce the network convergence time, an adaptive learning rate is used, both for fuzzy consequents models and Takagi-Sugeno models. Thus, if the error is reduced for num_{red} consecutive epochs, the learning rate is increased by a factor μ^{up}. In case error grows for num_{inc} consecutive epochs or oscillates for num_{osc} consecutive epochs, the learning rate is reduced by a factor of μ^{down}. As for the stopping criterion, training finishes when the Root Mean Squared Error (RMSE) reaches some satisfactory threshold, stabilizes or overtraining happens.

5. A comparative study: Simulation results

One of the most commonly used case studies in system identification consists of the prediction of the

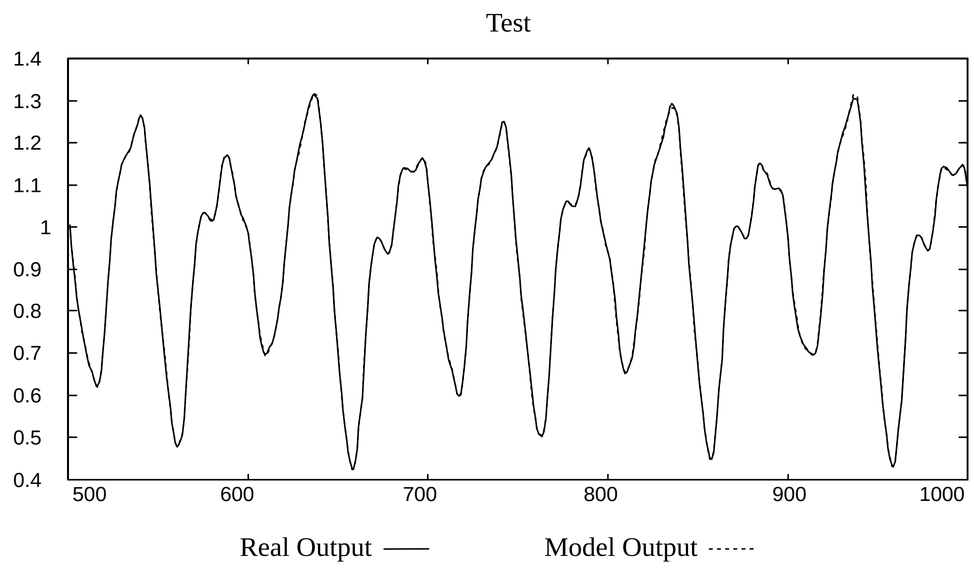

Fig. 5. Chaotic series: output prediction in a first-order Takagi-Sugeno model with algebraic operators and two-sided Gaussians.

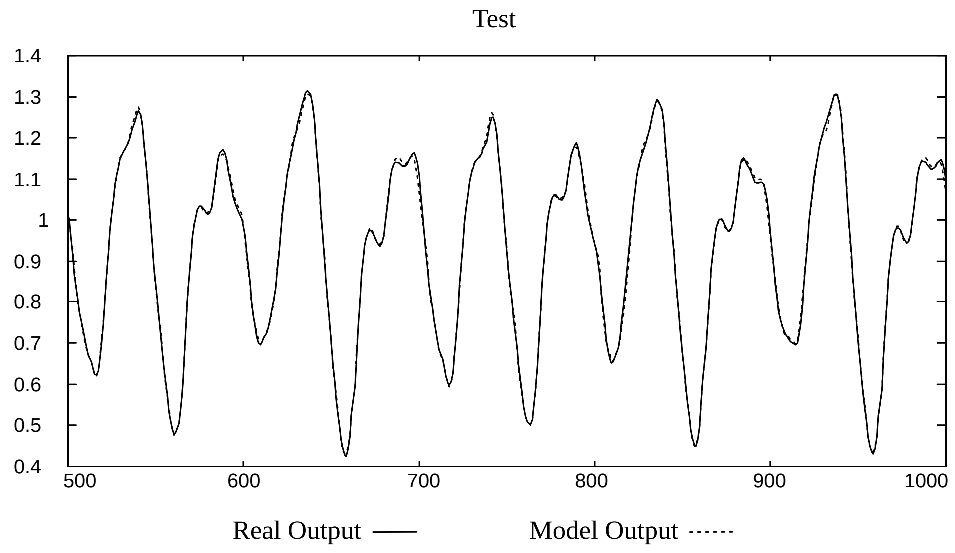

Fig. 6. Chaotic series: output prediction in a linguistic model with algebraic operators and regular Gaussians.

Mackey-Glass chaotic time series [9], described by Eq. (51):

$$\dot{x}(t) = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t) \tag{51}$$

It does not show a clear periodic behavior and it is also very sensible to initial conditions. The problem consists of predicting future values of the series.

The application of the techniques described previously is carried out based on identification data from the "IEEE Neural Network Council, Standards Committee, Working Group on Data Modelling Benchmarks", which are also used in the analysis of several other methodologies. So, in order to obtain a numeric solution, the fourth order Runge-Kutta method was applied. For integration, it was assumed $x(t) = 0,\ t < 0$, and a time interval of 0.1. The initial condition $x(0) = 1.2$ and the τ parameter ($\tau = 17$) were also defined. In this case, $[x(t-18), x(t-12), x(t-6), x(t)]$ are used to predict $x(t+6)$. Based on the parameterization described, data was obtained in the interval $t \in [0; 2000]$. Then, 1000 input-output pairs were selected from in-

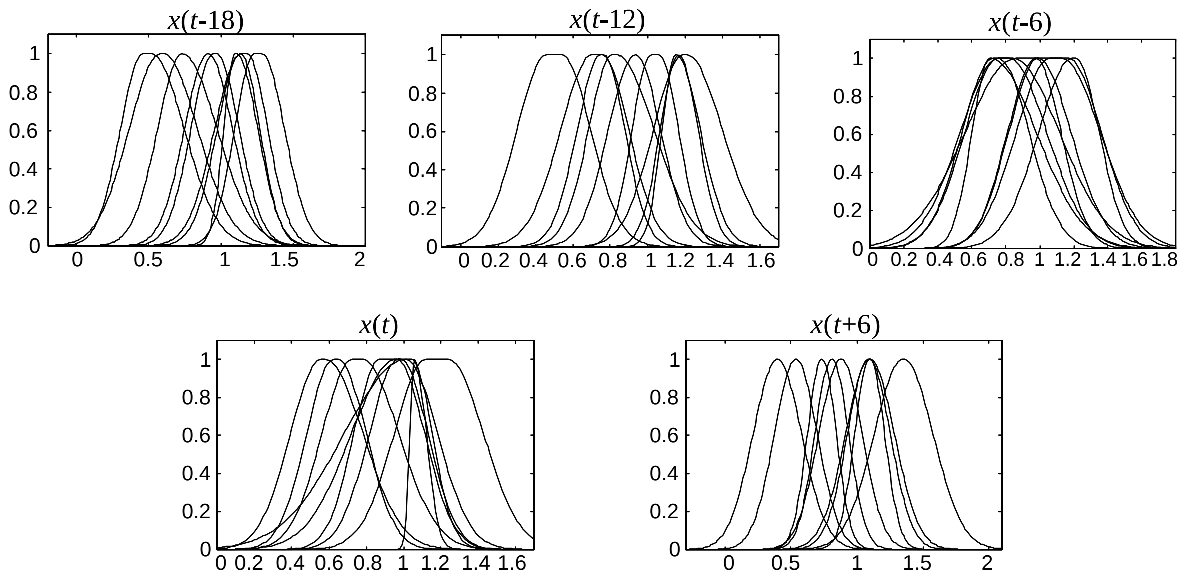

Fig. 7. Method 1: membership functions obtained.

Table 3
Method 1: network parameters

		MF1	MF2	MF3	MF4	MF5	MF6	MF7	MF8	MF9
$x(t-18)$	c_L	0.45890	0.57942	0.73123	0.90927	0.94710	1.09923	1.13540	1.13788	1.23283
	c_R	0.51762	0.60231	0.73124	0.91222	0.96767	1.10719	1.13541	1.18007	1.28358
	σ_L	0.15626	0.21683	0.17104	0.16290	0.15573	0.07965	0.16155	0.18267	0.13798
	σ_R	0.22855	0.23605	0.24308	0.16850	0.15877	0.14930	0.13472	0.14538	0.15774
$x(t-12)$	c_L	0.45925	0.70339	0.74875	0.81388	0.93538	1.01849	1.15110	1.16626	1.18986
	c_R	0.53720	0.76558	0.75697	0.83325	0.93538	1.06013	1.15110	1.16627	1.20855
	σ_L	0.15982	0.17073	0.12746	0.13132	0.14550	0.10050	0.07397	0.09911	0.15912
	σ_R	0.15850	0.11472	0.14399	0.20298	0.13431	0.10582	0.13079	0.10362	0.19192
x(t-6)	c_L	0.70946	0.72954	0.75930	0.75956	0.86176	0.97492	0.98703	1.05702	1.19090
	c_R	0.71303	0.81760	0.84009	0.75956	1.12831	0.97531	0.98705	1.16716	1.21067
	σ_L	0.10654	0.18578	0.22580	0.19969	0.30088	0.16496	0.17036	0.19722	0.20099
	σ_R	0.26867	0.22552	0.27750	0.17792	0.22936	0.15089	0.19838	0.19709	0.13985
$X(t)$	c_L	0.55795	0.62578	0.2673	0.86851	0.95754	0.96978	1.01331	1.05637	1.12134
	c_R	0.58266	0.65020	0.77437	0.96156	1.01022	0.98347	1.04280	1.05638	1.23770
	σ_L	0.16476	0.14495	0.16442	0.13607	0.23840	0.15556	0.33283	0.02425	0.17506
	σ_R	0.19925	0.14922	0.20002	0.15158	0.17186	0.14799	0.11069	0.06402	0.19160
$x(t+6)$	c_L	0.39628	0.53563	0.73117	0.80466	0.87284	1.08335	1.09021	1.09188	1.34457
	c_R	0.39659	0.53869	0.73476	0.80815	0.88113	1.08971	1.09708	1.09832	1.35395
	σ_L	0.18464	0.16231	0.10678	0.12415	0.16290	0.15945	0.11147	0.18100	0.21765
	σ_R	0.18453	0.16027	0.10587	0.12569	0.15706	0.16103	0.11065	0.18301	0.22208

terval $t \in [118; 1117]$. The data collected is depicted in Fig. 4. This data set was divided in order to get the training and validation suits: first and last 500 samples, respectively.

Using the samples obtained, the chaotic time series was modeled, according to the procedures described in the previous sections. The parameter r_a was assigned the value 0.5, resulting 9 fuzzy rules. Next, the network, with four inputs and one output, was trained.

After the application of the set of methodologies described, the results in Table 2 were obtained, where FC, CC and FOC denote respectively fuzzy, constant and first order consequents. The algorithms described were programmed in C++ and the experiments were conducted on a PC under Microsoft Windows © NT, running at 266 MHz on a Pentium II processor, with 64 MB RAM.

The results presented allow some conclusions to

Table 4
Method 1: rule base obtained by the proposed method

Rule	$x(t-18)$	$x(t-12)$	$x(t-6)$	$x(t)$	$\Rightarrow$	$x(t+6)$
1	MF1	MF4	MF6	MF6		MF7
2	MF2	MF1	MF3	MF5		MF5
3	MF3	MF2	MF5	MF9		MF9
4	MF4	MF6	MF8	MF7		MF8
5	MF5	MF3	MF1	MF3		MF6
6	MF6	MF7	MF9	MF8		MF3
7	MF7	MF8	MF7	MF4		MF2
8	MF8	MF5	MF2	MF2		MF4
9	MF9	MF9	MF4	MF1		MF1

Table 5
Comparison with methods from other authors

	Method	Number of rules	Number of free parameters	Error index
1	FC	9	90	0.033
2	CC	9	153	0.023
3	FOC	9	189	0.014
4	Chiu	25	125	0.014
5	ANFIS	16	104	0.007
6	ANN with backpropagation	–	540	0.02
7	NEFPROX (A)	129	105	0.155
8	NEFPROX (G)	26	38	0.313

be drawn. Using regular Gaussian functions presents some advantages in case fuzzy consequents are utilized. In fact, the higher complexity that results from the use of two-sided Gaussian functions does not originate a significant gain in terms of model accuracy, and so regular Gaussians are preferable. As for fuzzy operators, algebraic operators lead to much better results than truncation operators. Models with constant consequents allow better results than linguistic models but first order models are the most accurate and need a considerable lower number of training epochs. However, linear optimization leads to high processing time, which may be problematic for real-time learning, unless code optimization is performed. As a consequence, zero order models appear to have a satisfactory trade-off between accuracy and computer efficiency. Figure 6 depicts the output for real and test data, regarding method 1. That figure shows the high prediction accuracy of the model, which is not the best one achieved in the simulations. In Fig. 5, the results for method 9 (first-order consequents, regular Gaussian functions and algebraic operators) are presented, where real output data can hardly be distinguished from the model output.

Despite the fact that only one experiment is presented, the authors tested the methodology in many commonly used benchmark problems (Box-Jenkins gas furnace, Narendra's benchmarks, etc.), getting consistent conclusions, which could be generalized.

One interesting point regarding method 1 (fuzzy consequents) is that, though it is called a linguistic model, it is hard to assign linguistic labels to its membership functions, as can be seen in Fig. 7 (the parameters for the membership functions in that figure are presented in Table 3, where MF stands for membership function). In fact, the membership functions are strongly overlapped, which makes it very hard to label them. Thus, the functions are labeled as MF1, MF2, . . . , MF9, and the rule base obtained (Table 4) is not very intelligible, in terms of human cognition. This problem of interpretability (or transparency) is being object of research by the authors.

Comparison with other methods

This example has been treated before by other methods. Table 5 shows the comparison data.

The first three lines refer to the results of the present work; lines 4 to 6 are extracted from [2]; lines 7 and 8 are adapted from [11]. From there, it can be concluded that neural networks trained by backpropagation require a significantly higher number of adjustable parameters than other methods, based on fuzzy systems. Moreover, fuzzy systems require fewer training epochs. Also, the fact that in the present work all the model parameters of first-order TS model are adapted (this does not happen in other methods) allows the number of rules to be reduced, without losing model precision, and as a consequence the number of rules is significantly reduced. In other methods, e.g., [2], the consequents are optimized by means of the least square estimator, but the antecedents are not changed. Thus, the model obtained is not fully optimized and, consequently, more rules are necessary to achieve the same model accuracy. If regular Gaussian functions would be used (leading to a small degradation of model accuracy), the number of free parameters would be reduced significantly, as can be seen in Table 2, by comparing lines 7 and 9. The ANFIS results are superior, but at the cost of a higher number of rules.

6. Conclusions

In this paper a comparative analysis of different fuzzy structures and parameterizations is performed. By the application of subtractive clustering an initial structure for the fuzzy model is obtained, which is used for the initialization of a fuzzy neural network. Next, the parameters are adjusted through the training of the network, according to the structure defined, i.e., linguistic, zero order or first order Takagi-Sugeno. Algebraic and truncation operators are used, as well as regular and two-sided Gaussian functions. The techniques described are applied to the Mackey-Glass time series, having been concluded that first order models are the most accurate. However, zero order models present the best trade-off between model accuracy and computer efficiency. In terms of fuzzy operators, algebraic operators lead to more precise models. As for membership functions, it was concluded that the additional complexity of two-sided Gaussians did not originate a significant gain. So, regular Gaussian functions are preferable.

Acknowledgements

This work was supported partially by the Portuguese Ministry of Science and Technology (MCT), under the project ALCINE PRAXIS/P/EEI/14155/98 and POSI Program financed by European Union.

The authors would like to thank all the comments and suggestions addressed by the anonymous reviewers that contributed to the improvement of the final version.

References

[1] M.F. Azeem, M. Hanmandlu and N. Ahmad, Generalization of adaptive neuro-fuzzy inference system, *IEEE Transactions on Neural Networks* **11**(6) (2000), 1332–1346.

[2] S.L. Chiu, Fuzzy model identification based on cluster estimation, *Journal of Intelligent and Fuzzy Systems* **2**(3) (1994), 267–278.

[3] R.N. Davé and R. Krishnapuram, Robust clustering methods: a unified view, *IEEE Transactions on Fuzzy Systems* **5**(2) (1997), 270–293.

[4] X. Hong and C.J. Harris, Generalized neuro-fuzzy network modelling algorithm using Bezier-Bernstein polynomial functions and additive decomposition, *IEEE Transactions on Neural Networks* **11**(4) (2000), 889–902.

[5] M. Jamshidi, A. Titli, L.A. Zadeh and S. Boverie, *Applications of Fuzzy Logic – Towards High Machine Intelligence Quotient Systems,* Prentice Hall, Upper Saddle River, NJ, 1997.

[6] J.-S.R. Jang, ANFIS: Adaptive Network-based Fuzzy Inference System, *IEEE Transactions on Systems, Man and Cybernetics* **23**(3) (1993), 665–685.

[7] J. Kim and N. Kabasov, Adaptive Neuro-Fuzzy inference systems and their application to nonlinear dynamical systems, *Neural Networks* **12**(9) (1999), 1301–1319.

[8] C.-T. Lin, A neural fuzzy control scheme with structure and parameter learning, *Fuzzy Sets and Systems* **70** (1995), 183–212.

[9] M.C. Mackey and L. Glass, Oscillation and chaos in physiological control systems, *Science* **197** (1977), 287–289.

[10] E.H. Mamdani, Applications of fuzzy algorithms for control simple dynamic plants, *Proc. of the IEE* **121** (1974), 1585–1588.

[11] D. Nauck and R. Kruse, Neuro-fuzzy systems for function approximation, *Fuzzy Sets and Systems* **101** (1999), 261–271.

[12] K. Pal and N.R. Pal, A neuro-fuzzy system for inferencing, *International Journal of Intelligent and Fuzzy Systems* **14**(11) (1999), 1155–1182.

[13] R.P. Paiva, Identificação Neuro-Difusa: Aspectos de Interpretabilidade (Neuro-Fuzzy Identification: Interpretability Issues), MSc Thesis, Department of Informatics Engineering, Faculty of Sciences and Technology, University of Coimbra, Portugal (in Portuguese), 1999.

[14] R.P. Paiva, A. Dourado and B. Duarte, Applying subtractive clustering for neuro-fuzzy modelling of a bleaching plant, *Proceedings of the European Control Conference – ECC'99,* CD-ROM1999.

[15] T. Takagi and M. Sugeno, Fuzzy identification of systems and its applications to modelling and control, *IEEE Transactions on Systems, Man and Cybernetics* **15**(1) (1985), 116–132.

[16] L.A. Zadeh, Outline of a new approach to the analysis of complex systems and decision processes, *IEEE Transactions on Systems, Man and Cybernetics* **3**(1) (1973), 28–44.

[17] J. Zhang and A.J. Morris, Recurrent neuro-fuzzy networks for nonlinear process modelling, *IEEE Transactions on Neural Networks* **10**(2) (March 1999), 313–326.