Tuning Complex Fuzzy Systems by Supervised Learning Algorithms

F. J. Moreno-Velo, I. Baturone, R. Senhadji, S. Sánchez-Solano

Instituto de Microelectrónica de Sevilla (IMSE-CNM)
Centro Nacional de Microelectrónica - CSIC
Avda. Reina Mercedes, s/n. Edif. CICA, E-41012, Sevilla, Spain
xfuzzy-team@imse.cnm.es

Abstract

Tuning a fuzzy system to meet a given set of input/output patterns is usually a difficult task that involves many parameters. This paper presents an study of different approaches that can be applied to perform this tuning process automatically, and describes a CAD tool, named xfsl, which allows applying a wide set of these approaches: (a) a large number of supervised learning algorithms; (b) different processes to simplify the learned system; (c) tuning only specific parameters of the system; (d) the ability to tune hierarchical fuzzy systems, systems with continuous output (like fuzzy controller) as well as with categorical output (like fuzzy classifiers), and even systems that employ user-defined fuzzy functions; and, finally, (e) the ability to employ this tuning within the design flow of a fuzzy system, because xfsl is integrated into the fuzzy system development environment Xfuzzy 3.0.

1. Introduction

Tuning the system behavior is often one of the most difficult task in the design flow of a fuzzy system. Much of the tuning effort is dedicated to search a proper configuration of the system parameters, because there is usually a large number of parameters. To confront this task, supervised learning algorithms are commonly used as automatic tuning methods. In supervised learning techniques, the desired system behavior is described by a set of input/output patterns and the objective is to minimize the error between the desired and the current system behavior.

The paper is structured as follows. Section 2 presents the different error functions that can be employed in supervised learning. Section 3 describes briefly some families of supervised learning algorithms. The problem of tuning under constraints is addressed in Section 4, while Section 5 summarizes some simplification processes that can be easily performed after tuning. Most of these possibilities has been included into a CAD tool, named xfsl, so as to automate the tuning process of complex fuzzy systems. This tool is briefly described in Section 6. Finally, Section 7 and 8 show several examples to illustrate the tuning of systems with either continuous or categorical outputs.

2. The error function

The first step in elaborating a supervised learning process supposes describing system deviation by means of a function, known as *error function*. A very commonly used error function is the mean square error (MSE):

$$MSE = \frac{1}{N} \cdot \frac{1}{M} \cdot \sum_{i,j} \left(\frac{y_{ij} - \tilde{y}_{ij}}{r_j} \right)^2 \quad , i = 1, \ldots N; j = 1, \ldots M \quad (1)$$

where N is the number of data patterns, M is the number of output variables in the system, y_{ij} is the j-th output generated by the system for the i-th pattern, $\tilde{y}_{ij}$ is the correct output expressed by the training pattern, and r_j is the range of the j-th output that is used to normalize the deviations.

It can be useful for the designer to select the relative influence of every output variable on the global deviation from its intended behavior. The following function can be used in this case:

$$WMSE = \frac{1}{N} \cdot \sum_{i,j} w_j \cdot \left(\frac{y_{ij} - \tilde{y}_{ij}}{r_j} \right)^2 \quad (2)$$

where w_j is the weight of the j-th output variable on the global system error. These weights should be normalized so as to sum 1.

It can be also useful to employ the absolute value instead of the quadratic error, with the corresponding options of variable normalization and weight accounting:

$$WMAE = \frac{1}{N} \cdot \sum_{i,j} w_j \cdot \left| \frac{y_{ij} - \tilde{y}_{ij}}{r_j} \right| \quad (3)$$

The above expressions assume a numerical output from the fuzzy system. However, it is possible to define fuzzy systems whose outputs are linguistic labels, as is the case of classifiers. In these systems, the output value is the linguistic label presenting the highest activation degree as a result of the inference process. A common definition for the devi-

This work has been partially supported by the Spanish CICYT Project TIC2001-1726.

ation in the behavior of this kind of system is the number of classification errors:

$$CE = \frac{1}{N} \cdot \frac{1}{M} \cdot \sum_{i,j} \delta_{ij} \quad (4)$$

where δ_i is 1 when the classification of the pattern has been incorrect and 0 otherwise. This type of function considers equally all classification failures, without taking into account the distance from a correct classification. To consider this information it is necessary to add a new term like the following:

$$ACE = \frac{1}{NM+1} \cdot \sum_{i,j} \delta_{ij} \quad (5)$$

with

$$\delta_{ij} = \begin{cases} 0 & if \ \ y_{ij} = \tilde{y}_{ij} \\ 1 + \dfrac{\alpha_{ij} - \tilde{\alpha}_{ij}}{N \cdot M} & if \ \ y_{ij} \neq \tilde{y}_{ij} \end{cases}$$

where $\tilde{\alpha}_i$ is the activation degree of the correct label, and α_i is the activation degree of the label selected by the system.

Another way of taking into account the distances from right classifications is to consider the following classification square error, which is a differentiable function:

$$CSE = \frac{1}{N} \cdot \frac{1}{M} \cdot \sum_{i,j} (\alpha_{ij} - \tilde{\mu}_{ij})^2 \quad (6)$$

with

$$\tilde{\mu}_{ij} = \begin{cases} 1 & if \ \ y_{ij} = \tilde{y}_{ij} \\ 0 & if \ \ y_{ij} \neq \tilde{y}_{ij} \end{cases}$$

The choice of an adequate error function for the learning process depends both on the type of fuzzy system to be tuned and on the algorithm selected for performing the process. For example, if the learning algorithm belongs to the family of gradient descent algorithms, the error function must be derivable, so that the classification errors *CE* and *ACE* can not be used.

3. Supervised learning algorithms

Since the objective of supervised learning algorithms is to minimize an error function, they can be considered as algorithms for function optimization. Some supervised learning algorithms that can be used to tune fuzzy systems are briefly described in the following.

3.1. Gradient descent algorithms

The equivalence between fuzzy and neural networks led to apply the neural learning processes to fuzzy inference systems. In this sense, a well-known algorithm employed in fuzzy systems is the *BackPropagation* algorithm, which modifies the parameter values proportionally to the gradient of the error function in order to reach a local minimum. Since the convergence speed of this algorithm is slow, several modifications were proposed like using a different learning rate for each parameter or adapting heuristically the control variables of the algorithm, thus leading to *Back-Propagation with Momentum*, *Adaptive Learning Rate*, *Adaptive Step Size* or *Manhattan* algorithms. An interesting modification that improves greatly the convergence speed is to take into account the gradient value of two successive iterations. This idea is followed by the algorithms *QuickProp* and *RProp* [1].

3.2. Conjugate gradient algorithms

Since the gradient indicates the direction of maximum function variation, it may be convenient to generate not only one step but several steps which minimize the function error in that direction. This idea, which is the basis of the *steepest-descent* algorithm, has the drawback of producing a zig-zag advancing because the optimization in one direction may deteriorate previous optimizations. The solution is to advance by conjugate directions that do not interfere each other. The several conjugate gradient algorithms reported in the literature (like *Polak-Ribiere*, *Fletcher-Reeves*, *Hestenes-Stiefel*, and *One-step Secant)* differ in the equations used to generate the conjugate directions. The main drawback of the conjugate gradient algorithms is the implementation of a linear search in each direction, which may be costly in terms of function evaluations. The line search can be avoided by using second-order information, as done by the *scaled conjugate gradient* [2].

3.3. Second-order algorithms

A forward step towards speeding up the convergence of learning algorithms is to make use of second-order information of the error function. Since the calculus of the second derivatives is complex, one solution is to approximate the Hessian by means of the gradient values of successive iterations. This is the idea of the algorithms of *Broyden-Fletcher-Goldarfb-Shanno* and *Davidon-Fletcher-Powell* [3]. An special case is when the function to minimize is a quadratic error because, in this case, the Hessian can be approximated by only the first derivatives of the error function, as done by the *Gauss-Newton* algorithm. Since this algorithm can lead to instability when the approximated Hessian is not defined positive, the *Marquardt-Levenberg* algorithm solves this problem by introducing an adaptive term.

3.4. Algorithms without derivatives

The gradient of the error function can not be always calculated because it can be too costly or not defined. In these cases, optimization algorithms without derivatives can be

employed. An example is the *Downhill Simplex* algorithm, which considers a set of function evaluations to decide a parameter change. Another example is *Powell's method*, which implements linear searches by a set of directions that evolve to be conjugate [4]. These algorithms are too much slower than the previous ones. A best solution can be to estimate the derivatives from the secants or to employ not the derivative value but its sign (as *RProp* does), which can be estimated from small perturbations of the parameters.

3.5. Statistical algorithms

All the above commented algorithms do not reach the global but a local minimum of the error function. The statistical algorithms can discover the global minimum because they generate different system configurations that spread the search space. One way of broadening the space explored is to generate random configurations and choose the best of them. This is done by the *blind search* algorithm whose convergence speed is extremely slow. Another way is to perform small perturbations in the parameters to find a better configuration as done by the *algorithm of iterative improvements*. A better solution is to employ *simulated annealing algorithms* [4]. They are based on an analogy between the learning process, which is intended to minimize the error function, and the evolution of a physical system, which tends to lower its energy as its temperature decreases. Several annealing schemes (like linear, exponential, classic, fast or adaptive) have been proposed, producing different versions of the simulated annealing algorithm.

4. Tuning fuzzy systems under constraints

The parameters to adjust in a fuzzy system usually have to meet several constraints. For instance, when tuning the parameters of a Gaussian membership function, the learning algorithms should always reject a negative value for the parameter representing the width of the function. The constraints that usually appear when tuning a fuzzy system parameter, p_i, are the following: "$p_i <= constant$", "$p_i < constant$", "$p_i >= constant$", "$p_i > constant$", or "$p_i < p_j$". The latter ones appear, for instance, between the three points that can define a triangular membership function. They are the most difficult constraints to maintain and should be avoided as much as possible. In this sense, a triangular function is better defined by its center, width, and slope.

Statistical algorithms manage constraints easily, by directly rejecting the random configuration generated if it does not meet the constraints. On the other hand, the algorithms based on any kind of gradient descent generate the same (deterministic) displacement at a given iteration, so that the solution is not to reject it if it is forbidden (it would be again generated in the next iteration) but to change it into another displacement accepted by the system. The usual solution is to try an smaller displacement in the same direction, as shown in Figure 1a. The problem is that this solution does not distinguish between parameters, so that the constraint on one parameter can limit not only the displacement of that parameter but also of other ones (as shown in Figure 1a). This problem can be avoided by managing the parameters independently. This means, in the example of Figure 1, that the system could be moved to the final point shown in Figure 1b. For those constraints relating several parameters (as those of the triangle shown in Figure 1c), a good solution is to consider them as particles subject to inelastic collisions in a one dimensional space (Figure 1d).

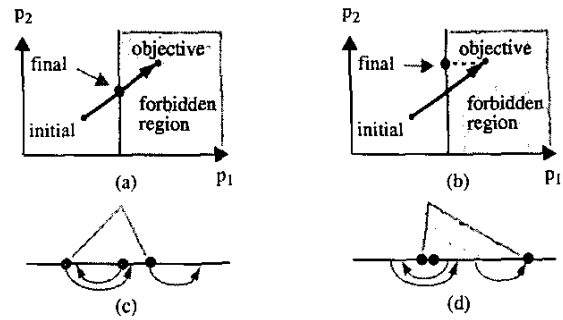


Figure 1: Learning under constraints.

Another interesting point to remark is that the stable point reached by the system after learning under constraints will provide or not the minimum possible error depending on the tuning algorithm employed. When the proposed displacement follows the gradient direction, the system evolves to a point where the gradient components over the non constrained parameters are null, that is, to the point of the frontier where the error is minimum (the point "b" in Figure 2). On the other hand, if the learning algorithm generates a direction which is not parallel to the gradient (like that shown with a dashed line in Figure 3), the stable point of the system will not provide a minimum error (the point "a" in Figure 2).

5. Simplification processes

Supervised learning can be used not only to tune fuzzy systems but also to help obtaining fuzzy models in identifi-

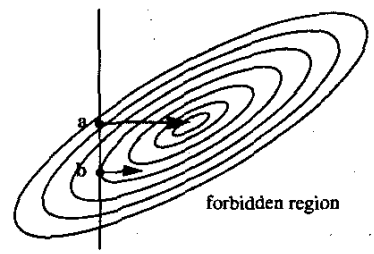


Figure 2: Stable points depending on the algorithm.

cation problems. Simplifying the fuzzy system obtained after a tuning process allows extracting a valuable information about the logical structure of the system. One of these simplification process consists in detecting and deleting those fuzzy rules and membership functions that are never activated sufficiently by any of the training input/output patterns.

A typical result of the tuning process is that the membership functions covering the output variables overlap each other in a high degree. A clustering process over these functions drives to a good and simpler fuzzy system. This clustering can be made automatically by means of the *Hard C-means* algorithm, where the number of cluster can be fixed manually, or can be selected by some cluster evaluation functions [5].

6. The Xfsl tool

In order to automate the tuning process of a fuzzy system we have developed the tool *xfsl*. *Xfsl* includes all the error functions and supervised learning algorithms described in Sections 2 and 3, and apply the solutions described in Section 4 to allow learning efficiently under constraints. Simplification methods described in Section 5 are also included and can be executed prior to or after the learning algorithms. In addition, the system parameters to tune can be selected by a graphical interface.

Xfsl allows the user to apply supervised learning algorithms to fuzzy systems specified with the XFL3 language [6], the formal language of Xfuzzy 3.0 [7]. XFL3 permits the description of complex fuzzy systems with hierarchical rule bases. Besides, there is no limitation in the number of rules within a rule base, linguistic variables, or linguistic labels covering the variables. The rules support complex logic relations in the premise part (with conjunctions, disjunctions, and linguistic hedges), and these operators as well as the implication operators, membership functions, or defuzzification methods can be defined freely by the user. The language XFL3 is the nexus between the different Xfuzzy 3.0 tools (dedicated to description, learning, verification, or synthesis of fuzzy systems).

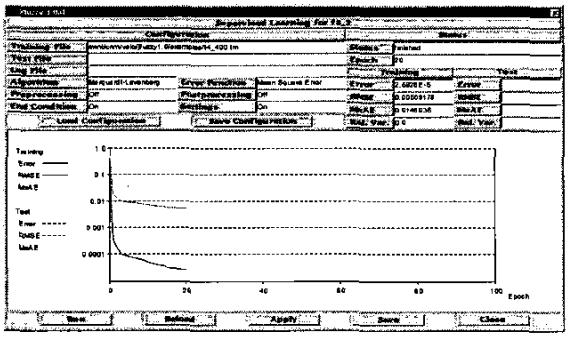


Figure 3: Main window of the tool *xfsl*.

Figure 3 illustrates the main window of *xfsl*. This window is divided into four parts. The left upper corner is the area to configure the learning process. The process state is shown at the right upper part. The central area illustrates the evolution of the learning, and the bottom part contains several control buttons to run or stop the process, to save the results, and to exit.

7. Fuzzy systems with continuous output

Since fuzzy systems with continuous output can be seen as interpolators, let us consider, as example, the problem of approximating the function shown at Figure 4a. We consider a two-input fuzzy system with 7 Gaussian membership functions per input, thus containing 49 rules. The Weighted Fuzzy Mean is selected as defuzzification method. Initially, input membership functions are homogeneously distributed in their universe of discourse, while the 49 output functions are equal and centered in their universe. All the parameters of these membership functions as well as the weights are going to be tuned, which means 126 parameters.

Figures 4b, c, and d show the evolution of the system behavior while being tuned by the different learning algorithms provided by *xfsl*. The gradient descent algorithms are shown in Figure 4b. It can be seen that modifications to the *BackPropagation* algorithm notoriously increase the convergence speed. Figure 4c is dedicated to the conjugate gradient and second order algorithms. As it is shown on this figure, these algorithms are significantly faster than the *steepest descent* algorithm, especially *BFGS* and *Marquardt-Levenberg* algorithms. Algorithms without derivatives and statistical algorithms are shown in Figure 4d. These algorithms are several orders of magnitude slower than the previous ones, so their use is only recommended when those are discarded (in non-derivable systems, for instance). It can be seen that *Powell*'s algorithm is much faster than *Downhill Simplex* algorithm. Concerning the statistical algorithms, *Simulated Annealing* increases the convergence speed with respect to *Blind Search* or *Iterative Improvement* algorithms.

The existence of non-linear parameters generates the presence of several local minima in the tuning process. Therefore, it is not possible to assert what is the best algorithm, since a very fast algorithm may be sometimes driven to a local minimum far away from the optimum behavior. A solution to this problem is to make several tuning processes with different random initial configurations, selecting the best of the learning results.

Within the learning process, the membership functions of the output variable tend to group around some common

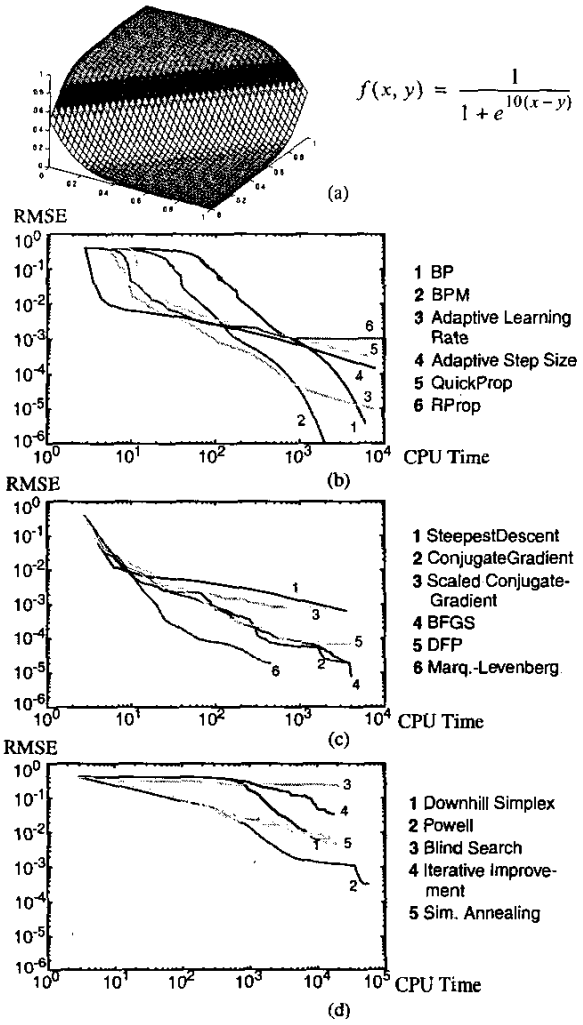
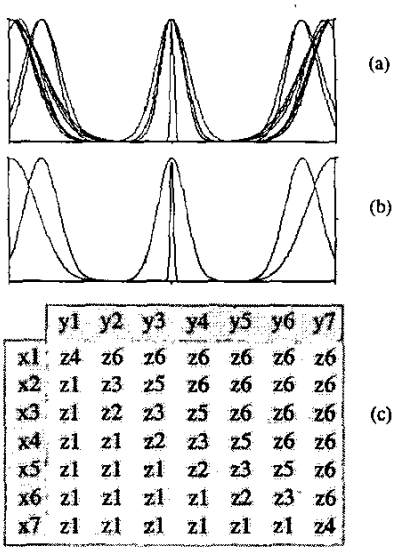


Figure 4: Comparison of the different algorithms.

forms (Figure 5a). Applying the clustering process supported by *xfsl*, the 49 membership functions are reduced to 6 (Figure 5b). This reduction leads to the simplified rule base shown in Figure 5c.

The use of hierarchical structures allows simplifying the description of a system because complex behaviors can be generated by composing simple rule bases. A relevant advantage of the *xfsl* tool is its ability to adjust hierarchical systems. To illustrate this type of learning process, we will consider again the problem of approximating the behaviour at Figure 4a, but now using a hierarchical fuzzy system with two cascaded rule bases, like that shown in Figure 6a. The initial description we have taken for the first rule base is very simple: two fuzzy sets for each input variable and four singleton values for the output, thus giving 4 rules (Figure 6b). The second rule base employs only one fuzzy set for the



	y1	y2	y3	y4	y5	y6	y7
x1	z4	z6	z6	z6	z6	z6	z6
x2	z1	z3	z5	z6	z6	z6	z6
x3	z1	z2	z3	z5	z6	z6	z6
x4	z1	z1	z2	z3	z5	z6	z6
x5	z1	z1	z1	z2	z3	z5	z6
x6	z1	z1	z1	z1	z2	z3	z6
x7	z1	z1	z1	z1	z1	z1	z4

Figure 5: Rule base obtained after learning and clustering.

input because two other ones are generated by using linguistic hedges, and three singleton values for the output (Figure 6c). The defuzzification method performed by both rule bases is the Fuzzy Mean method. Since initially all the output values are equal, the input-output relation provided by this system is flat.

The influence of the parameters on the global behavior of a hierarchical system is complex. In terms of learning, this means the existence of a lot of local minima which make no useful the application of gradient based learning algorithms. Contrary to the previous example, the statistical algorithms provide now better results. In particular, we have used the *Blind Search* algorithm followed by *Marquardt-Levenberg*'s algorithm. In the latter algorithm, *xfsl* does not compute the derivatives (it is not possible in hierarchical systems) but estimates them from small parameter modifications.

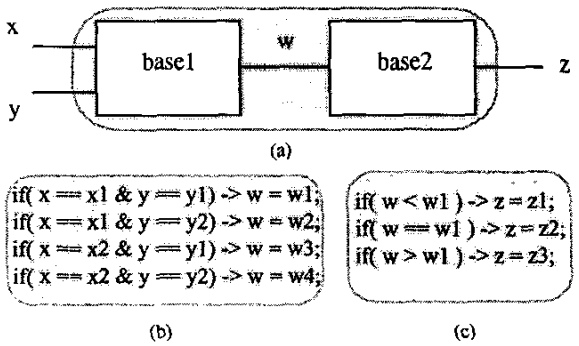


Figure 6: Tuning a hierarchical system.

After learning, the global system behavior approximates the target behavior with an RMSE of 0.41%. What is very interesting is that the rule bases have learnt the intrinsic composition of the target function. The first rule base identifies a subtracting relation between y and x (with a certain scaling factor), while the second rule base identifies an stepwise relation. Between 25 and 36 rules are required by a grid-based fuzzy system (using the Fuzzy Mean defuzzification method) to perform as well as a hierarchical system with only 7 rules, thus showing the importance of tuning hierarchical descriptions for a CAD tool.

8. Fuzzy systems with discrete output

A fuzzy classifier can be seen as a fuzzy system in which the membership functions of the output variable represent the different categories to which the output may belong. The output provided by a fuzzy classifier is generally the output membership function with the highest activation degree. Since the output values of these systems are categories, the learning process faces an additional obstacle. It should modify the parameters of the membership functions associated to the input variables to improve the success rate of the classification. Contrary to the case of fuzzy interpolators, fuzzy classifiers can perform better when reducing their rule bases since classification boundaries not parallel to the grid partition can be obtained.

As an example of tuning a fuzzy classifier, let us consider the problem illustrated in Figure 7. It shows a set of 80 data grouped into 4 different categories with 20 data each one (C1, C2, C3, and C4). The fuzzy classifier to be tuned by *xfsl* contains 9 rules initially, with 3 membership functions covering each input variable, as shown at the top and left parts of Figure 7.

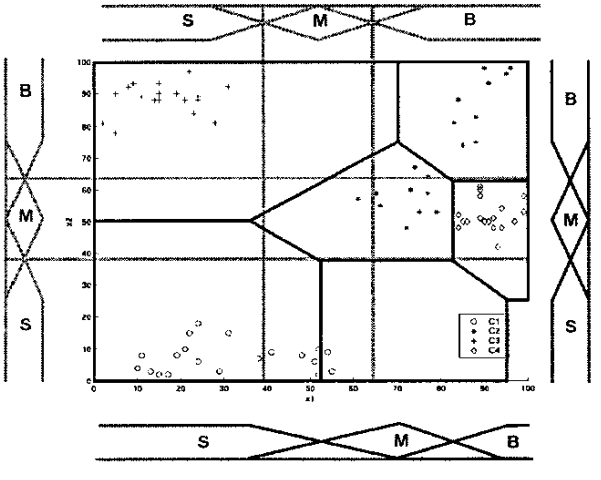


Figure 7: Example of a classification problem.

Applying the pruning process of *xfsl*, the 9 rules are reduced to 6, and the membership functions are learned as shown at the bottom and right parts of Figure 7. The classification boundaries implemented by the 6 rules reach a classification rate of 100%, as shown in Figure 7.

9. Conclusions

The tool *xfsl* presented herein represents an important effort towards the automatization of the learning process in the design of fuzzy systems. The wide set of algorithms included (from gradient-based to statistical) allows solving many application problems. The incorporated methods of clustering and pruning permits the simplification of the fuzzy system considered. Its capability of tuning hierarchical fuzzy systems makes it also possible to simplify the description of a system because complex behaviors can be usually generated by composing simple rule bases. Its ability to work with systems that employ linguistic hedges allows adjusting the system as well as maintaining its linguistic meaning, which is very interesting when extracting knowledge from data. Since the tool is integrated into the environment Xfuzzy 3.0, it is possible not only to tune a system but also using other tools to graphically define it, to represent its behavior by 2-D or 3-D plots, and to simulate, monitor or synthesize software descriptions of it. As part of Xfuzzy 3.0, *xfsl* is distributed freely under the GNU General Public License from the Xfuzzy official web page (http://www.imse.cnm.es/Xfuzzy/).

References

[1] S. Haykin, "Neural Networks. A Comprehensive Foundation", IEEE Press Macmillan, 1994.

[2] Møller, M.F., "A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning", Neural Networks, Vol. 6, pp. 525-533, 1993.

[3] Scales, L.E., "Introduction to Non-Linear Optimization", Springer-Verlag New York Inc., 1985.

[4] Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P., Numerical Recipes in C, Cambridge University Press, 1997.

[5] Bezdek, J.C., Pal, N.R., "Cluster validation with generalized Dunn's indices", Proc. 2nd NZ Int. Two-Stream Conf. on ANNES, pp. 190-193, Dunedin, 1995.

[6] F. J. Moreno-Velo, S. Sánchez-Solano, A. Barriga, I. Baturone, D. R. López, "XFL3: A New Fuzzy System Specification Language", Mathware & Soft Computing, pp. 239-253, December 2001.

[7] F. J. Moreno-Velo, I. Baturone, S. Sánchez-Solano, A. Barriga, "Rapid Design of Fuzzy Systems with XFUZZY", submitted to FUZZ-IEEE'2003.