XFSL: A TOOL FOR SUPERVISED LEARNING OF FUZZY SYSTEMS

F. J. Moreno Velo I. Baturone S. Sánchez Solano A. Barriga

Instituto de Microelectrónica de Sevilla - Centro Nacional de Microelectrónica
Avda. Reina Mercedes s/n, (Edif. CICA),
E-41012, Sevilla, Spain

European Symposium on Intelligent Technologies, Hybrid Systems and their implementation on Smart Adaptive Systems (EUNITE-2001), pp. 241-246, Tenerife, Dec. 2001.

XFSL: A TOOL FOR SUPERVISED LEARNING OF FUZZY SYSTEMS

F. J. Moreno Velo I. Baturone S. Sánchez Solano A. Barriga
Instituto de Microelectrónica de Sevilla
Centro Nacional de Microelectrónica - CSIC
Avda. Reina Mercedes, s/n. Edif. CICA, E-41012, Sevilla, SPAIN
xfuzzy-team@imse.cnm.es

ABSTRACT: This paper presents Xfsl, a tool for the automatic tuning of fuzzy systems using supervised learning algorithms. The tool provides a wide set of learning algorithms, which can be used to tune complex systems. An important issue is that Xfsl is integrated into the fuzzy system development environment Xfuzzy 3.0, and hence, it can be easily employed within the design flow of a fuzzy system.

KEYWORDS: Automatic Tuning, Supervised Learning, CAD Tools, Fuzzy Systems.

1.- INTRODUCTION

One of the research areas of our group is the development of CAD tools for fuzzy system design. In particular, we are working on developing Xfuzzy 3.0, an environment which includes different tools for describing, tuning, simulating and synthesizing fuzzy systems. All these tools share the formal language XFL3 which allows defining complex systems by using hierarchical rule bases and user-defined membership functions, fuzzy connectives, and linguistic hedges [15].

The tuning stage is usually one of the most complex task when designing fuzzy systems. The system behavior depends on the logic structure of its rule base and the membership functions of its linguistic variables. The tuning process is very often focused on adjusting the different membership function parameters that appear in the system definition. Since the number of parameters to simultaneously modify is high, a manual tuning is clearly cumbersome and automatic techniques are required. The two learning mechanisms most widely used are supervised and reinforcement learning. In supervised learning techniques the desired system behavior is given by a set of training (and test) input/output data while in reinforcement learning what is known is not the exact output data but the effect that the system has to produce on its environment, thus making necessary the monitoring of its on-line behavior.

The aim of this paper is to describe the supervised learning tool, named Xfsl, integrated into Xfuzzy 3.0. This tool eases the tuning stage within the design flow of a fuzzy system. Xfsl is the evolution of the tool Xfbpa [14], which was developed for the previous version of Xfuzzy (Xfuzzy 2.0). The main advantages of Xfsl compared with its predecessor are the following: (a) it includes a wider set of learning algorithms, (b) it has the capability of implementing clustering and pruning processes of the membership functions defined in the system, and (c) it is able to apply the learning algorithms even to systems that employ user-defined fuzzy functions (membership or connective functions, linguistic hedges or particular defuzzification methods).

2.- SUPERVISED LEARNING ALGORITHMS

Since the objective of supervised learning algorithms is to minimize an error function that summarizes the deviation between the actual and the desired system behavior, they can be considered as algorithms for function optimization. The main supervised learning algorithms reported in the literature are briefly described in the following.

2.1.- GRADIENT DESCENT ALGORITHMS

The equivalence between fuzzy systems and neural networks [9] 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 [18], 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 differ-

This work has been partially supported by the Spanish CICYT Project TIC98-0869 and the FEDER Project 1FD97-0956-C3-02.

ent learning rate for each parameter or adapting heuristically the control variables of the algorithm [8]. An interesting modification that improves greatly the convergence speed is to take into account the gradient value of two successive iterations because this provides an approximated information about the curvature of the error function. This idea is followed by the algorithms *QuickProp* [4] and *RProp* [17].

2.2.- CONJUGATE GRADIENT ALGORITHMS

The gradient-descent algorithms generate a change step in the parameter values which is a function of the gradient value at each iteration (and possibly at previous iterations). 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 differ in the equations used to generate the conjugate directions [19].

The main drawback of the conjugate gradient algorithms is the implementation of a lineal search in each direction, which may be costly in terms of function evaluations. The line search can be avoided by using second-order information, that is, by approximating the second derivative with two close first derivatives. The *scaled conjugate gradient* algorithm is based on this idea [12].

2.3.- 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, that is, of its second derivatives or, in matricial form, of its Hessian. 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* [5].

An special case is when the function to minimize is a quadratic error because, in this case, the Hessian can be approximated with only the first derivatives of the error function, as done by the *Gauss-Newton* algorithm. Since this algorithm can lead to unstability when the approximated Hessian is not positive defined, the *Marquardt-Levenberg* algorithm solves this problem by introducing an adaptive term [1].

2.4.- 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 [16], which considers a set of function evaluations to decide a parameter change. Another example is *Powell's method* [2], which implements linear searches by a set of directions that evolve to be conjugate. This kind of 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.

2.5.- 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* [10]. 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 [6][11][20]. An alternative is the use of *genetic algorithms* [3][7], originated in an analogy between the evolution of species and the evolution of a population of system configurations. They broaden even more the explored area because they keep several system configurations, allowing both light perturbations on a given configuration and heavy perturbations on some other configurations.

Simulated annealing and genetic algorithms provides good results when the number of parameters to adjust is low. When it is high, the convergence speed is so extremely slow than it is usually preferred to generate random configurations, apply gradient descent algorithms to them and finally select the best solution.

3.- THE XFSL TOOL

Xfsl is a tool that allows the user to apply supervised learning algorithms to fuzzy systems specified with the XFL3 language, the formal language of Xfuzzy 3.0. 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 conjuntions, disjuntions, 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 tools included into Xfuzzy 3.0.

Figure 1 shows the main window of Xfuzzy 3.0. From this window, the user may execute the tools dedicated to description, learning, verification, or synthesis of fuzzy systems. In particular, the user may execute the learning/tuning tool Xfsl. Xfuzzy 3.0 has been programmed in Java so that it can be executed in any platform containing the JRE (Java Runtime Environment).

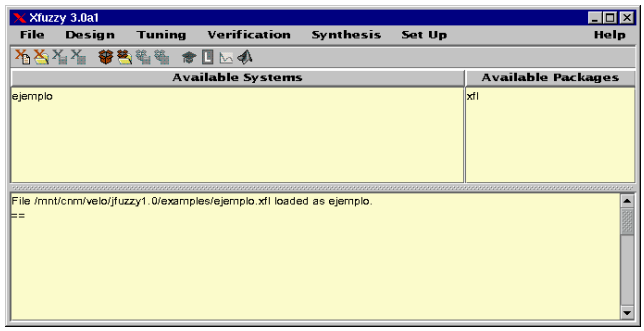


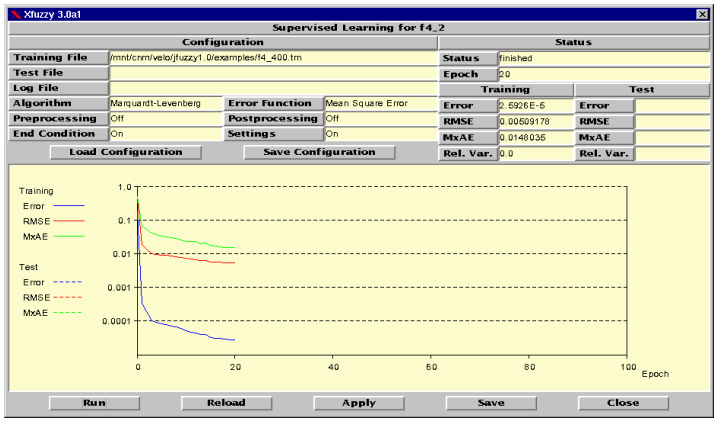
Figure 1: Xfuzzy 3.0 main window.

Figure 2a 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.

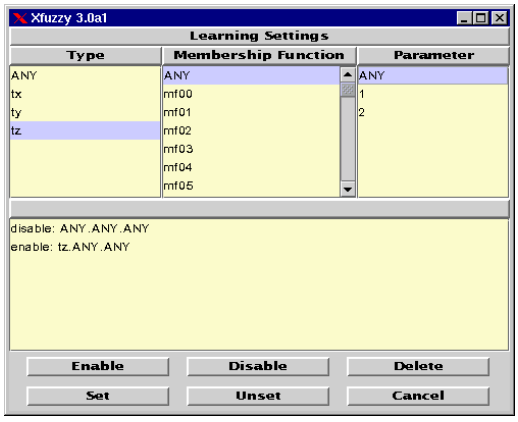
In order to configure the learning process, the first step is to select a training file that contains the input/output data of the desired behavior. A test file, whose data are used to check the generalization of the learning, can be also selected. The log file allows to save the learning evolution in an external file. An end condition has to be also specified to finish the learning process. This condition is a limit imposed over the number of iterations, the maximum error goal, or the maximum absolute or relative deviation (considering both the training or the test error). A complete learning configuration can be saved in an external file that will be available for subsequent processes.

Xfsl admits most of the algorithms commented in Section 2. Regarding gradient descent algorithms, it admits *Steepest Descent*, *Backpropagation*, *Backpropagation with Momentum*, *Adaptive Learning Rate*, *Adaptive Step Size*, *Manhattan*, *QuickProp* and *RProp*. Among conjugate gradient algorithms, the following are included: *Polak-Ribiere*, *Fletcher-Reeves*, *Hestenes-Stiefel*, *One-step Secant* and *Scaled Conjugate Gradient*. The second-order algorithms included are: *Broyden-Fletcher-Goldarfb-Shanno*, *Davidon-Fletcher-Powell*, *Gauss-Newton* and *Mardquardt-Levenberg*. Regarding algorithms without derivatives, the *Downhill Simplex* and *Powell's method* can be applied. Finally, the statistical algorithms included are *Blind Search* and *Simulated Annealing* (with lineal, exponential, classic, fast, and adaptive annealing schemes).

Xfsl can be applied to any fuzzy system described by the XFL3 language, even to systems that employ particular functions defined by the user. What must be considered is that the features of the system may impose limitations over the learning algorithms to apply (for instance, a non derivative system does not be tuned by a gradient-descent algorithm).



(a)



(b)

Figure 2: (a) Xfsl main window. (b) Selecting parameters to tune.

An interesting feature of Xfsl is that several error functions can be minimized: mean square error (the default function), mean absolute error, classification error (for classification problems), and advanced classification error (to consider not only the number of classification fails but also the distances from right classifications). If the system has several output variables, they can be weighted by different values so that the designer can select their relative influence on the global error. Another useful feature of Xfsl is that the user can select the system parameters to tune by a graphical interface, as shown in Figure 2b.

In addition, Xfsl contains two processing algorithms to simplify the designed fuzzy system. The first algorithm prunes the rules and reduces the membership functions that do not reach a significant activation or membership degree. The second algorithm clusters the membership functions of the output variables. This is very useful for system identification problems.

4.- APPLICATION EXAMPLES

Figure 3 shows three examples of the application of Xfsl to function approximations. The mathematical expression and the graphical representation of the three functions to fit are shown in Figures 3a and 3b, respectively. We intend to approximate these functions by means of a fuzzy system composed by 49 rules, two input variables (defined by 7 gaussian membership functions) and one output variable (defined by 49 gaussian membership functions). Initially, input membership functions are homogeneously distributed in their universe of discourse, while output membership functions are equal and centered in their universe. All the parameters of these membership functions are going to be tuned to fit the fuzzy system behavior to the target functions. This implies a total number of 126 parameters to tune.

Figures 3c, 3d and 3e 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 3c. It can be seen that modifications to the Back Propagation algorithm notoriously increases the convergence speed. Figure 3d 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 Mardquardt-Levenberg algorithms. Algorithms without derivatives and statistical algorithms are shown in Figure 3e. 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.

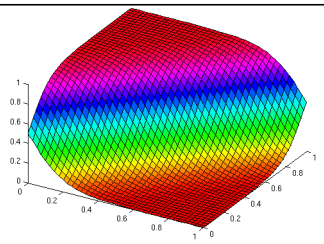
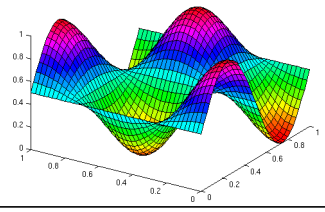
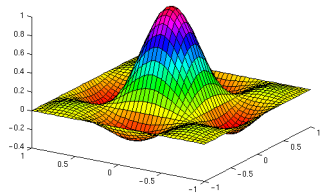
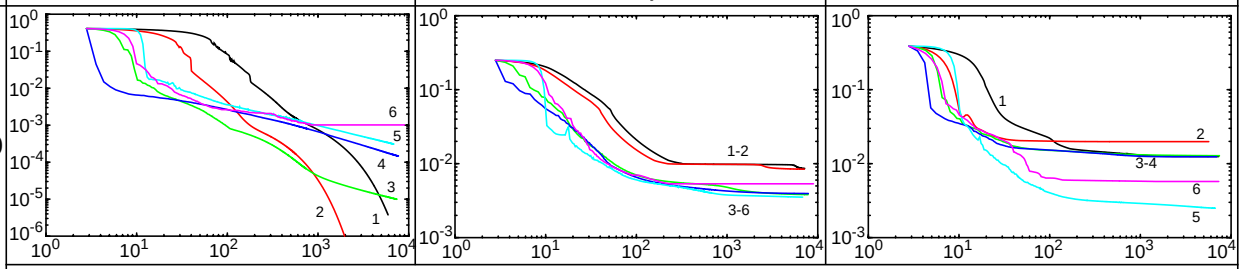
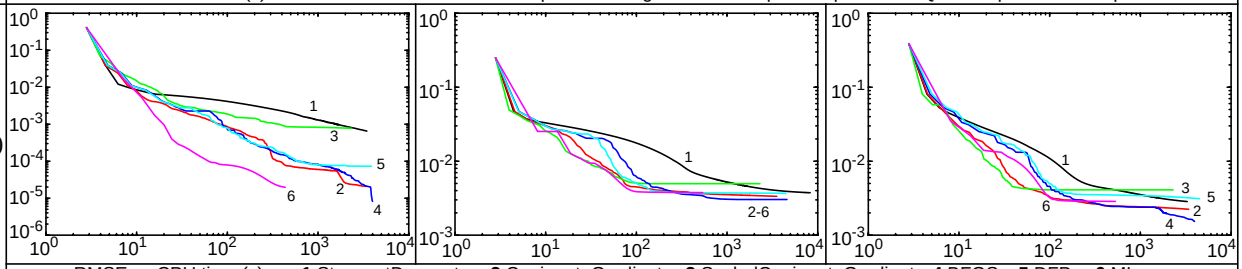
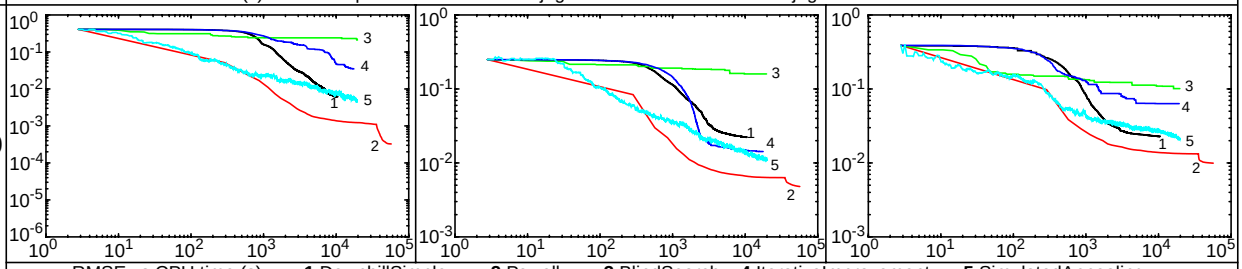
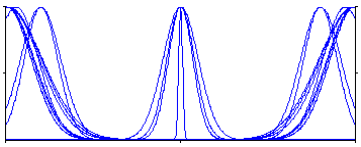
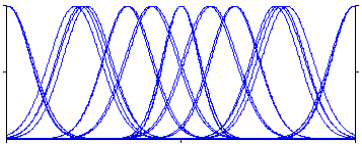
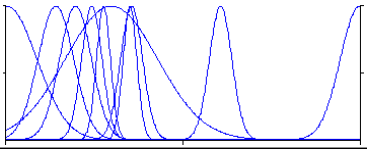
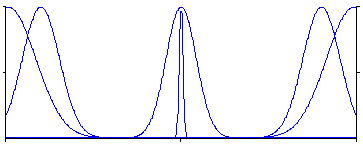
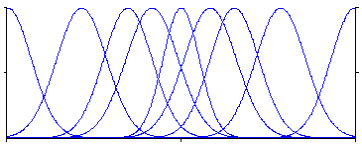
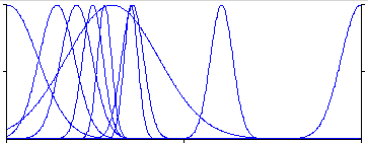
In these examples, the fuzzy system behavior depends linearly on the membership function parameters of the output variable, and non-linearly with respect to the membership function parameters of the input variables. This dependence leads the system's *Root Mean Square Error* (RMSE) to a fast decrease in the first steps of most learning algorithms (mainly due to the tuning of the output variable parameters), following with a slower reduction, when all the parameters, either lineal or non-linear, are being simultaneously modified.

The existence of non-linear parameters generates the presence of several local minima in the tuning process. This make the learning results for an specific algorithm to depend not only on the convergence speed of that algorithm towards a local minimum, but also on the quality of that minimum. 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 form 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 input variables are not significantly modified. On the other hand, the membership functions of the output variable result clearly moved and deformed, and tend to group around some common forms (Figure 3f). The tool offers a postprocessing algorithm to make a clustering over these functions, obtaining a reduced set of membership functions for the output variable (Figure 3g). This reduction leads to the simplification of the rule base of the fuzzy system (Figure 3h). Using the capabilities of the XFL3 language to describe complex antecedents in fuzzy rules, the rule base can be reduced from 49 rules to 9 rules (examples 1 and 2) or 10 rules (example 3).

5.- 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, the incorporated methods of clustering and pruning, and the capability of tuning complex systems make Xfsl a flexible and powerful tool. A relevant issue is that the tool is integrated into a fuzzy system development environment, which makes it possible to combine Xfsl with other tools for graphical representation, simulation or synthesis. As a future work, we are also interested in developing a similar tool focused on applying reinforcement learning.

	Example 1	Example 2	Example 3
(a)	$f(x, y) = \frac{1}{1 + e^{10(x-y)}}$	$f(x, y) = \frac{1}{2} \cdot (1 + \sin(2\pi x) \cdot \cos(2\pi y))$	$f(x, y) = \frac{\sin(2\pi x)}{2\pi x} \cdot \frac{\sin(2\pi y)}{2\pi y}$
(b)			
(c)	 RMSE vs CPU time (s): **1** BP **2** BPM **3** AdaptiveLearningRate **4** AdaptiveStepSize **5** QuickProp **6** RProp		
(d)	 RMSE vs CPU time (s): **1** SteepestDescent **2** ConjugateGradient **3** ScaledConjugateGradient **4** BFGS **5** DFP **6** ML		
(e)	 RMSE vs CPU time (s): **1** DownhillSimplex **2** Powell **3** BlindSearch **4** IterativeImprovement **5** SimulatedAnnealing		
(f)			
(g)			

(h)

	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

	y1	y2	y3	y4	y5	y6	y7
x1	z3	z4	z6	z7	z6	z4	z3
x2	z9	z8	z2	z1	z2	z8	z9
x3	z9	z8	z2	z1	z2	z8	z9
x4	z5	z5	z5	z5	z5	z5	z5
x5	z1	z2	z8	z9	z8	z2	z1
x6	z1	z2	z8	z9	z8	z2	z1
x7	z7	z6	z4	z3	z4	z6	z7

	y1	y2	y3	y4	y5	y6	y7
x1	z3	z2	z4	z7	z4	z2	z3
x2	z2	z5	z1	z0	z1	z5	z2
x3	z4	z1	z6	z8	z6	z1	z4
x4	z7	z0	z8	z9	z8	z0	z7
x5	z4	z1	z6	z8	z6	z1	z4
x6	z2	z5	z1	z0	z1	z5	z2
x7	z3	z2	z4	z7	z4	z2	z3

Figure 3: Application examples of function approximations.

6.- REFERENCES

[1] Battiti, R., First- and Second-Order Methods for Learning: Between Steepest Descent and Newton's Method, Neural Computation No. 4, pp. 141-166, 1992.

[2] Brent, R.P., Algorithms for Minimization Without derivatives, Prentice-Hall, 1973.

[3] Buckles, B. P., Petry, P. E., Eds., Genetic Algorithms, IEEE Computer Society Press, 1992.

[4] Fahlman, S., E., Faster-learning variations on back-propagation: an empirical study, in Proc. of the 1988 Connectionist Models Summer Schools, pp. 38-51, Morgan Kauffmann, 1988.

[5] Fletcher, R., Practical Methods of Optimization, John Wiley & Sons, Ltd., 1986.

[6] Geman, S., Geman, D., Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 6, pp. 721-741, 1984.

[7] Goldberg, D. E., Genetic Algorithms in Search, Optimization and Machine Learning, Addison Wesley, 1989.

[8] Jacobs, R., Increased rates of convergence through learning rate adaptation, Neural Networks, Vol. 1, N. 4, pp. 295-308, 1988.

[9] Jang, J-S. R., Sun, C-T., Functional equivalence between radial basis function networks and fuzzy inference systems, IEEE Trans. Neural Networks, Vol. 4, N. 1, pp. 156-159, 1993.

[10] Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., Optimization by simulated annealing, Science, Vol. 220, pp. 671-680, 1983.

[11] Mendoça, P. R. S., Caloba, L. P., New simulated annealing algorithms, in Proc. IEEE Int. Symposium on Circuits and Systems, pp. 1668-1671, Hong-Kong, 1997.

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

[13] Moreno Velo, F.J., Baturone, I., Sánchez Solano, S., Barriga, A., Xfuzzy 3.0: A Development Environment for Fuzzy Systems, 2nd International Conference in Fuzzy Logic and Technology (EUSFLAT'2001), Leicester, 2001, submitted to.

[14] Moreno, F.J., López, D.R., Barriga, A., Sánchez-Solano, S., XFBPA: A Tool for Automatic Fuzzy System Learning, Proc. 5th European Congress on Intelligent Techniques and Soft Computing (EUFIT'97), pp. 1084-1088, Aachen, Sep. 1997.

[15] Moreno, F.J., Sánchez-Solano, S., Barriga, A., Baturone, I., López, D.R., XFL3: A New Fuzzy System Specification Language, in Proc. 5th WSES/IEEE World Multiconference On Circuits, Systems, Communications & Computers (CSCC'2001), pp. 6791-6796, Rethymnon, 2001.

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

[17] Riedmiller, M., Braun, H., RPROP: A fast and robust backpropagation learning strategy, in Proc. 4th Australian Conference on Neural Networks, pp. 169-72, Sydney, 1993.

[18] Rumelhart, D. E., Hinton, G. E., Williams, R. J., Learning representations by back-propagation, Nature, Vol. 323, pp. 533-536, 1986.

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

[20] Szu, H., Hartley, R., Fast simulated annealing, Physics Letters A, Vol. 122, pp. 157-162, 1987.