AUTOMATIC TUNING OF COMPLEX FUZZY SYSTEMS WITH XFUZZY

F. J. Moreno-Velo\textsuperscript{1}, I. Baturone\textsuperscript{2}, A. Barriga\textsuperscript{2}, S. Sánchez-Solano\textsuperscript{2}

\textsuperscript{1} DIESIA - Escuela Politécnica Superior - Universidad de Huelva
Ctra. De Palos de la Frontera s/n, E21071, La Rábida-Huelva, Spain

\textsuperscript{2} Instituto de Microelectrónica de Sevilla - Centro Nacional de Microelectrónica - CSIC
Avda. Reina Mercedes s/n, (Edif. CICA), E-41012, Sevilla, Spain.

\textit{Fuzzy Sets and Systems (In Press)}

© 2007 ELSEVIER. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author’s copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder.
Automatic Tuning of Complex Fuzzy Systems with Xfuzzy

F.J. Moreno-Velo a, I. Baturone a, A. Barriga b, S. Sánchez-Solano b

a DIESIA - Escuela Politécnica Superior - Universidad de Huelva
Ctra. de Palos de la Frontera, s/n. E-21011 La Rábida-Huelva SPAIN

b Instituto de Microelectrónica de Sevilla (IMSE-CNM)
Avda. Reina Mercedes, s/n Edif. CICA. E-41012 Sevilla SPAIN

Abstract

Tuning a fuzzy system to meet a given set of requirements is usually a difficult task that involves many parameters. Since doing it manually is often cumbersome, several CAD tools have been reported to automate this process. The tool we have developed, xfsl, tries to reduce the limitations of other tools. In this sense, it includes a wide set of supervised learning algorithms and is able to cope with complex fuzzy systems. In particular, xfsl is able to adjust hierarchical fuzzy systems; systems that employ fuzzy functions defined freely by the user, like membership or connective functions, defuzzification methods, or even linguistic hedges; and fuzzy systems with continuous outputs (such as fuzzy controllers) as well as categorical outputs (such as fuzzy classifiers). Several examples included in this paper illustrate all these issues. Another relevant advantage is that xfsl is integrated into the fuzzy system development environment Xfuzzy 3, and, hence, it can be easily employed within the design flow of a fuzzy system.

Key words: Automatic Tuning, Supervised Learning, CAD Tools, Linguistic Interpretation.

1 Introduction

The behavior of a simple fuzzy system which contains only a rule base depends on both its structure (the number of rules, the number of fuzzy sets covering

Email address: francesco.moreno@diesia.ubu.es (F.J. Moreno-Velo).

Preprint submitted to Fuzzy Sets and Systems 11 November 2005
the input and output universes of discourse, etc.) and its parameters (those
which define the membership functions associated with the input and output
variables). Although the usual approach to design a simple fuzzy system is to
translate the knowledge of a human expert expressed linguistically, problems
can appear because the way to implement the translation is not unique and/or
the knowledge is not always available or possible to realize. This is why many
researchers have worked and are still working on applying automatic tuning
techniques to fuzzy systems.

If the tuning is applied only to the membership functions representing the
antecedents and consequents of the rules, the fuzzy system is usually called a
self-tuning system or a system with parameter learning. In a more general
case, if the rule base is also tuned, the system is usually called a self-organizing
fuzzy system or a system with structural learning.

The tuning techniques that have been used to adjust a fuzzy system can be
grouped into the following categories: (a) meta-level heuristic rules, which
implement the knowledge of an expert on tuning the system; (b) supervised
and nonsupervised algorithms taken from the neural network domain, which
require a set of numerical training data; (c) reinforcement learning, which
is applied when the only feedback about the system performance is a re-
ward/punishment signal; and (d) genetic algorithms, which try to improve
the system performance according to a set of objectives included within an
adequate fitness function. When input/output training data are available, su-
pervised learning algorithms usually provide the best results for tuning a fuzzy
system.

Most of supervised learning techniques have been proposed for simple fuzzy
systems, that is, systems with a unique rule base and particular membership
functions, connective operators and defuzzification methods. However, theo-
retical studies on fuzzy systems and their practical applications to solve more
challenging problems has resulted in the introduction of a large set of mem-
bership functions (triangular, trapezoidal, Gaussian, B-splines, etc.), connec-
tive operators (minimum, product, bounded sum, etc.), and defuzzification
methods (center of area, mean of maxima, etc.) suited for particular applica-
tions [21]; on the other hand it has brought about the use of linguistic
hedges (slightly, more or less, strongly, etc.) as well as certainty degrees (or
rule weights) to increase the expressiveness and linguistic interpretability of
the rules [25]; and it has led to the development of hierarchical systems (con-
taining modules connected in cascade, in parallel, or in a hybrid architecture,
each one applying its appropriate and usually different fuzzy operators, and
interchanging data which could be either fuzzy or non-fuzzy). As far as we
know, very few of the proposed tuning techniques have been applied to these
complex fuzzy systems and, in the same way, none of the CAD tools de-
veloped for automatic tuning is able to support them. The first commercial CAD
environments for designing fuzzy systems (such as CubiCalc from HyperLogic Corp., TILShell from Togai InfraLogic, fuzzyTech from INFORM, and FIDE from Aptronix) were restricted to applying a small number of learning algorithms and imposed hard constraints on the system they could tune: only a few kinds of membership functions can be used in the description of the system, and important issues like linguistic hedges and a hierarchically structured knowledge are not allowed. Something similar happens to the Fuzzy Logic Toolbox of Matlab from MathWorks, Inc. (which basically implements the neuro-fuzzy ANFIS model proposed in [9]) and to CAD environments developed in an academic context (such as NEFCCLASS and NEFCON from the University of Magdeburg [15] [14], which only include gradient-descent learning algorithms for tuning fuzzy systems with particular features).

The CAD tool for tuning fuzzy systems that is presented in this paper tries to reduce these limitations: it can apply many supervised learning algorithms and does not impose hard restrictions on the system which is being adjusted. It is integrated into the environment Xfuzzy 3 developed by our group and, hence, the adjusted fuzzy system can be described, identified, simplified, verified and synthesized with other tools of Xfuzzy 3 [12]. The tool has been named xfsl for being the XFuzzy tool for applying Supervised Learning. More information about Xfuzzy 3 can be found at its official Web page [27], from which it is distributed freely under the GNU General Public License.

This paper focuses on describing the capabilities of xfsl to apply automatic parameter learning techniques to fuzzy systems and how this facilitates greatly their design process. The structure of the paper is as follows. Section 2 starts by describing the complex and flexible fuzzy systems which can be tuned by xfsl and goes on by summarizing the wide set of error functions and supervised learning algorithms that have been included so as to support this flexibility. Section 3 includes several application examples (already used by other authors in the literature) related to approximation and classification problems. They illustrate how xfsl compared to other proposals in the literature offers a higher flexibility regarding error functions, learning algorithms and the complexity of the supported fuzzy system (hierarchical or not, with linguistic edges and any kind of fuzzy operators), thus obtaining better results. Finally, conclusions are given in Section 4.

2 The Xfsl tool

Xfsl is the tool of Xfuzzy 3 that allows the user to apply supervised learning algorithms to complex fuzzy systems. A formal specification language, named XFL3, has been developed to support this complexity [13]. An important characteristic of XFL3 is that the functional and logical structures of a fuzzy
system are described independently.

Regarding the logical structure, XFL3 allows defining hierarchical systems with fuzzy modules connected in series and/or in parallel, which can interchange crisp or fuzzy values. Each fuzzy module can use its own operator set for its rule base, thus allowing a fuzzy system to be formed by different kinds of fuzzy modules (controllers, classifiers, etc.). A rule base in XFL3 contains a set of logic rules relating the linguistic values of the input variables with those of the output variables. The antecedents of these rules, which define the relationships among the input linguistic variables, can be formed by any combination of fuzzy connectives (and, or) and linguistic hedges (greater or equal to, greater than, smaller or equal to, smaller than, not equal to, strongly, more or less, slightly) applied to those linguistic terms of the input variables. On the other side, each rule consequent describes the assignment of a linguistic value to an output variable. In addition, certainty degrees or weights can be assigned to the rules.

Regarding the functional structure of a fuzzy system, that is, the mathematical description of those functions used in the logical structure of the system, the designer can freely define them. The usual functions shown in Table 1 have been predefined in Xfuzzy 3. The definition of a function concerning tuning should include, apart from its name and the parameters that specify its behavior, the constraints on these parameters (and the method for updating them while maintaining their constraints) and the description of its differential function (if it is employed in gradient-based learning mechanisms). This means that xfsl can be applied to systems which employ particular functions defined by the user.

Table 1
Functions defined for fuzzy operators

<table>
<thead>
<tr>
<th>Function type</th>
<th>Possible assigned functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary (T-norms)</td>
<td>min, prod, bounded_prod, drastic_prod.</td>
</tr>
<tr>
<td>Binary (S-norms)</td>
<td>max, sum, bounded_sum, drastic_sum.</td>
</tr>
<tr>
<td>Binary (Implication)</td>
<td>dienes.rescher, mizumoto, lukasiewicz, dubois.prade, zadeh, goguen, godel, sharp.</td>
</tr>
<tr>
<td>Unary</td>
<td>not, sugeno, yager, pow, parabola.</td>
</tr>
<tr>
<td>Membership functions</td>
<td>trapezoid, triangle, isosceles, slope, bell, sigma, rectangle, singleton, parametric.</td>
</tr>
<tr>
<td>Defuzzification methods</td>
<td>CenterOfArea, FirstOfMaxima, LastOfMaxima, MeanOfMaxima, FuzzyMean, WeightedFuzzyMean, Quality, GammaQuality.</td>
</tr>
</tbody>
</table>

The Graphical User Interface of xfsl, which allows to configure the learning
process, is shown in Fig. 1. The first step in this configuration is to select a training file which contains the input/output data of the desired behavior (a test file, whose data are used to check the generalization of the learning, and a log file, which allows the learning evolution to be saved onto an external file, can also be selected). Subsequent steps are to select the learning algorithm and the error function (which measures the deviation of the fuzzy system behavior from the desired input/output pattern set). A relevant feature of xfs is that the user can select the system parameters to be tuned by using a graphical interface, as shown in Fig. 2.

![Graphical User Interface of xfs](image)

**Fig. 1. Graphical User Interface of xfs**

Other options which can be selected in the configuration area of xfs are two simplification processes to be applied before or after the tuning process. One of them consists in detecting and deleting those fuzzy rules and membership functions that are never activated sufficiently by any of the training input/output patterns. The other one is a clustering process over the membership functions covering the output variables, because a typical result of the tuning process is that these functions overlap each other to a high degree. This clustering is carried out automatically by means of the Hard C-means algorithm, where the number of clusters can be fixed manually, or may be selected by some cluster evaluation functions [2]. In this sense, xfs can be very useful not only for parametric but also for structural learning.

The last configuration step is to select the end condition to finish the learning process. This condition can be a limit imposed over the number of iterations, the maximum error goal, or the maximum absolute or relative deviation (considering either the training or the test error).
2.1 Supervised learning algorithms within Xfsl

Xfsl admits most of the supervised learning algorithms reported in the literature. Regarding gradient descent algorithms, it admits Steepest-Descent, Backpropagation, Backpropagation with Momentum, Adaptive Learning Rate, Adaptive Step Size, Manhattan, QuickProp and RProp. Since the convergence speed of BackPropagation is slow, several modifications were proposed such as using a different learning rate for each parameter, adapting the control variables of the algorithm heuristically [8], or taking into account the gradient value of two successive iterations [6] [19].

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 means of conjugate directions that do not interfere with each other. The conjugate gradient algorithms included in xfsl are the following: Polak-Ribiere, Fletcher-Reeves, Hestenes-Stiefel, One-step Secant and Scaled Conjugate Gradient [23] [11].

An effective step towards speeding up the convergence of learning algorithms is to make use of second-order information from the error function, that is, to make use 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. Among the algorithms based on this idea xfsl includes Broyden-Fletcher-Goldfarb-Shanno, Davidon-Fletcher-Powell[7], Gauss-Newton and Marquardt-Levenberg algorithms [1].
The computation of the system gradient, which is essential to the algorithms summarized above is done in xfsol by applying the chaining rule. The gradient of the error function cannot always be calculated because it can be too costly or not defined. In these cases, optimization algorithms without derivatives can be employed within xfsol by applying the Downhill Simplex algorithm [18] or Powell’s method [3]. These kinds of algorithms are much slower than the previous ones. An alternative to their use that xfsol offers is 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.

All the above mentioned algorithms do not reach the global but a local minimum of the error function. The statistical algorithms are more global because they generate different system configurations which spread the search space. The statistical algorithms included in xfsol are Blind Search and Simulated Annealing (with lineal, exponential, classic, fast, and adaptive annealingschemes) [10][26]. Statistical algorithms may provide good results when the number of parameters to be adjusted is low. When it is high, the convergence speed is so extremely slow that it is usually preferred to generate random configurations, to apply gradient-descent algorithms to them and finally to select the best solution.

An important issue when tuning fuzzy systems is that the parameters to be adjusted usually have to meet several constraints. For instance, when tuning the parameters of a Gaussian membership function, the learning algorithm 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: "$p_i \leq \text{constant}$", "$p_i < \text{constant}$", "$p_i \geq \text{constant}$", "$p_i > \text{constant}$", or "$p_i < p_j$". The latter ones, which appear, for instance, between the three points defining a triangular membership function, are the most difficult constraints to maintain and should be avoided as much as possible. In this sense, better results are obtained if triangular functions are defined by their centers, widths, and slopes.

Statistical algorithms manage constraints easily, by directly rejecting the random configuration generated if it does not meet the constraints. On the other hand, algorithms based on any kind of gradient-descent computation always generate the same (deterministic) displacement at a given iteration. Hence, the solution is not to reject a forbidden displacement (it would be generated again in the next iteration) but to change it into another one accepted by the system. The usual solution of trying a smaller displacement in the same direction, as shown in Fig. 3a, 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 Fig. 3a). This problem has been avoided in xfsol by managing the parameters independently. This means,
in the example of Fig. 3, that the system would be moved to the final point shown in Fig. 3b. For those constraints relating several parameters (like those of the triangle shown in Fig. 3c), the solution implemented in xfsl is to consider the parameters as particles subject to inelastic collisions in a one-dimensional space (Fig. 3d). Other approaches to constrained fuzzy system tuning can be found, for example, in [24].

![Fig. 3. Learning under constraints](image)

### 2.2 Error functions admitted in Xfsl

The adequate error function for a tuning process depends on the features of the system to tune (if its outputs are numerical or linguistic labels), the learning algorithm (if it needs or not derivatives) and the weight given for each output deviation. This is why seven error functions have been defined in xfsl.

For fuzzy systems with continuous output variables, xfsl includes the well-known mean square error (MSE):

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

(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 used to normalize the deviations.

Since the relative influence of every output variable on the global system deviation can be very useful, a weighted mean square error (WMSE) has been
also included:

\[ W \text{MSE} = \frac{1}{N} \cdot \sum_{i,j} w_j \cdot \left( \frac{y_{ij} - \tilde{y}_{ij}}{r_j} \right)^2 \]  

(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.

The absolute instead of the square error is minimized by \textit{xfsl} if the mean absolute error (MAE) or the weighted mean absolute error (WMAE) are selected.

The above expressions assume a numerical output from the fuzzy system. However, there are fuzzy systems whose outputs are linguistic labels, as occurs with classifiers. In these systems, the output value is the linguistic label presenting the highest activation degree as a result of the inference process. A usual definition for the behavior deviation of this kind of system is the number of classification errors (CE):

\[ CE = \frac{1}{N} \cdot \frac{1}{M} \cdot \sum_{i,j} \delta_{ij} \]  

(3)

where \( \delta_{ij} \) is 1 when the pattern classification has failed and 0 otherwise.

Since it is interesting to take into account the distance of the classification failures from the right classifications, an advanced classification error (ACE) has been also defined as follows:

\[ ACE = \frac{1}{N \cdot M + 1} \cdot \sum_{i,j} \delta_{ij} \]  

(4)

with

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

(5)

where \( \tilde{\alpha}_{ij} \) is the activation degree of the right label, and \( \alpha_{ij} \) 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, and, hence, can be used with gradient-descent algorithms:

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

where $\delta_{ij}$ is 0 when the pattern classification has failed and 1 otherwise.

3 Application examples

3.1 Tuning fuzzy systems to approximate functions

The simplest structure of a fuzzy system capable of approximating any static behavior consists of a single, complete and grid-based rule base, which contains all the possible combinations of the fuzzy sets representing the input variables. These are the fuzzy systems considered in this subsection to illustrate the use of xfsl.

Fig. 4a shows a nonlinear function which has been selected as an example of target behavior. Fig. 4b-d show the initial description of the fuzzy system to be tuned. The universes of discourse of the two input variables are covered by 7 homogeneously distributed bell-shaped membership functions. The complete knowledge base is formed by 49 rules and the output variable is described by 49 identical bell-shaped membership functions, each one being assigned to each rule. The defuzzification method employed to compute the non-fuzzy output is the Weighted Fuzzy Mean. Initially, the system generates a flat surface because all the output membership functions are equal (Fig. 4e).

Taking into account that the previously described initial fuzzy system is differentiable, the most adequate learning algorithms to tune it are those based on the gradient, in particular, the second-order algorithms, which are the fastest ones. The results included in the following have been obtained by using Marquardt-Levenberg’s algorithm and have considered that all the membership function parameters are adjustable, which amounts to 126 parameters.

Fig. 5 shows the results of tuning. The membership functions of the input variables have been modified to have an overlapping degree of 3, although they do not cover the two input universes of discourse equally because the target behavior is not symmetrical in $x$ and $y$. Regarding the output variable, there is a trend of its membership functions to cluster. In this case, the clustering process reduces their number to 9. The RMSE of the system prior to the clustering was 0.4%. The clustering process increases this error to 0.8% but
a posterior tuning makes it possible to obtain again an RMSE of 0.4%. The description can be further simplified by using the simplification tool of Xfuzzy.

The results obtained by xfs1 can improve others reported in literature. For example, the target function considered above has also been employed in [20] to test a rule identification algorithm. Using triangular membership functions for the inputs, a 5x5 rule base, and the Fuzzy Mean defuzzification method, the algorithm proposed in that work obtains a final RMSE of 13.0%. Another identification algorithm proposed in [17] obtains a RMSE of 3.0% when approximating the same function with a similar fuzzy system. Applying Marquardt-Levenberg’s algorithm with xfs1, the RMSE obtained was 2.2%. The resulting approximation error values are different in the three cases because the constraints on the fuzzy system tuned are also different. The number of output linguistic labels in [20] is limited to 5. In our case, it has not been limited a priori but xfs1 has performed an automatic clustering which has grouped them into 7 labels. On the other hand, the overlapping degree of the input membership functions in [17] is restricted to 2, while it is free in our case, thus allowing for a better approximation. Anyway, the same constraints can be imposed with xfs1.

3.2 Tuning a hierarchical fuzzy system

A relevant advantage of xfs1 is its ability to adjust hierarchical systems. To illustrate this type of learning process, let us consider the target behavior in Fig. 6a. We question if a hierarchical fuzzy system with two cascaded rule bases, like that shown in Fig. 6b, will be able to learn the intrinsic composition of our target behavior. The initial description we have taken for the first rule

![Initial fuzzy system to be tuned](image)

Fig. 4. Initial fuzzy system to be tuned: (a) target function; membership functions of the input (b) and output (c) variables; (d) rule base; (e) input-output surface.
Fig. 5. Results after tuning: membership functions of variable $x$ (a) and $y$ (b); output membership functions before (c) and after (d) clustering; (e) rule base.

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 great deal of local minima which may make the application of gradient-based learning algorithms useless. Contrary to the example of the previous subsection, the statistical algorithms provide now better results when tuning the fuzzy system described in Fig. 6. In particular, we have used the Blind Search algorithm followed by Marquardt-Levenberg’s algorithm (to find the corresponding local minimum rapidly). In the latter algorithm, $xfsl$ does not compute the derivatives (it is
not possible in hierarchical systems) but estimates them from small parameter modifications.

Fig. 7 shows the results obtained after learning. The global system behavior approximates the target behavior with an RMSE of 0.41% (Fig. 7a). What is very interesting is that the rule bases have learnt the intrinsic composition of the target function. The first rule base identifies a subtractive relation between \( y \) and \( x \) (with a certain scaling factor), while the second rule base identifies the step-wise relation. The bump that appears in Fig. 7c at small values of \( w \) does not have any influence because those values are never generated by the ranges considered for \( x \) and \( y \).

Fig. 7. Results after tuning: (a) global input-output surface; behavior of rule bases base1 (b) and base2 (c).

Let us compare these results with the approximation degree reached by non-hierarchical fuzzy systems (Fig. 8). Tuning a non-hierarchical fuzzy system with 9 rules (3 fuzzy sets per each input), the RMSE achieved is 9.4%. Increasing the number of rules to 25, the RMSE is 2.7%; with 49 rules, the RMSE is 0.63%; and with 64 rules the RMSE decreases to 0.23%. This means that between 49 and 64 rules are required by a grid-based fuzzy system to perform as a hierarchical system with only 7 rules, thus showing the advantages of using hierarchical descriptions.

Fig. 8. Results using fuzzy systems with a grid structure: (a) 9; (b) 25; (c) 49 rules.

The capability of xfsi to tune hierarchical fuzzy systems allows for a further improvement of the results reported in literature for certain approximation problems. Let us consider, for example, the two-dimensional function \( f(x,y) = \sin(x \ast y) \) employed in [4] and [17]. Using normalized triangular membership functions for the inputs, a 8x8 non-hierarchical rule base, and
the Fuzzy Mean defuzzification method, the algorithm proposed in [17] obtains a final RMSE of 2.67% after learning with 400 input/output training data. Applying Marquardt-Levenberg’s algorithm with xfs1 to the same fuzzy system (using the same training data number) but without imposing the constraint of normalized input functions, the RMSE obtained was 1.13%. If xfs1 is applied to a hierarchical fuzzy system, the results are much better. Let us consider, for example, a hierarchical system similar to that of the previous example, that is, composed of two rule bases in cascade which employ the Fuzzy Mean as defuzzification method. The first rule base uses two triangular fuzzy sets for each input variable and four singleton values for the output, thus resulting in 4 rules. The second rule base uses three bell-shaped membership functions for the input and three singleton values for the output, thus containing 3 rules. Marquardt-Levenberg’s algorithm has been applied again with xfs1 but now estimating the derivatives from parameter perturbations (since the hierarchical system is not differentiable). The obtained RMSE, after learning is only of 0.66% in just 13 iterations. Hence, we have designed a fuzzy system which, with only 7 rules, performs twice or 4 times better (in terms of approximation error) than a fuzzy system with 64 rules.

3.3 Tuning a fuzzy classifier

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. 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.

We have selected the well-known IRIS data base as a classification problem. It contains 150 measurements corresponding to 2 features of 3 types of Iris plants: Setosa, Versicolour, and Virginica. The features represent the petal lengths and widths. The initial rule base of the fuzzy classifier to be tuned is formed by three rules, one per each plant type. The rules have been selected (from the set generated by a grid partitioning of the input variables) as those with the greatest efficiency [15] (Fig. 9a-b).

The classification square error has been selected in this example as the error to minimize through the learning process. Since it is a differentiable function, gradient-based algorithms can be employed. The membership functions of the input variables have been learnt by using xfs1 with the RProp algorithm (Fig. 9c-d), giving a success rate of 96.7% (only 5 samples are misclassified). Although this rate is high, the linguistic interpretation of the fuzzy rules is not clear enough because the fuzzy set “medium” overlaps a great deal with the
"large" or "small" sets.

Fig. 9. Fuzzy classifier without linguistic hedges: (a) rule base; membership functions of the variables length and width before (b) and after learning (c)(d).

Taking advantage of the linguistic hedges admitted by the language XFL3, we have defined a fuzzy classifier with an easy linguistic interpretation (Fig. 10). Its rule base uses only the fuzzy set “medium” because the concept of “large” is understood as “greater than medium”, and “small” as “smaller than medium”. This ensures that the three linguistic concepts of “medium”, “large”, and “small” always overlap by 50%. The tuning of this fuzzy classifier has to meet more constraints than the previous one, thus resulting in a classifier whose success rate is usually worse. In this example, however, the constraints help the gradient-based algorithm find a better local minimum, improving the success rate to 97.3% (concerning the training data). This way, we have obtained an efficient system with linguistic interpretability.

Some other authors have used different kinds of fuzzy classifiers on the IRIS data base, obtaining success rates of 96.7% or 97.3% (5 or 4 misclassified data) [16][22]. A relevant feature of xfsl is that the latter result is obtained with only three rules whose membership functions maintain their linguistic meaning throughout the learning process due to the use of linguistic hedges.

The Wisconsin Breast Cancer database is also a well-known classification benchmark. This data set contains a collection of 683 tumors described each by 9 features and grouped into 2 classes: benign and malignant. We have used xfsl in order to tune a fuzzy system with 3 labels per variable and 4 rules (2 per class), which have been selected as the most efficient rules by [15]. The classification square error and the RProp algorithm have been selected to configure the learning process. Using xfsl, the training process improves the success rate from 95.0% to 97.2% (19 misclassified data). Some other proposals with a success rate of 98% or better can be found, but they are based on fuzzy systems with a higher number of rules, such as [28] (35 rules) or [5] (103 rules).
Fig. 10. Fuzzy classifier with linguistic hedges: (a) rule base; membership functions of the variables \textit{length} and \textit{width} before (b) and after learning (c)(d).

4 Conclusions

The \textit{xfsl} tool presented here represents an important effort towards the automatization of the learning process in the design of fuzzy systems. It offers a wide set of algorithms (from gradient-based to statistical) to tune the parameters of the fuzzy system by minimizing different error functions (from weighted mean square error to square classification error). The user is responsible for selecting the adequate algorithm and error function for the system and the application.

\textit{Xfsl} performs different methods to compute or estimate the gradient as well as to maintain the constraints of the parameters of the system. In addition to the supervised algorithms, the simplification processes offered by \textit{xfsl} are very helpful for optimizing the structure of the fuzzy system under design since they can reduce the number of rules and membership functions. Moreover, the capability of \textit{xfsl} to tune 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. Besides, \textit{xfsl} can adjust any system described by the language XFL3, which allows for simple knowledge base specifications thanks to the use of expressive rules. In particular, \textit{xfsl} can adjust systems that employ linguistic hedges, which is very interesting for maintaining the linguistic meaning of a given rule base or for extracting linguistic knowledge from a set of numerical data.

References


17


