NeuroFAST: On-line Neuro-Fuzzy ART-Based Structure and Parameter Learning TSK Model

Spyros G. Tzafestas, Fellow, IEEE, and Konstantinos C. Zikidis, Student Member, IEEE.

Abstract—NeuroFAST is an on-line fuzzy modeling learning algorithm, featuring high function approximation accuracy and fast convergence. It is based on a 1st-order Takagi-Sugeno-Kang (TSK) model, where the consequence part of each fuzzy rule is a linear equation. Structure identification is performed by a fuzzy Adaptive Resonance Theory (ART)-like mechanism, assisted by fuzzy rule splitting and adding procedures. The well known δ rule continuously performs parameter identification on both premise and consequence parameters. Simulation results indicate the potential of the algorithm. It is worth noting that NeuroFAST achieves a remarkable performance in the Box and Jenkins gas furnace process, outperforming all previous approaches compared.

Index terms—TSK fuzzy reasoning model, fuzzy ART learning, structure/parameter identification, δ rule.

I. INTRODUCTION

Fuzzy set theory [1] was initially proposed as a tool for the expression and manipulation of human-like, expert knowledge. Combined with the learning ability of artificial neural networks, it was proved to be a powerful mathematical construct, enabling the symbolic expression of machine learning results. In the last few years, the application of neuro-fuzzy methods to nonlinear process identification using input-output data is a very active area. A comprehensive survey can be found in [2], with a plethora of references. One of the most influential fuzzy reasoning models was proposed by Takagi and Sugeno in [4]. In this model, the consequent part of each fuzzy rule is expressed as a linear function of the input variables, instead of a fuzzy set [3], reducing the number of required fuzzy rules. Since then, Sugeno and his colleagues established what is called today the Takagi-Sugeno-Kang (TSK) model [5][6].

Fuzzy modeling involves structure and parameter identification. The second is usually (and easily) addressed by some gradient descent variant, e.g., the least squares algorithm or back-propagation. Structure identification is a more difficult task, often tackled by off-line, trial-and-error approaches, like the unbiasness criterion [5][7]. One of the most common methods for structure initialization is uniform partitioning of each input variable range into fuzzy sets, resulting to a fuzzy grid. This approach is followed in ANFIS, a well-known TSK model learning algorithm [8]. In [10][11][12][20] the TSK model was used for designing various neurofuzzy controllers.

In the proposed approach, the human way of thinking is exploited as much as possible, and is incorporated into an automated procedure. NeuroFAST is a fuzzy modeling learning algorithm based on the TSK model and features on-line structure and parameter identification, very good numerical accuracy and fast convergence, for use in supervised, real time function approximation tasks. The input space is automatically partitioned into fuzzy subsets, using a modified fuzzy ART (Adaptive Resonance Theory) [19] mechanism. Fuzzy rules that tend to give high output error are split in two, by a specific fuzzy rule splitting procedure, resulting in a fuzzy k-d tree structure. To cope with ‘hard’ regions of extreme nonlinearities, a new fuzzy rule is created wherever the output error exceeds a dynamic threshold. At the same time, all adaptive parameters are tuned by the δ rule. The proposed algorithm is an extension to [18], where a fuzzy ARTMAP [25] module was employed. A more technically detailed presentation of the proposed architecture appears in [9].

The performance of NeuroFAST is verified by two examples: the fuzzy modeling of a nonlinear function and the prediction of the Box and Jenkins gas furnace process [21], a famous benchmark for system identification algorithms, where NeuroFAST exhibits outstanding performance over other existing methods.

II. PRESENTATION OF NEUROFAST

It is assumed that the input and output variables are known. In this study, issues concerning the choice of input variables from all possible variables are not dealt with. For discussions on this subject the reader is referred to [5][13][16] and [32].

A. General description and basic concepts

The core of the present system is the TSK model [4], namely a set of IF...THEN rules with fuzzy implications and 1st-order functional consequence parts, which was proved to be a universal approximator [26]. The format of the fuzzy rule $R_i$ is:
The number of rules is determined by the user, depending on the task and the available or expected training data. The algorithm creates linear models that approximate locally the function-to-be-learned. Structure identification sets a coarse fuzzy partitioning of the domain, while parameter identification optimally adjusts premise and consequent parameters.

A modified fuzzy ART mechanism [19] is employed for domain partitioning. The idea of utilizing fuzzy ART concepts for structure learning was introduced by Lin et al. [22][23][24]. Fuzzy ART is an unsupervised algorithm, which receives a stream of input patterns and automatically creates recognition categories or hyperboxes. These recognition categories start as points in the input space and increase in size to incorporate new points that are presented, until the whole input space is covered. The maximum size of these hyperboxes and implicitly the number of the required hyperboxes can be adjusted by a parameter called "vigilance". In our case, the input stream is formed by the input variable vector at every time step. The (crisp) hyperboxes are fuzzified, providing the implications of the fuzzy rules.

Fuzzy ART does not allow for any dependence on the output error, and results into a more-or-less uniform hyperbox allocation. This is not desired, because the function-to-be-learned is not assumed to exhibit uniform 'difficulty' in its whole domain. To help the algorithm learn better in 'hard' areas, a fuzzy rule splitting technique is employed: periodically, all rules are examined and the rule with the worst local performance is divided into two hyperboxes, after a guillotine cut across one dimension. This results to a fuzzy k-d tree structure. The question that arises is across which dimension the cut should be made. Various strategies have been proposed, including the balanced sampling criterion [15], direct evaluation [5] and regional linearity [15], employed in the proposed approach. The fuzzy rule splitting procedure adds to the algorithm computational complexity and overhead; this is the price for on-line structure identification.

Apart from fuzzy rule splitting, another procedure is fuzzy rule adding: a new rule is created wherever the output error exceeds a high threshold.

After all available rules have been used, the rule splitting and adding procedures are terminated. However, fuzzy ART learning remains active as a 'watchdog', preventing uncovering of the input space. The δ rule, which is active from the beginning, continues and fine tunes all parameters, until the algorithm reaches some stopping criterion.

The presentation of the algorithm will be made through a multi-input single-output example. A certain familiarity with the fuzzy ART model [19] is assumed.

B. Adaptive parameters

The adaptive parameters are: the weights associated with the membership functions (which will be called input weights), the slopes of the membership functions, and the weights of the consequence parts of the rules (which will be called output weights). At the beginning, there are \( N \) uncommitted or available nodes (each node corresponds to a hyperbox, a fuzzy subset, and a fuzzy rule). Suppose that there are \( M \) input variables. Let:

\[
\mathbf{w}_i = (w_{i1}, \ldots, w_{iM}, v_i, 1, \ldots, v_{2M})
\]

be the input weights defining the \((i)th\) hyperbox. Note that: \(v_j = 1 - v_j\) (complement coding form [19]).

As input vectors are presented, hyperboxes are created (which means that nodes become committed) and expand to cover the input space. These hyperboxes are fuzzified, forming the implications of the fuzzy rules. The fuzzy implication of the \((i)th\) rule is defined by \(\mathbf{w}_i\) and \(\mathbf{f}_i\), where \(\mathbf{f}_i\) is the vector of the slopes of the associated membership functions:

\[
\mathbf{f}_i = (f_{i1}, \ldots, f_{iM}, f_{iM+1}, \ldots, f_{2M}) \in \mathbb{R}^{2M}
\]

Details on the membership functions will be given in the simulation section. The consequence parts of the fuzzy rules are linear equations of the input variables (with a bias term), as mentioned above. Let:

\[
c_i = (c_{i0}, c_{i1}, \ldots, c_{iM}) \in \mathbb{R}^M, \quad i = 1, \ldots, N
\]

be the output weights of the \((i)th\) fuzzy rule.

C. Learning Parameters and Performance Indices

The ART learning parameters are the choice \(\alpha\), vigilance \(\rho\), and learning rate \(\beta\). The learning rate parameters for the \(\delta\) rule are \(\delta_1\), \(\delta_2\), and \(\delta_3\), and are associated with the updating of \(\mathbf{c}\), \(\mathbf{w}\), and \(\mathbf{f}\), respectively. All \(\delta\) rule learning rates decrease slowly with time. Another parameter is \(P\), which adjusts how often the check for the 'worst' fuzzy rule takes place.

Initialization of these parameters with some standard values is expected to provide acceptable performance. However, optimal setting is not a trivial task and usually requires some trial-and-error testing.

The global performance index is the Mean Square Error (MSE). A number of local performance indices are kept for each fuzzy rule, utilized by the rule splitting procedure.

D. Main Body of the Algorithm

Let \(x = (x_1, \ldots, x_M)\) be the input vector at a given time step and \(y\) the associated desired output value. All these variables must be normalized to [0,1].

1) Calculation of node activation:

The first step is to calculate the activation of each committed node for this input vector. Let \(M_i(x)\) be the fuzzy implication membership function value or firing strength of the \((i)th\) rule. The rules/nodes whose firing
strength $M_i(x)$ is higher than a small fixed threshold, e.g., 0.001, are activated and take part in output calculation and weight updating. All other nodes remain idle and are ignored. The sum of all rule firing strengths is defined as:

$$S(x) = \sum_{i=0}^{\text{all activated rules}} M_i(x)$$

If $S(x)$ is zero (which means that there are no active rules), the algorithm imperatively performs fuzzy ART learning (hyperbox expansion/creation) to incorporate this new input vector. In order to prevent possible uncovering of input space, fuzzy ART learning is called for even if $S(x)$ is below a certain level (set to 0.1).

2) Fuzzy ART learning:

During the first learning stage or whenever the sum of the rule firing strengths is low, the fuzzy ART mechanism operates as follows [19]:

The choice function of every committed node is calculated: $|x \land w_i| [x + w_i]$. The node with the highest choice value is selected. If node $i$ is selected, the match function $|x \land w_i| [|x|]$ is calculated. This value is compared to the vigilance $\rho$. If $|x \land w_i| / |x| < \rho$, mismatch reset occurs, and a new search starts. If no committed node satisfies the vigilance criterion (becomes resonant), an uncommitted one is chosen and initialized. If a committed node, e.g., the $(i)$ th node, is chosen and satisfies the vigilance criterion, then input weight updating is made:

$$w_i^{new} = \beta(|x \land w_i| + (1-\beta) w_i^{old})$$

3) Re-calculation of node activation:

If fuzzy ART learning was performed, the node activations $M_i(x)$ are re-calculated. If $S(x)$ is non-zero, the algorithm carries on to the following steps. Otherwise, it stops and proceeds to the next input, a case not so unusual during the first learning stage.

4) Output calculation:

The output value is calculated for each active rule:

$$y_i = c_0 + c_1 x_1 + ... + c_M x_M$$

where $x_1, ..., x_M$ are the input variables after a simple linear trasformation, intended to intensify their small variations inside each fuzzy set. The global output value is the weighted average of the output values of the activated fuzzy rules:

$$\text{output} = \frac{\sum_{i=0}^{\text{all activated rules}} M_i(x) \cdot y_i}{\sum_{i=0}^{\text{all activated rules}} M_i(x)}$$

provided that:

$$S(x) = \sum_{i=0}^{\text{all activated rules}} M_i(x) > 0$$

5) Training error and performance index:

The training error is the desired output value minus the actual output:

$$\text{error} = y - \text{output}$$

$MSE$ is updated as follows:

$$MSE^{new} = 0.9995 \cdot MSE^{old} + 0.0005 \cdot \text{error}^2$$

This is a convenient way of storing and updating $MSE$. The factor 0.9995 may depend on the task.

6) Weight updating:

Only the activated fuzzy rules take part in the weight updating procedure. First, the input weights and the slopes of the membership functions are updated, according to the $\delta$ rule. The updating equations for one membership function are derived in the Appendix, while analytical details can be found in [9]. The output weights are also updated according to the $\delta$ rule (in this case equivalent to the LMS algorithm):

$$c_j(t+1) = c_j(t) + h_1 \frac{M_j(x)}{S(x)} \cdot \text{error} \cdot x_j,$$

where $i=0,...,N$, $j=0,...,M$.

For every rule, there are $M$ possible pairs of fuzzy rules that could result after a potential cut. All the possible consequence parts are trained, in order to obtain tentative performance indices. For the $(i)$ th rule, there are $M \times 2 \times (M+1)$ tentative weights: $M$ possible cuts by two resulting rules by $M+1$ output weights for each rule. These weights are updated in the same manner as the output weights. Finally, the local tentative performance indices and counter are updated.

7) Rule splitting procedure:

Every $P$ time steps, all committed nodes are checked and the one with the highest local mean square error (worst performance) is split, provided that it has been activated at least $P$ times. The best possible cut is determined by the highest tentative performance index. The output weight vectors $c_1$ and $c_k$ of the corresponding fuzzy rules are initialized with the values of the vector $c_1$ before the cut. Finally, all relevant local performance indices and counters are reset.

8) Rule adding:

If the mean square error exceeds a threshold equal to 10$\cdot$MSE, an uncommitted node is used, and is initialized. This is actually a safety feature, originating from the fuzzy ARTMAP algorithm [25][18], having little or no effect in most of the cases.

III. SIMULATION RESULTS

A. Membership functions and inference methods

Five membership functions will be used, combined either with Mamdani’s or Larsen's inference methods. The first two membership functions are simple, piecewise linear functions, which use quantities already calculated in the fuzzy ART module. The output of

| TABLE I. MEMBERSHIP FUNCTIONS AND FUZZY IMPLICATIONS |
Membership function corresponding to the \((i)\) th rule of the \((j)\) th variable of \(\mu_{ji}(x_j) = \max(0,1-f_{ij} \cdot (u_j-x_j)),\) if \((x_j < u_j)\)
\[
\begin{align*}
\max(0,1-f_{ij} \cdot (u_j-x_j)), & \quad \text{if } x_j < u_j \\
1, & \quad \text{otherwise}
\end{align*}
\]

Fuzzy implication membership function of the \((i)\) th rule
\[
\tilde{M}_i(x) = \max(0,1-f_i + f_i \cdot x)
\]

2) Second membership function:
This is a trapezoidal function and is used with both min and product inference. The range of this function is also \([0,1]\). This membership function has similar computational requirements (in terms of computer time) with the first one, but offers better results, since there are more degrees of freedom.

3) Third membership function:
It is the double logistic function, i.e., two logistic functions joined together forming a bell-like curve. The two logistic functions are allowed to have different slopes. In the case of different slopes, the joining point \(x_{join}\) is the point where the two logistic functions have equal values. A drawback of this membership function is that it is not normalized, since the logistic function never reaches unity or zero.

4) Fourth membership function:
In an attempt to normalize the double logistic membership function, each logistic function is divided by its value at the joining point. In this way, both logistic functions attain the value of 1 at this point. The joining point \(x_{join}\) is on the middle between the input weights \(u_j\) and \(v_j\) defining the category \(\text{mean}_j\).

5) Fifth membership function:
If a category is relatively 'small', the corresponding membership function should desirably attain relatively 'higher' values, in order to have a considerable contribution to the global output value and therefore efficiently learn the function behavior in its area. Following this idea, the fifth function is not exactly a...
The comparison results appear in Table II. NeuroFAST performs relatively very well, using only two rules. The task is treated as if it were on-line, while most of the previous approaches used off-line, trial-and-error methods. However, it is noted that the piecewise linear nature of the first three fuzzy implication membership functions does not fit well to this task and sometimes prevents the algorithm from reaching a satisfactory performance index.

C. Box and Jenkins Gas Furnace Process Modeling

This is a common benchmark for testing system identification techniques. The data are from a furnace, where air and methane are combined. The input feed rate of methane and the concentration of CO2 in the output gases are sampled, giving 296 data pairs, which can be found in [21]. This is a dynamical process with one input \( x(t) \) and one output \( y(t) \). The aim is to predict current output \( y(t) \) using past input and output values, with the lowest mean square error.

As in some of the previous approaches, the following six input variables were used: \( x(t-1), x(t-2), x(t-3), y(t-1), y(t-2), y(t-3) \).

In Fig. 1 is the mean square error obtained versus the number of fuzzy rules, using the fuzzy implication membership function \( 3M(x) \). Comparison results with previous approaches are provided in Table III. It is noted that NeuroFAST with 20 rules attains the best performance reported up to now. It is also worth noting that even with one rule (6-input linear system) it outperforms many approaches with more rules.

![Fig. 1. The mean square error (MSE) of the proposed algorithm applied to the Box and Jenkins gas furnace process prediction versus the number of fuzzy rules, using the fuzzy implication \( 3M(x) \). Each run lasted approximately 50000 epochs. All learning parameters are kept fixed, except for the vigilance \( \rho \), which increases by the number of fuzzy rules: \( \alpha = 0.01, \beta = 0.1, \rho \in [0.0, 0.85], lr_1 = 0.5, lr_2 = 0.1, lr_3 = 100 \).](image)

<table>
<thead>
<tr>
<th>Model</th>
<th>Number of rules</th>
<th>Identification data</th>
<th>Prediction data</th>
<th>50 data randomly selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Linear model</td>
<td>-</td>
<td>12.7</td>
<td>11.11</td>
<td>-</td>
</tr>
<tr>
<td>2 FMM [5]</td>
<td>3</td>
<td>1.5</td>
<td>2.1</td>
<td>4.2</td>
</tr>
<tr>
<td>3 SOFIA [7]</td>
<td>4</td>
<td>1.8</td>
<td>2.9</td>
<td>3.4</td>
</tr>
<tr>
<td>4 FNN Type I [14]</td>
<td>8</td>
<td>0.84</td>
<td>1.22</td>
<td>-</td>
</tr>
<tr>
<td>5 FNN Type II [14]</td>
<td>4</td>
<td>0.73</td>
<td>1.28</td>
<td>-</td>
</tr>
<tr>
<td>6 FNN Type III [14]</td>
<td>8</td>
<td>0.63</td>
<td>1.25</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Proposed algorithm</th>
<th>Membership function</th>
<th>Number of rules</th>
<th>Identification data</th>
<th>Prediction data</th>
<th>Data from the whole domain</th>
<th>Percentage of successful runs</th>
<th>Stopping criterion</th>
<th>( f ) ( Ir_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 ( 1M(x) )</td>
<td>2</td>
<td>1.22</td>
<td>2.62</td>
<td>4.33</td>
<td>50%</td>
<td>150000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8 ( 2M(x) )</td>
<td>2</td>
<td>1.47</td>
<td>3.24</td>
<td>3.18</td>
<td>65%</td>
<td>100000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9 ( 3M(x) )</td>
<td>2</td>
<td>0.76</td>
<td>1.31</td>
<td>2.53</td>
<td>54%</td>
<td>100000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10 ( 4M(x) )</td>
<td>2</td>
<td>0.90</td>
<td>1.45</td>
<td>3.77</td>
<td>100%</td>
<td>30000</td>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>11 ( 5M(x) )</td>
<td>2</td>
<td>0.79</td>
<td>1.84</td>
<td>6.91</td>
<td>100%</td>
<td>30000</td>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>12 ( 6M(x) )</td>
<td>2</td>
<td>0.71</td>
<td>1.36</td>
<td>3.03</td>
<td>100%</td>
<td>30000</td>
<td>2</td>
<td>100</td>
</tr>
</tbody>
</table>

| \( N=2, M=3 \). Averaged over 100 runs. \( P = 5000, d = 1.7, \alpha = 0.01, \beta = 0.1, \rho = 0, lr_1 = 0.1, lr_m = 0.0, lr_3 = 0.1 \). |
NeuroFAST 6 20 0.00001
NeuroFAST 6 15 0.00040
NeuroFAST 6 1 0.06544
Kim and Kasabov [33] 2 15 0.00042
ANFIS [8] taken from [33] 2 25 0.00073
Tzafestas and Zikidis [9] 6 2 0.049
Lin and Cunningham [16] 5 4 0.071
Zikidis and Vasilakos [31] 6 2 0.064
Wang and Langari [30] 6 2 0.066
Sugeno and Tanaka [6] 6 2 0.068
Xu [29] 2 25 0.328
Tong [27] 2 19 0.469
Box and Jenkins [21] 6 - 0.202

IV. CONCLUSION
A new method was proposed for on-line structure and parameter learning of a functional reasoning fuzzy system. Structure identification is executed by a fuzzy ART module. Specific fuzzy rule splitting and adding procedures, provide better coverage of ‘difficult’ areas of the input space. Premise and consequent parameters are fine tuned by the use of the δ rule. Simulation results demonstrate the remarkable capabilities of the proposed method.

Future work will be dealing with a meta-learning scheme for automatic adjustment of the learning parameters employed in this algorithm.

APPENDIX
Consider the cost function:
$$E(W) = \frac{1}{2}(y - \text{output})^2 = \frac{1}{2}\text{error}^2,$$
where y and output are the current desired and actual output of the system and W is a generalized vector containing all free parameters of the learning process (in our case it should contain w, c, and f). The aim is to iteratively minimize the cost function E(W) over the whole input space. According to the δ rule, in order to perform gradient descent, the change to each parameter $W_k$ should be proportional to the negative of the gradient of E(W) with respect to $W_k$:
$$\Delta W = -lr \cdot \nabla_w E(W) \quad \text{or} \quad \Delta W_k = -lr \frac{\partial E(W)}{\partial W_k}$$
where lr is the learning rate.

Using this rule, deriving the updating equations of the output weights is relatively easy. However, the updating equations of the premise parameters (input weights w and slope values f) are more complex and depend on the membership functions and the inference method. The study of the input weights updating equation will be made using the fuzzy implication $\mu_M(x)$, which is a characteristic example and provides flexibility (many adaptive parameters, i.e., degrees of freedom) and low computational overhead (no non-linear calculations).

For each variable $j$, the weights $w_j$ and $w_{M+j}$ and the slopes $f_j$ and $f_{M+j}$ define the membership function associated with this variable. All these parameters should be positive. Therefore, $w_j \leq 1 - w_{M+j}$.

Therefore, the δ rule should not be allowed to make an update that would violate any of these two requirements. Furthermore, for this fuzzy implication, as well as for $\mu_M(x)$ and $\mu_{M+j}$, the premise parameters are changed only if the current input vector x does not lie in the associated hyperbox, since inside the hyperbox these functions are constant. Hence, updating takes place only if $|x \cdot w_j| < |w_j|$.

Considering $w_j$, we use the chain rule:
$$\frac{\partial E}{\partial w_j} = (y - \text{output}) \cdot (1) = -\text{error}$$
As mentioned earlier, the output of the algorithm is:
$$\text{output} = \sum_{i, \text{rule active}} M_i(x) \cdot y_i$$
Consequently:
$$\frac{\partial \text{output}}{\partial M_i(x)} = \frac{\sum_{i, \text{rule active}} M_i(x) \cdot y_i}{S(x)}$$

Considering the term $\frac{\partial M_i(x)}{\partial f_j}$:
$$\frac{\partial M_i(x)}{\partial f_j} = \frac{\sum_{i, \text{rule active}} M_i(x) \cdot y_i}{S(x)}$$
Finally, from the definition of $\frac{\partial \mu_M(x_j)}{\partial f_j}$ (Table I):
$$\frac{\partial \mu_M(x_j)}{\partial f_j} = -f_j$$
Combining all the above results:
$$\frac{\partial E}{\partial w_j} = -\text{error} \cdot \frac{y_j - \text{output}}{S(x)} \frac{\sum_{i, \text{rule active}} M_i(x) \cdot (-f_j)}{\mu_M(x_j)}$$
Therefore, the input weight updating rule is:
$$\Delta w_j = -\text{lr} \cdot \frac{y_j - \text{output}}{S(x)} \frac{\sum_{i, \text{rule active}} M_i(x) \cdot (-f_j)}{\mu_M(x_j)} \cdot f_j$$
For every input variable, only one 'side' of the corresponding membership function is updated: if $x_j < u_j$, the following two equations apply directly, otherwise if $x_j > u_j$, $j$ should be replaced by $M + j$ in the indices of $x$, $w$, and $f$.

Following the same lines for $f_{ij}$, we observe that the only difference is that the term $\partial y_{ij}(x_j)/\partial u_{ij}$ is used instead of $\partial y_{ij}(x_j)/\partial u_{ij}$, giving the rule:

$$\Delta u_{ij} = -l_j \cdot \text{error} \cdot \frac{\partial y_{ij}(x_j)}{\partial u_{ij}} \cdot \frac{\partial M_i(x_j)}{\partial w_{ij}} (w_{ij} - x_j)$$

Using these guidelines, one can obtain the updating equations for each fuzzy implication, which can also be found in [9].

REFERENCES
