Adaptive Neuro-Fuzzy Inference Systems (ANFIS)
Library for Simulink™

Ilias S. Konsoulas
MSc, CEng, MIET

1. Introduction

In these notes the operation and usage of the ANFIS library is described. This library consists of six different ANFIS models.

The Adaptive Neuro-Fuzzy Inference System (ANFIS) is a sophisticated hybrid network that belongs to the category of neuro-fuzzy neural networks. The main reference used to develop this library was the comprehensive text by J-S. R. Jang, C-T. Sun and E. Mizutani [1].

This library was implemented in MatLab and the six functions are Level-1 Simulink functions (S-functions). It was tested and run successfully in the environment of Simulink 5.0, R13 (MatLab 7.13.0.564 (R2011b)).

The ANFIS system is already implemented in MatLab as a built-in function of the Fuzzy Logic Toolbox. This library provides the means to use ANFIS in Simulink and therefore, it is implemented to work in an "on-line" mode.

2. Types of Various ANFIS Systems (Scatter, Grid, ART, MISO, MIMO)

Six types of ANFIS systems were developed and comprise this library. The "Multiple-Inputs-Single-Output" or MISO systems are simply labeled as "ANFIS" while the "Multiple-Inputs-Multiple-Outputs" or MIMO systems are labeled as "CANFIS") (see Fig. 3) in agreement with terminology introduced in [1].

Each of these 2 groups comes in three flavors: the "Scatter", the "Grid" and the "ART" types. This naming refers to the employed method of input space partitioning. The partitioning of the space of input variables affects directly the architecture, the operation and therefore the approximating capacity of each ANFIS/CANFIS system.

In general, Grid-type input space partitioning is computationally greedy as the number of input variables (here signals) increases. In such a case, the number of fuzzy rules increases exponentially with the number of inputs as indicated by the relation:

$$N_{Rules} = N_{Input\_Terms}^{N_{Inputs}} \tag{1}$$

This is just another occurrence of the widely known "curse of dimensionality". To surpass this problem, we resort to the Scatter-type input space partitioning where the number of fuzzy rules equals the number of the fuzzy subsets of each input:

$$N_{Rules} = N_{Input\_Terms}\,. \tag{2}$$

This trick saves the day in terms of required computational power, but reduces the approximating power of the ANFIS. This happens because this architecture does not partition the input space thoroughly but only along (or close to) the area that includes the main diagonal of the space spanned by the input variables. If data appear away from the main diagonal, it is very

likely that the scatter type ANFIS model will not perform as well there. Detailed description of Input Space Partitioning techniques is found in 4.5.1 of [1].

A much smarter (and algorithmically more complex) way of input space partitioning is the application of the Fuzzy-ART (ART for Adaptive Resonance Theory) algorithm proposed by Carpenter et al. in 1991 [2]. The ART-type ANFIS model resembles the Scatter-type ANFIS model but it is much smarter in the clustering of input data. Being a well-known unsupervised learning technique, the fuzzy-ART algorithm creates input data categories (and therefore we allocate the net's membership functions there) exclusively on areas of the input space where data appear.

3. The “Grid” and “Scatter” ANFIS/CANFIS Architectures

3.1 ANFIS and CANFIS Networks

The ANFIS system, as its name suggests, is an adaptive neuro-fuzzy inference machine. Like "pure" (i.e. non-fuzzy) artificial neural networks or classic fuzzy systems, it works as a universal approximator [1]. Its purpose is to approximate or “learn” simple or complicated mappings (i.e. nonlinear functions) from an input space (usually multivariate) to a univariate or multivariate output space.

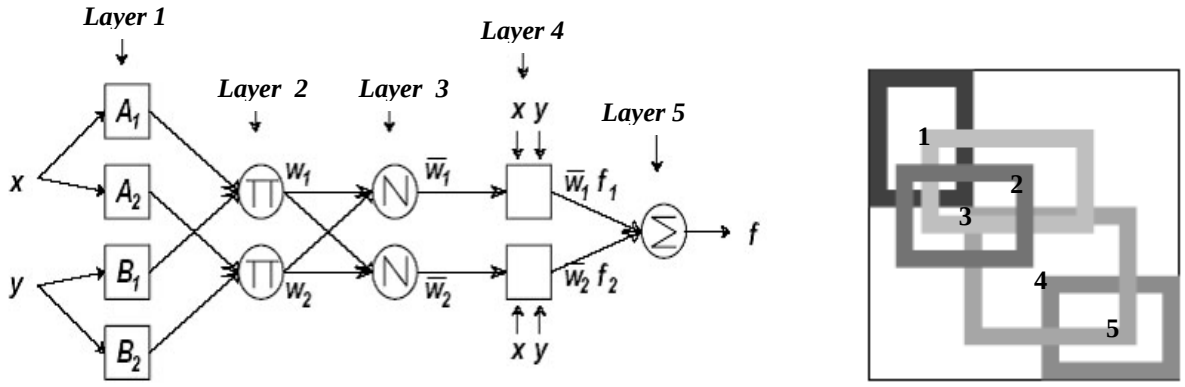


Fig. **1** "Scatter" Type ANFIS Network Architecture (adopted from [1]).

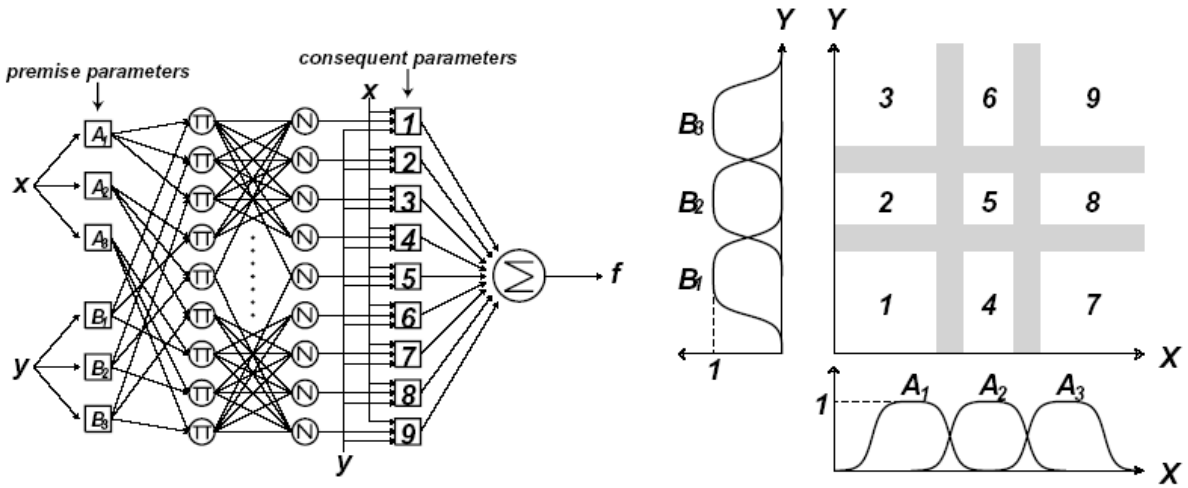


Fig. **2** "Grid" Type ANFIS Network Architecture (adopted from [1]).

3.2 Description of ANFIS Operation

The classic ANFIS (and CANFIS) consist of five layers (see Fig.1 and Fig.2). In the following lines we describe briefly the operation of each layer. For a full description please refer to [1].

Layer No.1 (Inputs Layer)

In this layer input fuzzification takes place. This means that each input is assigned a *membership value* to each fuzzy subset that comprises that input's *universe of discourse*. Mathematically, this function can be expressed as:

$$Out_{ij}^{(1)} = \mu_j(In_i^{(1)}) . \quad (3)$$

Where $Out_{ij}^{(1)}$ is the layer 1 node's output which corresponds to the *j*-th linguistic term of the *i*-th input variable $In_i^{(1)}$. As the default membership function for every ANFIS system in this library we have selected the *generalized Gaussian function*:

$$\mu_j(x_i) = \frac{1}{1+\left|\frac{x_i - c_{ij}}{a_{ij}}\right|^{b_{ij}}}, \; i = 1,..., \text{NumInVars}, \; j = 1,..., \text{NumInTerms}. \quad (4)$$

while the triplet of parameters (a_{ij}, b_{ij}, c_{ij}) are referred to as *premise parameters* or *non-linear parameters* and they adjust the shape and the location of the membership function. Those parameters are adjusted during the training mode of operation by the error back-propagation algorithm. Alternatively, here one may use the well-known bell-shaped membership function:

$$\mu_j(x_i) = e^{-\frac{1}{2}\left(\frac{x_i - c_{ij}}{\sigma_{ij}}\right)^2}, \; i = 1,..., \text{NumInVars}, \; j = 1,..., \text{NumInTerms}. \quad (5)$$

Layer No.2 (Fuzzy AND Operation)

Each node in this layer performs a fuzzy-AND operation. For all ANFIS networks of the library, the *T-norm operator* of the algebraic product was selected. This results to each node's output being the product of all of its inputs (every input term node that is connected to that rule node):

$$Out_k^{(2)} = w_k = \prod_{i=1}^{N_{Inputs}} Out_{ij}^{(1)}, \text{for all the term nodes } j \text{ connected to the } k\text{ - th rule node}. \quad (6)$$

$$k = 1,..., \text{NumRules}.$$

The output of each node in this layer represents the firing strength (or activation value) of the corresponding fuzzy rule.

Layer No.3 (Normalization)

The output of the *k*-th node is the firing strength of each rule divided by the total sum of the activation values of all the fuzzy rules. This results in the normalization of the activation value for each fuzzy rule. This operation is simply written as:

$$Out_k^{(3)} = \overline{w_k} = \frac{Out_k^{(2)}}{\sum_{m=1}^{N_{Rules}} Out_m^{(2)}}, \quad k = 1,..., \text{NumRules}. \tag{7}$$

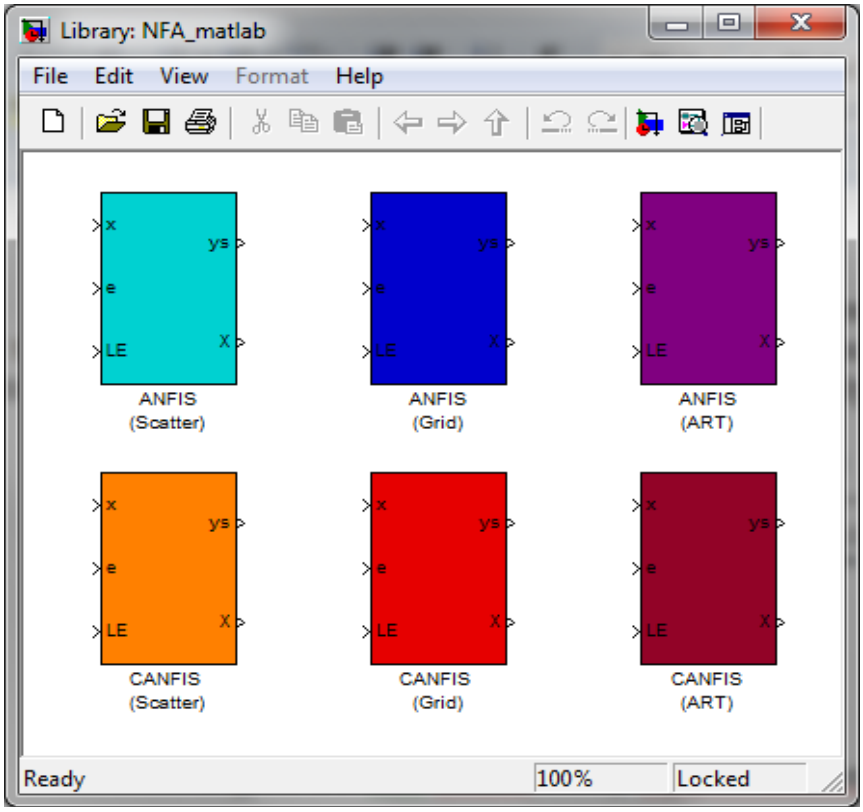


Fig. **3** The ANFIS Library for Simulink.

Layer No. 4

Each node k in this layer is accompanied by a set of adjustable parameters $a_{1k}, a_{2k}, \ldots, a_{N_{Inputs}k}, a_{0k}$ and implements the linear function:

$$Out_k^{(4)} = \overline{w_k} f_k = \overline{w_k}(a_{1k} In_1^{(1)} + a_{2k} In_2^{(1)} + \ldots + a_{N_{Inputs}k} In_{N_{Inputs}}^{(1)} + a_{0k}) \; . \tag{8}$$

$k = 1,...,$ NumRules.

The weight $\overline{w_k}$ is the normalized activation value of the *k*-th rule, calculated by aid of (7). Those parameters are called *consequent parameters* or *linear parameters* of the ANFIS system and are adjusted by the RLS algorithm.

Layer No.5 (Output Layer)

For ANFIS (MISO) this layer consists of one and only node that creates the network's output as the algebraic sum of the node's inputs:

$$Out^{(5)} = \sum_{k=1}^{N_{Rules}} Out_k^{(4)} = \sum_{k=1}^{N_{Rules}} \overline{w_k} f_k = \frac{\sum_{k=1}^{N_{Rules}} w_k f_k}{\sum_{k=1}^{N_{Rules}} w_k}. \quad (9)$$

3.3 Description of CANFIS Operation

The operation of CANFIS network is the same as that of ANFIS up to Layer 3. The MIMO CANFIS network architecture changes from Layer 4 and forward.

Layer No.4 for CANFIS (MIMO)

In such a system, the output of the *k*-th fuzzy rule that influences the *m*-th network output, is written as:

$$Out_{km}^{(4)} = \overline{w_k} f_k^m = \overline{w_k}(a_{1k}^m In_1^{(1)} + a_{2k}^m In_2^{(1)} + \ldots + a_{N_{Inputs}k}^m In_{N_{Inputs}}^{(1)} + a_{0k}^m), \quad (10)$$

k = 1,..., NumRules, m = 1,..., NumOutVars.

The parameters $a_{1k}^m, a_{2k}^m, \ldots, a_{N_{Inputs}k}^m, a_{0k}^m$ are the consequent parameters of the CANFIS system that represent the contribution of the *k*-th rule to the *m*-th output.

Layer No. 5 for CANFIS (MIMO)

The *m*-th output of the network is computed as the algebraic sum of the *m*-th node's inputs:

$$Out_m^{(5)} = \sum_{k=1}^{N_{Rules}} Out_{km}^{(4)} = \sum_{k=1}^{N_{Rules}} \overline{w_k} f_k^m, \quad m = 1,..., \text{NumOutVars}. \quad (11)$$

4. The Fuzzy-ART ANFIS and Fuzzy-ART CANFIS Network Architectures

The ANFIS-ART and CANFIS-ART networks consist of six layers. They are the "smarter" types of ANFIS/CANFIS networks included in this library because they employ 2 algorithms for parameter learning (i.e. RLS and error - backpropagation) and 1 algorithm for automatic structure learning (i.e. fuzzy-ART). In the following lines we describe briefly the operation of each layer. A descriptive representation of a 3-input ANFIS-ART network with 3-fuzzy rules is shown in Fig. 4.

The code segment that executes the Fuzzy-ART algorithm was adopted (with the necessary modifications) from the well-structured, reliable and tractable Fuzzy-ART codes developed by Aaron Garrett [4].

4.1 Description of ANFIS-ART Operation.

Layer No.1 (Inputs Normalization Layer)

The ANFIS-ART uses the technique of *complement coding* from fuzzy-ART [2] to normalize the input training data. Complement coding is a normalization process that replaces an *n*-dimensional input vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ with its 2*n*-dimensional complement coded form **x'** such that:

$$\mathbf{x}' \equiv [\,\bar{x}_1, 1-\bar{x}_1, \bar{x}_2, 1-\bar{x}_2, \ldots, \bar{x}_n, 1-\bar{x}_n\,] \tag{12}$$

where $[\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n] = \bar{\mathbf{x}} = \mathbf{x} / \|\mathbf{x}\|$. As mentioned in [2], complement coding helps avoiding the problem of category proliferation when using fuzzy-ART for data clustering. Having this in mind, we can write the I/O function of the first layer as follows:

$$Out_i^{(1)} \equiv (\,\overline{In_i^{(1)}},\, 1-\overline{In_i^{(1)}}\,), \qquad i = 1, 2, \ldots, \text{NumInVars.} \tag{13}$$

Layer No.2 (Input Fuzzification Layer)

The nodes belonging to this layer are called input-term nodes and each represents a term of an input-linguistic variable and functions as an 1-D membership function. Here we use the following trapezoidal membership function:

$$Out_{ij}^{(2)} = 1 - g(In_{ij}^{(2)} - \upsilon_{ij}^{(2)}, \gamma) - g(u_{ij}^{(2)} - In_{ij}^{(2)}, \gamma),\ i = 1, 2, \ldots, \text{NumInVars} \tag{14}$$

where $u_{ij}^{(2)}$ and $\upsilon_{ij}^{(2)}$ are, the left-flat and right-flat points of the trapezoidal membership function of the *j*-th input-term node of the *i*-th input linguistic variable. $In_{ij}^{(2)}$ is the input to the *j*-th input-term node from the *i*-th input linguistic variable (i.e. $In_{ij}^{(2)} = Out_i^{(1)}$). Also, the function *g*(.) is defined as:

$$g(s,\gamma) = \begin{cases} 1, & \text{if } s\gamma > 1 \\ s\gamma, & \text{if } 0 \le s\gamma \le 1 \\ 0, & \text{if } s\gamma < 0. \end{cases} \tag{15}$$

The parameter γ regulates the fuzziness of the trapezoidal membership function. More details on how this membership function works on the real *n*-dimensional space combining *n* inputs, can be found in [3].

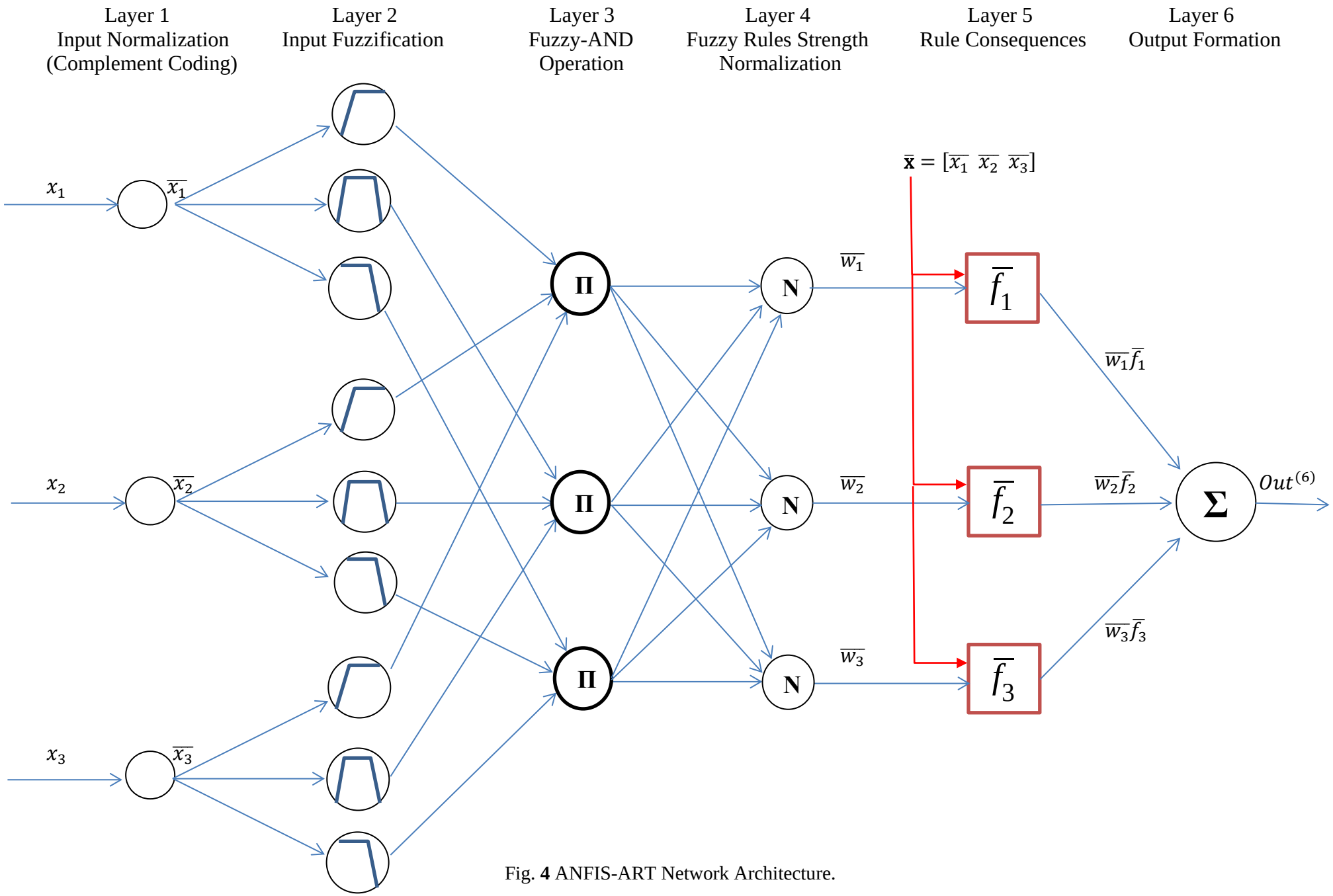


Fig. **4** ANFIS-ART Network Architecture.

Layer No.3 (Fuzzy-AND Operation)

Each node in this layer performs a fuzzy-AND operation. Similarly to the other ANFIS networks of the library, the *T-norm operator* of the algebraic product was selected. This results each node's output to be the product of all its inputs:

$$Out_{k=j}^{(3)} = w_{k=j} = \prod_{i=1}^{N_{Inputs}} Out_{ij}^{(2)}, \quad k\,(=j) = 1,..., \text{NumInTerms}. \qquad (16)$$

The output of each node in this layer represents the firing strength (or activation value) of the corresponding fuzzy rule. Note that the number of the fuzzy rules equals the number of input term nodes. The latter is common for all the input variables. Therefore, each fuzzy rule may be assigned an index *k* equal to the corresponding index *j* of the input term node, which is common for each input linguistic variable.

Layer No.4 (Normalization of Each Rule Firing Strength)

The output of the *k*-th node in this layer, is the firing strength of each rule divided by the total sum of the activation values of all the fuzzy rules. This results in the normalization of the activation values of all fuzzy rules:

$$Out_k^{(4)} = \overline{w_k} = \frac{Out_k^{(3)}}{\sum_{m=1}^{N_{Rules}} Out_m^{(3)}} \qquad k\,(=j) = 1, 2, ..., \text{NumRules}. \qquad (17)$$

Layer No. 5

Each node *k* in this layer is accompanied by a set of adjustable parameters $a_{1k}, a_{2k}, \ldots, a_{N_{Inputs}k}, a_{0k}$ and implements the linear function:

$$Out_k^{(5)} = \overline{w_k}\,\overline{f_k} = \overline{w_k}(a_{1k}\overline{In_1^{(1)}} + a_{2k}\overline{In_2^{(1)}} + \ldots + a_{N_{Inputs}k}\overline{In_{N_{Inputs}}^{(1)}} + a_{0k})$$
$$k = 1, 2, ..., \text{NumRules}. \qquad (18)$$

The weight $\overline{w_k}$ is the normalized activation value of the *k*-th rule calculated by aid of (17). Those parameters are called *consequent parameters* or *linear parameters* of the ANFIS system and are regulated by RLSE algorithm.

Layer No.6 (Output Layer)

For ANFIS-ART (MISO) this layer consists of one and only node that creates the network's output as the algebraic sum of the node's inputs:

$$Out^{(6)} = \sum_{k=1}^{N_{Rules}} Out_k^{(5)} = \sum_{k=1}^{N_{Rules}} \overline{w_k}\,\overline{f_k} = \frac{\sum_{k=1}^{N_{Rules}} w_k\,\overline{f_k}}{\sum_{k=1}^{N_{Rules}} w_k}. \qquad (19)$$

4.2 Description of CANFIS-ART Operation.

The operation of CANFIS-ART network is the same as that of ANFIS-ART up to Layer 4. The MIMO CANFIS-ART network architecture changes from Layer 5 and forward.

Layer No.5 for CANFIS-ART (MIMO)

In this network, the output of the *k*-th fuzzy rule that contributes to the *m*-th network output, is written as:

$$Out_{km}^{(5)} = \overline{w_k}\,\overline{f_k^{\,m}} = \overline{w_k}\,(a_{1k}^m\,\overline{In_1^{(1)}} + a_{2k}^m\,\overline{In_2^{(1)}} + \ldots + a_{N_{Inputs}k}^m\,\overline{In_{N_{Inputs}}^{(1)}} + a_{0k}^m) \qquad (20)$$

k = 1, 2, ..., NumRules, m = 1, 2, ..., NumOutVars.

The parameters $a_{1k}^m, a_{2k}^m, \ldots, a_{N_{Inputs}k}^m, a_{0k}^m$ are the consequent parameters of the CANFIS-ART system that represent the contribution of the *k*-th rule to the *m*-th output.

Layer No. 6 for CANFIS-ART (MIMO)

The *m*-th output of the network is computed as the algebraic sum of the *m*-th node's inputs:

$$Out_m^{(6)} = \sum_{k=1}^{N_{Rules}} Out_{km}^{(5)} = \sum_{k=1}^{N_{Rules}} \overline{w_k}\,\overline{f_k^{\,m}}, \quad m = 1,..., \text{NumOutVars.} \qquad (21)$$

5. Block Inputs, Outputs and Parameters

One very nice thing about this library is that the inputs and the outputs for all the six blocks that consist this library, are identical.

5.1 Inputs

Input **x**: This is the entrance for the actual inputs to the (C)ANFIS system. If we want to supply the system with more than one input, then this must be a vector formed by aid of a multiplexer.

Input **e**: This is where we supply the (C)ANFIS with the training error signal. This signal has the same dimensions as the output signal. Therefore and less surprisingly, for the MISO ANFIS it is just a scalar and for a CANFIS it's a vector.

Input LE: This is a two-state discrete, scalar Learning Enable (LE) input. As its name suggests, it enables (LE=1) or disables (LE=0) the learning of (C)ANFIS. When LE=1 the training of the network takes place while LE=0 switches training off and approximating operation based on experience gathered by previous training takes place.

5.2 Outputs

Output $\mathbf{y_s}$: This is the main output for each (C)ANFIS block. For ANFIS this is a scalar while for CANFIS it's a vector.

Output **X**: This output provides the parameters (or state) of the (C)ANFIS network during training in vector format in order to be saved for future use. It serves an important purpose since we would have to train the (C)ANFIS systems from scratch each time we started a new Simulink session. The save function is performed using the standard simulink sink block "To File". There, we simply store successive values of **X** vector in a big matrix. When the time to store a new snapshot of **X** comes, the whole vector is added as a new column to the right-most column of the existing matrix. This "incremental" evolution of the ANFIS system parameters matrix allows for subsequent observation of every single parameter's "personal history" by looking at the corresponding row of the matrix. Also, in the "To File" block parameters we can select the name of the .mat file where the snapshots of the **X** vector are going to be stored. We may also choose a more descriptive name for the big matrix that keeps all those vectors by providing for the "Variable Name" entry.

5.3 ANFIS Block Parameters.

Ita (η) : This is the "learning rate" constant used in the error back-propagation algorithm to adjust Layer 1 parameters. These are the parameters of the membership functions to the fuzzy subsets the input space is partitioned to.

alpha (α): This is the "momentum term" constant that relates to the error back-propagation algorithm used to adjust the parameters of Layer 1.

lambda (λ): This is the "forgetting factor" associated with the Recursive Least Squares (RLS) algorithm that is used to adjust the linear parameters of Layer 4.

[NumInVars NumInTerms]: This self-explanatory 2-element vector contains the number of input signals (or variables) and the selected number of input linguistic terms. Therefore, the number of terms equals to the common number of fuzzy subsets each input is partitioned to.

Inputs' "Universe of Discourse" Start: MinsVector: This vector contains the maximum lower bound (MLB) for each input. Together with the next vector parameter, this vector of MLB values is used to determine the normal "universe of discourse" of the inputs in order to partition it in a uniform way.

Inputs' "Universe of Discourse" End: MaxsVector: This vector contains the minimum upper bound (MUB) for each input. This vector of MUB values together with the previous vector parameter, assume that the "universe of discourse" for each input is known a priori.

Initial ANFIS States: Here one may choose between 2 possible values. When a 0 is entered, it initializes the ANFIS system to a "tabula rasa" state meaning that it knows nothing about the task it is meant to be used for and that it's ready for initial training. On the other hand, if we enter the vector that corresponds to a column of the **X** output, saved to a .mat file from previous training, then the ANFIS is set to that state. This way we can "reload" to the system "experience" acquired through previous training.

Sampling Time: This parameter sets the sampling time of the block. It should be set equal to the fundamental sample time of the model where it is used.

5.4 CANFIS Block Parameters.

All of the CANFIS parameters are the same and serve a similar purpose as those of ANFIS described above. The only difference is:

[NumInVars NumInTerms NumOutVars]: This self-explanatory three-element vector parameter contains three numbers of great importance for the architecture of the CANFIS system. NumInVars is the number of Inputs (i.e. the size of **x** input vector). NumInTerms is the number of the selected number of input linguistic terms (same as in ANFIS). NumOutVars is the number of outputs that CANFIS has to track i.e. the dimension of $\mathbf{y_s}$ output.

5.5 ANFIS-ART and CANFIS-ART Block Parameters.

These blocks have many common parameters with the non-ART ANFIS and CANFIS blocks. The parameters coming under the same labels and description, have an identical function and purpose as those described above. However, these 2 blocks have a number of unique parameters that should be set by the user and are attributed to the incorporation of the fuzzy-ART algorithm for input space partitioning. For details please refer to [2].

rho_a (ρ_a) : This is the "vigilance parameter" of the fuzzy-ART algorithm that serves the task of input space partitioning. By default, $0 < \rho_a < 1$. The closer ρ_a is to zero, the bigger (coarser) the learned categories become. On the other hand, when ρ_a is set close to one, the categories become finer. For the exact description of the way the fuzzy-ART parameters influence the ART learning, please refer to [2].

Alpha (a): This is the "choice parameter" of the fuzzy-ART algorithm.

Beta (β): This is the fuzzy-ART "learning rate" parameter.

MF Inclination Factor (γ): This parameter sets the fuzziness of the trapezoidal membership functions that constitute the function of the input term nodes. A big γ (i.e. > 4) implies a crisper fuzzy set. A low γ gamma value (i.e. < 4) creates a fuzzier set.

Max Num. of Input Variables Terms (== Max Num of Fuzzy Rules): This number sets the maximum number of input terms that will be generated by the fuzzy-ART algorithm. Due to the fact that the number of the input term nodes is equal to the number of fuzzy rules, this number also limits the number of maximum allowable fuzzy rules that will be generated. As explained thoroughly by the fuzzy-ART theory, the number of the generated categories for a given dataset is directly influenced by the "vigilance parameter" ρ_a. When the categories (or clusters) created on-line reaches this number, the creation of new categories stops. Only the adjustment of the Layer 2 parameters $u_{ij}^{(2)}$ and $v_{ij}^{(2)}$ by aid of error-backpropagation continues to happen (as long as LE=1).

5.6 Block Parameter's Typical Values.

Ita (η) : $10^{-6} \leq \eta \leq 0.1$, alpha ($\alpha$): $0 \leq \alpha \leq 10^{-4}$.

lambda (λ): $1\text{-}10^{-3} \leq \lambda \leq 1$. The RLS algorithm is very sensitive to this parameter. If set to a lower than 0.95 value, RLS may diverge and the whole simulation will crush spectacularly!

[NumInVars NumInTerms NumOutVars].

NumInVars, NumOutVars : While theoretically "the sky is the limit" inputs should be limited to an "absolutely necessary" group of relevant inputs. If you increase the NumInVars to a value higher than 8 and you use the Grid version of the ANFIS systems, you will notice a considerable drop in execution speed. Up to 3 outputs were simultaneously tracked in my thesis experiments without any performance issues.

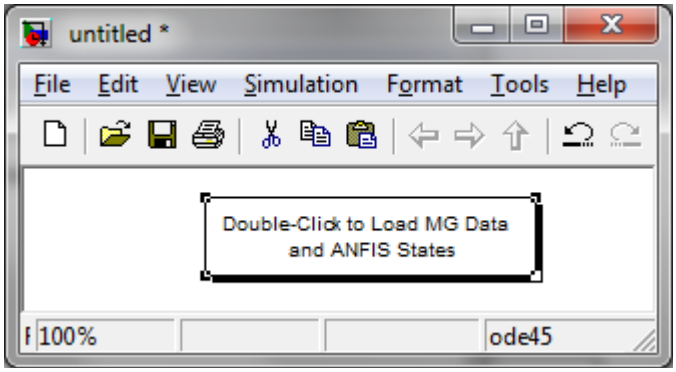
NumInTerms: $2 \leq \text{NumInTerms} \leq 7$.

6. Demos

6.1 Setting-Up the Workspace

In order to help the future users of this library, several demo experiments have been developed and accompany the library models. There are 8 ANFIS and 3 CANFIS demos. Here we describe what each demo does and how you will make it runnable at your computer.

Each demo contains a block that has to be double-clicked in order to generate, format and load the necessary data into the current workspace. A representative sample of this block is shown below:



This block contains a small set of instructions that have to be modified slightly before and after the first time a demo is run. The user has to right-click on it, and select "Block Properties" -> Callbacks->OpenFcn. There, appears the small set of instructions that have to run before the simulation is ready to start. A typical set of these instructions are:

```
clc;
clear;
SimDataGenAnfis1;
load anfis_scatter.mat
Ts = 1;
```

The first 2 commands are known MatLab commands that clear the screen and the workspace. The command SimDataGenAnfis1 is a custom function that generates, formats and loads the input/output data pairs necessary for the training and checking sessions of the simulation. The next command: *load anfis_scatter.mat* is the most important since it loads the states of the ANFIS or CANFIS model used in each experiment. This command must be commented with a % symbol at the first time each demo is to be run because initially, no such data exist. The user has to train a network at least once in order for this .mat file to be generated. Therefore, the second time you want to run the demo (in order to check what ANFIS really learnt), you must uncomment this line. When this command is enabled, it loads a matrix variable in the workspace (e.g. anfis_scatter_states) which contains the states of the ANFIS network. You must enter the name of this variable at the dedicated entry of the ANFIS network's mask titled: "Initial ANFIS States: $[x_0, d_0]$[1]". By doing this, and by double-clicking the loading box the ANFIS/CANFIS will know from what initial conditions to start the next simulation. This mechanization is absolutely necessary because it allows the gained training experience to be stored and restored as desired without having to retrain the network from scratch each time we want to run a simulation.

6.2 Demos Description

As already stated, 2 families of demos come together with the ANFIS/CANFIS library. They are, the ANFIS demos and the CANFIS demos. In this section we provide a short description of each demo presented.

6.2.1 ANFIS Demos

- ANFIS_Scatter_MG.mdl

In this demo we train an ANFIS/Scatter model on the prediction of the Mackey-Glass (MG) chaotic time series. The network takes as input the current and 3 past values of the MG time-series $[x(t\text{-}3)\ x(t\text{-}2)\ x(t\text{-}1)\ x(t)]$ and is trained to predict the value of the next one $x(t+1)$. The simulation lasts for 1000 samples (or time instants) of which the first half is dedicated to training (LE=1) and the second half is dedicated to checking (LE=0).

- ANFIS_Grid_MG.mdl

Same as the previous one but using an ANFIS/Grid model to do the job.

- ANFIS_ART_MG.mdl

Similar to ANFIS_Scatter_MG.mdl but employing an ANFIS/ART model. Here, the total time of the simulation lasts 2000 samples (or time instants). Again, the first half is dedicated to training and the last half is dedicated to checking.

- anfis_art_narmax.mdl

[1] The user must insert a zero there at the very first time an ANFIS/CANFIS system is trained, to indicate that training starts from scratch.

In this demo, a nonlinear single-input-single-output (SISO) NARMAX[2] system is identified by use of an ANFIS/ART network. The simulation lasts 1000 time increments. During the first half of the simulation training is taking place while in the last half, the ANFIS/ART is checked for learning efficiency. The system accepts 4 inputs from the ARMA process [$u(t\text{-}2)$ $u(t\text{-}1)$ $y(t\text{-}2)$ $y(t\text{-}1)$] and is trained to estimate the value of the current output $y(t)$. The closer the ANFIS/ART output is to the actual output the better the network has learnt the behavior of the NARMAX system. There is an error and a comparison scope for the user to inspect the progress of the ANFIS/ART learning.

- bb_anfis_grid.mdl (requires the Fuzzy Logic Toolbox)

This demo shows how an ANFIS/Grid network can be trained to successfully control the well-known "ball and beam system". The framework of this demo was taken from a similar demo of the Fuzzy Logic Toolbox and it was modified to suit our purposes. Each simulation session (training or checking) lasts for 1000 seconds. In the training session, a four-input ANFIS/Grid is trained by the control law on the task of controlling the ball and beam system. During training, the manual switch must forward the output of the control law in order for the plant to be controlled. Also, the user must manually set the LE step input to 1 for this session. On the contrary, before starting the checking session and after loading the learnt ANFIS/Grid states, we should set the manual switch so as to forward the output of the ANFIS system and set the LE step input to 0. As described, training and checking require 2 distinct simulation runs and not one single run as happened in all the above demos.

- bb_anfis_scatter.mdl (requires the Fuzzy Logic Toolbox)

Quite similar to the above demo, this one demonstrates how an ANFIS/Scatter network can be trained to learn how to control the ball and beam system.

- Adapt_Equalizer.mdl

In this demo, an ANFIS/Grid system plays the role of an adaptive equalizer at the receiver of a rudimentary communications system. The transmitted bits are received by the equalizer after they have undergone distortion by a non-minimum phase channel and an additive noise source. The ANFIS/Grid equalizer takes 2 inputs as received from the channel [$x(t\text{-}1)$ $x(t)$] and decides upon the correct value of the transmitted bit. The nature of the channel creates a non-linearly separable classification problem which the ANFIS solves successfully with minimal resource usage. For more details on this demo the interested reader is referred to chapter 19 of [1]. The demo lasts for 5000 samples of which only the first 1000 are dedicated to training.

- slcp_anfis.mdl (requires the Fuzzy Logic Toolbox)

Here, an ANFIS/Scatter net learns from a purely Fuzzy Logic controller how to control the cart and pole system. This demo is included in the Fuzzy logic toolbox and has been modified to demonstrate how an ANFIS system can successfully extract knowledge from an existing controller by simply exploiting its I/O data pairs for its own training. The ANFIS/Scatter shares common inputs with the fuzzy logic controller and learns how to generate an adequate control law for controlling the cart and pole system. Training and checking can only happen in distinct

[2] **N**on-linear **A**uto-**R**egressive **M**oving **A**verage with e**X**ogenous Inputs System.

simulations. During training the manual switch must forward to the cart and pole system the control signal generated by the controller whereas in the checking session it must forward the output signal of the ANFIS/Scatter. During training the LE input must be manually set to 1 while in the checking session this input must be manually set to 0. Also before checking starts, the "Initial ANFIS States [x0; d0;]" must be specified by inserting the name of the matrix variable that holds the initial states (i.e. x_anfis_states).

6.2.2 CANFIS Demos

- CANFIS_Scatter_Lorenz.mdl (requires the s-function sfunxyz.m[3])

In this demo, we teach a CANFIS/Scatter network the Lorenz chaotic system. This system is described by 3 ODE''s which have been numerically solved beforehand and the resulting trajectory of the system has been stored in the lorenz_data.mat file. We feed the CANFIS/Scatter net with the current and previous points of the trajectory and we train the net to predict the next point. Therefore, the input vector of the ANFIS/Scatter network has the form: [$x(t\text{-}1)$ $y(t\text{-}1)$ $z(t\text{-}1)$ $x(t)$ $y(t)$ $z(t)$] and the desired output is of the form [$x(t\text{+}1)$ $y(t\text{+}1)$ $z(t\text{+}1)$]. The simulation lasts for 2000 samples. The first 600 samples suffice to train the ANFIS/Scatter system on the Lorenz attractor.

- CANFIS_Grid_Lorenz.mdl (requires the s-function sfunxyz.m)

This is similar to the above demo but employs a CANFIS/Grid system for the same task.

- CANFIS_ART_Lorenz.mdl (requires the s-function sfunxyz.m)

Same as CANFIS_Scatter_Lorenz.mdl demo but employs an CANFIS/ART system for learning the Lorenz chaotic 3D time series.

Depending on the Configuration Parameters of the Simulation, you may receive some warnings when running the described demos. Those are mainly caused because of the algebraic loop present at the generation of the error signal. You can safely ignore (or suppress them at the Configuration Parameters/Diagnostics menu) these warnings because Simulink successfully resolves the algebraic looping issue for us and selects the correct output sample for use.

7. Backpropagation Implementation Verification

7.1 Implementation Pitfalls

Implementing the backpropagation algorithm for (C)ANFIS is not a simple task. Due to the complex nature of the neuro-fuzzy networks' architectures you can hardly be 100% sure of the correctness of your implementation. You may even see a steadily decreasing mean squared (MSE) or instantaneous error $e(n)$ and still have several tiny bugs "thriving" in some dark corners of your code. There is a simple but very efficient way to verify that your

[3] Which can be downloaded from: http://www.mathworks.com/matlabcentral/fileexchange/3019-airlib/content/Airlib/sfunxyz.m

backpropagation implementation works 100% bug-free and it comes under the name: gradient checking.

7.2 Gradient Checking

Gradient checking is your ultimate weapon to become sure of the correctness of your backpropagation implementation. Remember that backpropagation is based on the *steepest descent method*[4] for optimizing a network's parameters. This means that all you need to check is how accurately the gradient of the error (or cost function J) $\partial J / \partial w$ w.r.t. to every adjustable net parameter w is calculated by your implementation. The alternative way to calculate this quantity is by numerically computing it as described in the next lines. If the difference of the 2 gradient values is of the order 10^{-8} or less, you may stay assured that you backpropagation routine works like a Swiss-made clock.

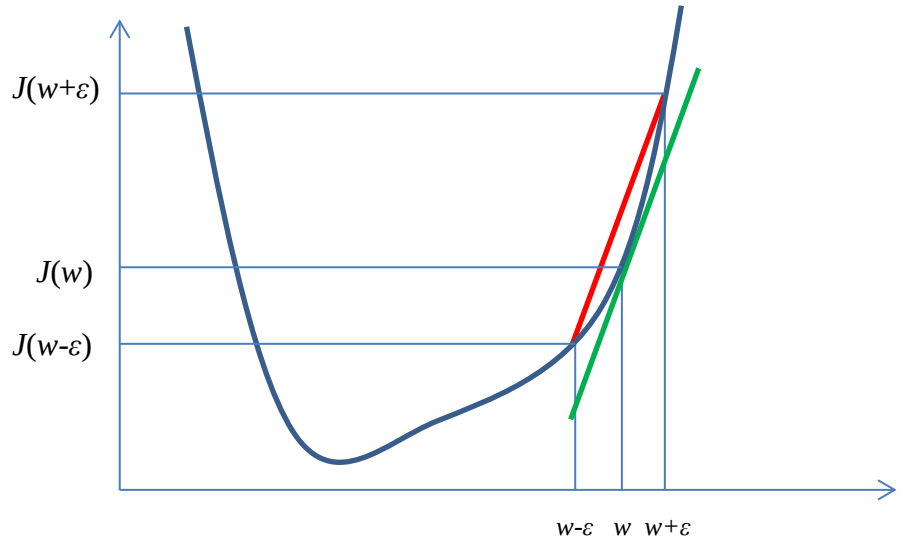


Fig. **5** Numerical Approximation of Cost Function Gradient.

Suppose that your neuro-fuzzy network optimizes a cost (or error) function $J(\mathbf{w})$. This is a multivariate cost function having as independent variables the whole set of networks' adjustable parameters grouped together in the vector $\mathbf{w} = [w_1, w_2, \ldots, w_i, \ldots w_n]$. Assume that we desire to check how accurate is our backpropagation implementation on estimating the value of the gradient of $J(\mathbf{w})$ w.r.t. parameter w_i, i.e. in evaluating $\partial J(\mathbf{w}) / \partial w_i$. The direct method of calculating the value of this partial derivative is given by the following formula:

$$\frac{\partial J(\mathbf{w})}{\partial w_i} = \frac{\partial J(w_1, w_2, \ldots, w_i, \ldots, w_n)}{\partial w_i} \approx \frac{J(w_1, w_2, \ldots, w_i + \varepsilon, \ldots, w_n) - J(w_1, w_2, \ldots, w_i - \varepsilon, \ldots, w_n)}{2\varepsilon} \tag{22}$$

Where ε is a small positive number: 10^{-4} or 10^{-5}. This numerical value corresponds to the slope of the red line segment in Fig. 5. What becomes obvious in this figure is that as $\varepsilon \to 0$, the slope of

[4] For details about steepest descent method see Appendices B and C.

the red chord tends to coincide with the slope of the green tangent line which is the true gradient of $J(\mathbf{w})$ at w_i. Therefore, if our backpropagation algorithm approximates this value at a satisfactory level (i.e. differs less that 10^{-8}), we can be certain that it correctly calculates the gradient in question. As suggested by formula (22), the rest of the parameters play no significant role during this direct gradient approximation (remain constant).

7.3 ANFIS/CANFIS Operational Assumptions

It can be proved that (C)ANFIS satisfies the conditions of the *Stone-Weirstrass theorem* [1] and therefore it is classified as a *universal approximator*. This fact guarantees that it has unlimited approximation power for matching any nonlinear function arbitrarily well, when the number of fuzzy rules is unrestricted.

However, in practice, (C)ANFIS will not solve all the approximation problems of this world. A basic and fundamental assumption that your data must satisfy is the existence of a continuous (or acceptably smooth) mapping f between the input and output data pairs. Furthermore, in order for the backpropagation to work in an acceptable manner, the partial derivatives of the error surface w.r.t. every single adjustable parameter should exist (i.e. remain finite).

7.4 Warning!

During the development of these S-functions no care at all was taken for input signals or parameter checking/validation. While it is well known that this is not a good programming practice, it is also assumed that the library user has a basic ANFIS background and has read these notes before attempting to use it.

Acknowledgements

I would like to express my deep appreciation and thankfulness to Dr. Giampiero Campa for his enormous help on coding this library of S-functions. Without his generous contribution this small library wouldn't become a reality.

Also, I would like here to thank Mr. Aaron Garrett for the successful coding of the fuzzy-ART and fuzzy ARTMAP algorithms and sharing them with the MatLab users community.

Appendix A

Number of ANFIS/CANFIS network parameters.

ANFIS is a sophisticated fuzzy-neural network with powerful approximating capabilities. This level of performance comes at a considerable price. First of all, this hybrid net uses five consecutive layers and takes advantage of 2 (or 3) training algorithms (i.e. back-propagation, RLS and ART). All the experience gained during training is stored in 2 families of parameters. These are Layer's 1 parameters (nonlinear or premise parameters) and Layer's 4 parameters (linear or consequent parameters). As already stated, premise parameters are tuned via the standard error back-propagation algorithm (and ART) while the consequent parameters via the classic RLS algorithm. In the following sub-paragraphs I provide an account of these two big families of parameters.

A.1 ANFIS Parameters

NumInVars: Number of input variables (in simulink is the number of input signals).

NumInTerms: Selected number of fuzzy subsets that "divide" the domain of each input. For example, imagine an experiment where we have a temperature input variable and divide the whole range of temp values in fuzzy subsets labeled with the following linguistic terms: "cold", "cool", "lukewarm", "warm", "hot". Apparently, in such a case, NumInTerms = 5. This number is common for each input variable.

NumRules: Number of fuzzy rules comprising the ANFIS inference engine.

NumInTerms: Number of terms of the input linguistic variables.

NumOutVars (or NOV): Number of output variables (or signals). For ANFIS is 1 and for CANFIS >1.

ns: Number of network's states. Here is the sum of all the parameters.

nds: Number of network's differential states. This is the sum of all parameter gradient values needed for error-backpropagation and RLS.

A.1.1 "Scatter" Type

$NumRules = NumInTerms;$

$ns = 3*NumInVars*NumInTerms + [(NumInVars+1)*NumRules]^2 + (NumInVars+1)* NumRules;$

$nds = 3*NumInVars*NumInTerms + (NumInVars+1)*NumRules;$

A.1.2 "Grid" Type

$NumRules = NumInTerms^{NumInVars};$

$ns = 3*NumInVars*NumInTerms + [(NumInVars+1)*NumRules]^2 + (NumInVars+1)*NumRules;$

$nds = 3*NumInVars*NumInTerms;$

A.1.3 "ART" Type

MaxNumRules = MaxNumInTerms;
ns = 2*NumInVars*MaxNumInTerms + [(NumInVars+1)*MaxNumRules]^2 + (NumInVars+1)*MaxNumRules + 1;
nds = 2*NumInVars*MaxNumInTerms + (NumInVars+1)*MaxNumRules;

A.2 CANFIS Parameters

A.2.1 "Scatter" Type

NumRules = NumInTerms;
ns = 3*NumInVars*NumInTerms + [(NumInVars+1)*NumRules]^2 + (NumInVars+1)*NumRules*NumOutVars;
nds = 3*NumInVars*NumInTerms;

A.2.2 "Grid" Type

NumRules = NumInTerms^{NumInVars};
ns = 3*NumInVars*NumInTerms + [(NumInVars+1)*NumRules]^2 + (NumInVars+1)*NumRules*NumOutVars;
nds = 3*NumInVars*NumInTerms;

A.2.3 "ART" Type

MaxNumRules = MaxNumInTerms;
ns = 2*NumInVars*MaxNumInTerms + [(NumInVars+1)*MaxNumRules]^2 +
+ (NumInVars+1)*MaxNumRules* NumOutVars + 1;
nds = 2*NumInVars*MaxNumInTerms + (NumInVars+1)*MaxNumRules*NumOutVars;

Appendix B

Derivation of Error Back-Propagation Algorithm for ANFIS and CANFIS (Scatter and Grid Types)

The error back-propagation algorithm is employed in order to adjust the parameters of Layer 1 via the steepest descent method. Those premise parameters (also called nonlinear parameters) are updated at each iteration (i.e. after each input-output pair is received during training) in order to minimize the following *instantaneous error function*:

$$E(n) = \sum_{m=1}^{NOV} E_m(n), \qquad m = 1,..., \text{NumOutVars.} \quad (23)$$

$$\text{where} \qquad E_m(n) = \frac{1}{2}[y_m^d(n) - Out_m^{(5)}(n)]^2$$

where $y_m^d(n)$ is the desired output and $Out_m^{(5)}(n)$ is the network's output. For each training data pair (inputs and outputs), the ANFIS operates in forward mode in order to calculate the current output $Out_m^{(5)}(n)$. Afterwards, starting from the outputs layer, and moving backwards, the error back-propagation executes to calculate the derivatives $\partial E / \partial w$ for each node at every Layer of the network as indicated in the following. Assuming that *w* is an adjustable network parameter, (e.g. a_j, b_j or c_j in (4)) then, this parameter is updated at each time step by the *steepest descent* method:

$$\Delta w \propto -\frac{\partial E}{\partial w} \quad (24)$$

and

$$w(n+1) = w(n) + \eta\left(-\frac{\partial E}{\partial w}\right). \quad (25)$$

where η ($0 < \eta < 1$) is the *learning rate* of the nets parameters. In the following we note as $\delta_i^{(j)}$ the error signal corresponding to the *i*-th node of the *j*-th layer.

Layer 5 (Output)

There is no parameter to adjust in this layer. We only have to calculate the error and back-propagate it to layer 4:

$$\delta_m^{(5)} = -\frac{\partial E(n)}{\partial Out_m^{(5)}(n)} = y_m^d(n) - Out_m^{(5)}(n), \qquad m = 1,..., \text{NumOutVars.} \quad (26)$$

Layer 4

The consequent parameters in this layer are adjusted via the RLS algorithm. Therefore back-propagation plays no adjustment role here. We only calculate the following quantity that is going to be used in the next layer:

$$\frac{\partial Out_m^{(5)}}{\partial Out_{km}^{(4)}} = 1, \ \forall \ k \tag{27}$$

Layer 3

Similarly to layer 5, here we only have to calculate the error at each node of this layer[5]:

$$\delta_k^{(3)} = -\frac{\partial E}{\partial Out_k^{(3)}} = -\sum_{m=1}^{NOV} \frac{\partial E_m}{\partial Out_k^{(3)}} = -\sum_{m=1}^{NOV} \frac{\partial E_m}{\partial Out_m^{(5)}} \frac{\partial Out_m^{(5)}}{\partial Out_{km}^{(4)}} \frac{\partial Out_{km}^{(4)}}{\partial Out_k^{(3)}} . \tag{28}$$

Taking into account eqs. (25) και (26) this can be simplified as:

$$\delta_k^{(3)} = \sum_{m=1}^{NOV} \delta_m^{(5)} \frac{\partial Out_{km}^{(4)}}{\partial Out_k^{(3)}}, \qquad k = 1,..., \text{NumRules}. \tag{29}$$

From (10) by differentiation we get:

$$\begin{aligned} \frac{\partial Out_{km}^{(4)}}{\partial Out_k^{(3)}} &= a_{1k}^m In_1^{(1)} + a_{2k}^m In_2^{(1)} + \ldots + a_{N_{Inputs}k}^m In_{N_{Inputs}}^{(1)} + a_{0k}^m \\ &= \sum_{i=1}^{N_{Inputs}} a_{ik}^m In_i^{(1)} + a_{0k}^m = f_k^m \end{aligned} . \tag{30}$$

Layer 2

$$\begin{aligned} \delta_{k_1}^{(2)} &= -\frac{\partial E}{\partial Out_{k_1}^{(2)}} = -\sum_{k_2=1}^{N_{Rules}} \frac{\partial E}{\partial Out_{k_2}^{(3)}} \frac{\partial Out_{k_2}^{(3)}}{\partial Out_{k_1}^{(2)}} = \\ &= \sum_{k_2=1}^{N_{Rules}} \delta_{k_2}^{(3)} \frac{\partial Out_{k_2}^{(3)}}{\partial Out_{k_1}^{(2)}} \end{aligned} . \tag{31}$$

and

$$\frac{\partial Out_{k_2}^{(3)}}{\partial Out_{k_1}^{(2)}} = \begin{cases} \dfrac{\sum_{i=1}^{N_{Rules}} Out_i^{(2)} - Out_{k_2}^{(2)}}{\left[\sum_{i=1}^{N_{Rules}} Out_i^{(2)}\right]^2}, & \text{εάν } k_1 = k_2 \\ -\dfrac{Out_{k_2}^{(2)}}{\left[\sum_{i=1}^{N_{Rules}} Out_i^{(2)}\right]^2}, & \text{εάν } k_1 \neq k_2 \end{cases} . \tag{32}$$

$k_1, k_2 = 1, 2, ...,$ NumRules.

[5] *NOV* = NumOutVars = Number of Output Variables.

Layer 1 (Inputs)

$$
\begin{aligned}
-\frac{\partial E}{\partial a_{ij}^{(1)}} &= -\frac{\partial E}{\partial Out_{ij}^{(1)}}\frac{\partial Out_{ij}^{(1)}}{\partial a_{ij}^{(1)}} \\
-\frac{\partial E}{\partial b_{ij}^{(1)}} &= -\frac{\partial E}{\partial Out_{ij}^{(1)}}\frac{\partial Out_{ij}^{(1)}}{\partial b_{ij}^{(1)}} \\
-\frac{\partial E}{\partial c_{ij}^{(1)}} &= -\frac{\partial E}{\partial Out_{ij}^{(1)}}\frac{\partial Out_{ij}^{(1)}}{\partial c_{ij}^{(1)}}
\end{aligned}
\tag{33}
$$

and

$$
\delta_{ij}^{(1)} = -\frac{\partial E}{\partial Out_{ij}^{(1)}} = -\sum_{k=1}^{N_{Rules}} \frac{\partial E}{\partial Out_k^{(2)}}\frac{\partial Out_k^{(2)}}{\partial Out_{ij}^{(1)}} = \sum_{k=1}^{N_{Rules}} \delta_k^{(2)}\frac{\partial Out_k^{(2)}}{\partial Out_{ij}^{(1)}} \tag{34}
$$

whereas

$$
\frac{\partial Out_k^{(2)}}{\partial Out_{ij}^{(1)}} = \begin{cases} \dfrac{Out_k^{(2)}}{Out_{ij}^{(1)}}, & \text{iff output } Out_{ij}^{(1)} \text{ is connected to } k-\text{th rule node} \\ 0 \quad , & \text{otherwise.} \end{cases} \tag{35}
$$

For the premise parameters of the fuzzy rules the following hold:

$$
\begin{aligned}
\frac{\partial Out_{ij}^{(1)}}{\partial a_{ij}^{(1)}} &= \frac{2b_{ij}^{(1)}}{a_{ij}^{(1)}} Out_{ij}^{(1)}(1-Out_{ij}^{(1)}) \\
\frac{\partial Out_{ij}^{(1)}}{\partial b_{ij}^{(1)}} &= \begin{cases} -2\ln\left|\dfrac{In_i^{(1)}-c_{ij}^{(1)}}{a_{ij}^{(1)}}\right| Out_{ij}^{(1)}(1-Out_{ij}^{(1)}), & \text{εάν } In_i^{(1)} \neq c_{ij}^{(1)} \\ 0, & \text{εάν } In_i^{(1)} = c_{ij}^{(1)} \end{cases} \\
\frac{\partial Out_{ij}^{(1)}}{\partial c_{ij}^{(1)}} &= \begin{cases} \dfrac{2b_{ij}^{(1)}}{In_i^{(1)}-c_{ij}^{(1)}} Out_{ij}^{(1)}(1-Out_{ij}^{(1)}), & \text{εάν } In_i^{(1)} \neq c_{ij}^{(1)} \\ 0, & \text{εάν } In_i^{(1)} = c_{ij}^{(1)} \end{cases}
\end{aligned}
\tag{36}
$$

i = 1, 2, ..., NumInVars
j = 1, 2, ..., NumInTerms.

In case we employ the bell-shaped membership function (5) the corresponding relations are:

$$
\begin{aligned}
\frac{\partial Out_{ij}^{(1)}}{\partial \sigma_{ij}^{(1)}} &= \frac{(In_i^{(1)}-c_{ij}^{(1)})^2}{(\sigma_{ij}^{(1)})^3} Out_{ij}^{(1)} \\
\frac{\partial Out_{ij}^{(1)}}{\partial c_{ij}^{(1)}} &= \frac{In_i^{(1)}-c_{ij}^{(1)}}{(\sigma_{ij}^{(1)})^2} Out_{ij}^{(1)}
\end{aligned}
\quad . \tag{37}
$$

And finally:

$$a_{ij}^{(1)}(n+1) = a_{ij}^{(1)}(n) + \eta\left(-\frac{\partial E}{\partial a_{ij}^{(1)}}\right)$$
$$b_{ij}^{(1)}(n+1) = b_{ij}^{(1)}(n) + \eta\left(-\frac{\partial E}{\partial b_{ij}^{(1)}}\right) \quad (38)$$
$$c_{ij}^{(1)}(n+1) = c_{ij}^{(1)}(n) + \eta\left(-\frac{\partial E}{\partial c_{ij}^{(1)}}\right)$$

i = 1, 2, ..., NumInVars
j = 1, 2, ..., NumInTerms.

Appendix C

Derivation of Error Back-Propagation Algorithm for ANFIS-ART and CANFIS-ART

The following derivation serves the adjustment of the Layer's 2 parameters $\upsilon_{ij}^{(2)}$ and $u_{ij}^{(2)}$ via the Error Back-Propagation Algorithm (EBPA), at each iteration in order to minimize the following *instantaneous error function*:

$$E(n) = \sum_{m=1}^{NOV} E_m(n),$$

where $E_m(n) = \frac{1}{2}[y_m^d(n) - Out_m^{(5)}(n)]^2$ m = 1,..., NumOutVars. (39)

Layer 6 (Output)

There is no parameter to adjust in this layer. We only have to calculate the error and back-propagate it to layer 4:

$$\delta_m^{(6)}(n) = -\frac{\partial E(n)}{\partial Out_m^{(6)}(n)} = y_m^d(n) - Out_m^{(6)}(n),\ m = 1,..., \text{NumOutVars.} \quad (40)$$

Layer 5

The consequent parameters in this layer are adjusted via the RLS algorithm. Therefore back-propagation plays no adjustment role here. We only calculate the following quantity that is going to be used in the next layer:

$$\frac{\partial Out_m^{(6)}}{\partial Out_{km}^{(5)}} = 1, \qquad k = 1, 2, \ldots, \text{NumRules.} \quad (41)$$

Layer 4

Similarly to layer 5, here we only have to calculate the error at each node of this layer:

$$\delta_k^{(4)} = -\frac{\partial E}{\partial Out_k^{(4)}} = -\sum_{m=1}^{NOV} \frac{\partial E_m}{\partial Out_k^{(4)}} = -\sum_{m=1}^{NOV} \frac{\partial E_m}{\partial Out_m^{(6)}} \frac{\partial Out_m^{(6)}}{\partial Out_{km}^{(5)}} \frac{\partial Out_{km}^{(5)}}{\partial Out_k^{(4)}}. \quad (42)$$

Taking into account eqs. (25) και (26) this can be simplified as:

$$\delta_k^{(4)} = \sum_{m=1}^{NOV} \delta_m^{(6)} \frac{\partial Out_{km}^{(5)}}{\partial Out_k^{(4)}}. \qquad k = 1,..., \text{NumRules.} \quad (43)$$

From (20) by differentiation we get:

$$\frac{\partial Out_{km}^{(5)}}{\partial Out_k^{(4)}} = a_{1k}^m \overline{In_1^{(1)}} + a_{2k}^m \overline{In_2^{(1)}} + \ldots + a_{N_{Inputs}k}^m \overline{In_{N_{Inputs}}^{(1)}} + a_{0k}^m$$
$$= \sum_{i=1}^{N_{Inputs}} a_{ik}^m \overline{In_i^{(1)}} + a_{0k}^m = \overline{f_k^m} \quad . \tag{44}$$

Layer 3

$$\delta_{k_1}^{(3)} = -\frac{\partial E}{\partial Out_{k_1}^{(3)}} = -\sum_{k_2=1}^{N_{Rules}} \frac{\partial E}{\partial Out_{k_2}^{(4)}} \frac{\partial Out_{k_2}^{(4)}}{\partial Out_{k_1}^{(3)}} =$$
$$= \sum_{k_2=1}^{N_{Rules}} \delta_{k_2}^{(4)} \frac{\partial Out_{k_2}^{(4)}}{\partial Out_{k_1}^{(3)}} \quad . \tag{45}$$

and

$$\frac{\partial Out_{k_2}^{(4)}}{\partial Out_{k_1}^{(3)}} = \begin{cases} \dfrac{\sum_{i=1}^{N_{Rules}} Out_i^{(3)} - Out_{k_2}^{(3)}}{\left[\sum_{i=1}^{N_{Rules}} Out_i^{(3)}\right]^2}, & \text{εάν } k_1 = k_2 \\ -\dfrac{Out_{k_2}^{(3)}}{\left[\sum_{i=1}^{N_{Rules}} Out_i^{(3)}\right]^2}, & \text{εάν } k_1 \neq k_2 \end{cases} \quad . \tag{46}$$

k_1 ,k_2 = 1, 2, ..., NumRules.

Layer 2

$$-\frac{\partial E}{\partial \upsilon_{ij}^{(2)}} = -\frac{\partial E}{\partial Out_{ij}^{(2)}} \frac{\partial Out_{ij}^{(2)}}{\partial \upsilon_{ij}^{(2)}}$$
$$-\frac{\partial E}{\partial u_{ij}^{(1)}} = -\frac{\partial E}{\partial Out_{ij}^{(2)}} \frac{\partial Out_{ij}^{(2)}}{\partial u_{ij}^{(2)}} \tag{47}$$

and

$$\delta_{ij}^{(2)} = -\frac{\partial E}{\partial Out_{ij}^{(2)}} = -\frac{\partial E}{\partial Out_{k=j}^{(3)}} \frac{\partial Out_{k=j}^{(3)}}{\partial Out_{ij}^{(2)}} = \delta_k^{(3)} \frac{\partial Out_{k=j}^{(3)}}{\partial Out_{ij}^{(2)}} \tag{48}$$

whereas

$$\frac{\partial Out_{k=j}^{(3)}}{\partial Out_{ij}^{(2)}} = \frac{Out_{k=j}^{(3)}}{Out_{ij}^{(1)}} \tag{49}$$

For the premise parameters of the fuzzy rules the following hold:

$$\frac{\partial Out_{ij}^{(2)}}{\partial \upsilon_{ij}^{(2)}} == \begin{cases} \gamma, \text{ if } 0 \le (In_{ij}^{(2)} - \upsilon_{ij}^{(2)})\gamma \le 1 \\ 0, \text{ otherwise} \end{cases}$$
$$\frac{\partial Out_{ij}^{(2)}}{\partial u_{ij}^{(2)}} == \begin{cases} -\gamma, \text{ if } 0 \le (u_{ij}^{(2)} - In_{ij}^{(2)})\gamma \le 1 \\ 0, \text{ otherwise} \end{cases} \quad (50)$$

And finally:

$$\upsilon_{ij}^{(2)}(n+1) = \upsilon_{ij}^{(2)}(n) + \eta\left(-\frac{\partial E}{\partial \upsilon_{ij}^{(2)}}\right)$$
$$u_{ij}^{(2)}(n+1) = u_{ij}^{(2)}(n) + \eta\left(-\frac{\partial E}{\partial u_{ij}^{(2)}}\right) \quad (51)$$

i = 1, 2, ..., NumInVars
j = 1, 2, ..., NumInTerms.

References

[1] Jang R. J.-S., Sun C.-T., Mizutani E., (1997), “Neuro-Fuzzy and Soft Computing, A Computational Approach to Learning and Machine Intelligence”, Prentice Hall, Upper Saddle River, NJ 07458.

[2] Carpenter G.A., Grossberg S., Rosen B.D., “Fuzzy ART: Fast Stable Learning and Categorization of Analog Patterns by an Adaptive Resonance System”, *Neural Networks*, vol. 4, pp. 759-771, 1991.

[3] C.J. Lin, C.T. Lin, "An ART-Based Fuzzy Adaptive Learning Control Network", *IEEE Transactions on Fuzzy Systems*, vol. 5, No. 4, Nov. 1997.

[4] A. Garrett, "Fuzzy ART and Fuzzy ARTMAP Neural Networks", 22-12-03, http://www.mathworks.com/matlabcentral/fileexchange/4306-fuzzy-art-and-fuzzy-artmap-neural-networks