Fuzzy Inference Systems and Their Corresponding Neural Networks

ANDREAS ZOLLMANN
zollmann@gmx.de

Dresden, June 12, 2002

Abstract

This seminar paper introduces the concept of fuzzy inference and explains two different types of neural networks - viz., Radial Basis Function Networks (RBFN) and Evolving Fuzzy Neural Networks (EFuNN) - that make use of this idea. A comparison suggests that EFuNN seems to be the superior approach.

Contents

1 Introduction

Modelling knowledge or physical behavior with conventional methods like predicate logic or differential equations often becomes difficult, if not all aspects of the system to be modelled are known. In this paper, a possible way out - Fuzzy inference systems - is shown. Later we will be concerned with neural networks that can model such systems. Radial Basis Function Networks (RBFNs) - which evolved independently of Fuzzy Inference Systems - are introduced, and its equivalence to certain Fuzzy Inference Systems, as described in [1], is shown. Finally, Evolving Fuzzy Neural Networks (EFuNNs) are elaborated, quite a new approach, that seems to overcome some of the stiffness of other NN architectures, and is particularly suited for online learning tasks.

2 Fuzzy Inference

A simple fuzzy inference rule concerning the physical behavior of gases would be

If temperature is high then volume is high,

where temperature and volume are linguistic variables being applied to appropriate membership functions characterising the predicate high.

Think of a body that has fallen a distance of Δh with no start velocity and has now a velocity of v. In the case that Δh is small, gravitation can be assumed as constant during the fall. Further, if v is small as well, air resistance can be disregarded, such that we get $v = \sqrt{2gh}$, where g is the gravitational constant. This could be captured by a fuzzy rule like this one:

If v is small and Δh is small, then $v = \sqrt{2gh}$.

Here, the consequent part is a non-fuzzy equation.

The rules stated above are of types (3) and (4) (extended by allowing for the square root), respectively, in the following classification of Fuzzy inference rules, introduced in [3]. All those rules have in common, that their conditions depend on a vector $(x_1, \ldots, x_n)$ of real-valued input variables, while the rule's output y $\in \mathbb{R}$ is one-dimensional. How inference functions representing the rules could exactly be defined, will be treated in the next subsection.

(1) Propositional rules, e.g., IF x1 is true AND x2 is false, THEN y is false

Although this rule uses only Boolean truth values, it represents a fuzzy inference function. The input variables x1 and x2 and the output y are reals from the interval [0;1] and state a degree of truth; "0" meaning "absolutely false", and "1" meaning "absolutely true". For instance, if x1=0.9 and x2=0.1, the inference function should map y to a value near, but not exactly equal to 0.

(2) Propositional rules with degrees of importance (DI) and certainty factors (CF), eg., IF x1 is true (DI 3.4) AND x2 is false (DI 1.2), THEN y is false (CF 8.5)

The DIs represent the importance of each condition element for obtaining the rule output, whilst the CF represents the strength of the rule relatively to other concurring fuzzy rules with similar conditions.

(3) Zadeh-Mamdani rules, e.g., IF x1 is SMALL OR x2 is HEAVY, then y is LARGE.

In order to accomplish a comparison of the input values x1 and x2 with the fuzzy predicates SMALL and HEAVY, each predicate is represented in the inference system by a membership function from the Real numbers into the interval [0;1], stating for each possible value of the input component a degree of "belonging to the predicate". Applying the input values to the respective membership functions is called fuzzification. Now the fuzzified conditional parts have to be applied to the OR-operator in order to obtain the firing strength (weight) of the rule. Depending on this firing strenth, the (fuzzy) consequent of the rule is generated, i.e. a value in [0;1] representing the belonging of y to LARGE is obtained. Under possible competition of other rules, this value and the other rules' suggestions must be merged to a single fuzzy output value, which then is defuzzified using the membership function for LARGE, and so the final crisp output is yielded. The steps above will be explained in more detail later on.

(4) Takagi-Sugeno rules, e.g., IF x1 is SMALL AND x2 is HEAVY, then y is $a \cdot x1 + b \cdot x2 + c$.

This rule is similar to (3), however, the output y is directly (without a defuzzification process) calculated by a linear function on the input variables (a, b and c are constants).

(5) Zadeh-Mamdani rules with DIs and CFs - see also rules (2) and (3). Example: IF x1 is SMALL (DI 0.5) AND x2 is HEAVY (DI 3), then y is LARGE (CF 2.3).

(6) Cluster rules of the form: IF input vector x belongs to an input cluster defined by its center and radius, THEN y belongs to an output cluster

defined by its center and radius. We will come back to them in the section on EFuNNs.

(7) Temporal rules of the form: IF x1 is present at a time moment t1 (with a degree of importance of DI1) AND/OR x2 is present at a time moment t2 (with a degree of importance DI2), THEN y is C (certainty factor CFc)

(8) Temporal, recurrent rules: IF x1 is A (DI1) AND/OR x2 is B (DI2) AND/OR y at time moment (t-k) is C, THEN y at time moment (t+n) is D (CFd)

2.1 Architecture and Behaviour of a Fuzzy Inference System

A Fuzzy Inference System is composed of five functional blocks [2]:

- a rule base containing the fuzzy rules
- a database defining the membership functions
- a decision-making unit which performs the inference operations on the rules
- a fuzzification interface that transforms each real-valued component of the input vector into degrees of match with the corresponding conditional predicate using the respective membership function
- a defuzzification interface transforming the fuzzy result of the inference into a real-valued output

A membership function for a Fuzzy predicate (also called: *linguistic label*) P assigns a real-valued input component x_i a degree of "being P" reaching from 0 (*not* $P(x_i)$) to 1 ($P(x_i)$). Typically either a strictly monotonic function, e.g. of the form

$$\frac{1}{\pi}\arctan[t(x_i - c)] + \frac{1}{2} \tag{1}$$

(strictly increasing and point-symmetric to $x_i = c$ yielding the membership value $\frac{1}{2}$ and the slope $\frac{t}{\pi}$) resp.

$$\frac{-1}{\pi}\arctan[t(x_i - c)] + \frac{1}{2} \tag{2}$$

(strictly *decreasing* with acceleration t and center $x_i = c$), or a bell-shaped Gauss-like function favoring some value $x_i = c$ as best fit, like

$$\frac{1}{1 + (\frac{x_i - c}{\sigma})^{2b}} \tag{3}$$

or

$$\exp\left[-\left(\frac{x_i - c}{\sigma}\right)^2\right] \tag{4}$$

is used. For computability reasons, piecewisely linear functions approximating the bell with a triangle or trapezoid, are also used in practice. The parameters of the membership functions are local and modified within each rule by the learning algorithm.

Fuzzy reasoning now works as follows: Assume an input vector $x \in \mathbb{R}^n$ is given (if using the temporal rules (7) and (8), this vector would include possibly needed earlier output values). To make things simpler, let us also assume, that no OR-operator is allowed in the rule premises. This is no great limitation, since a rule of the form "IF F1 OR F2 THEN G" can be split into two rules "IF F1 THEN G" and "IF F2 THEN G" and the properties of the wanted Or-operator can be preserved by properly choosing the final output generation function (s. a. the paragraph below Step 3). Now, for each rule r_j (j = 1, ..., m) in the rule base, an intermediate output o_j is calculated:

(Step 1) Fuzzification: Apply the x-components stated in the premise part of the rule to the membership functions of the corresponding predicates.

For rules of type (1) or (2), we have "is true" or "is false" instead of predicates. Those rules expect the inputs already to be fuzzy values in [0;1]. Hence, for conditional parts of the form "xi is true", x_i is returned itself, and for "xi is false", $1 - x_i$ is calculated. Actually, one could regard rules of type (1) and (2) as particular type-(3) and -(5) rules, respectively, where "is true" is assigned the identity and "is false" $\lambda x.(1 - x)$ as membership function and no alteration of those functions during learning is allowed.

(Step 2) Combination: The firing strength w_j of rule r_j is obtained from the single fuzzified x-components by some AND-operator. If degrees of importance (DIs) were given in the conditional parts, the AND-operator should use them for weighting the conjunctive elements. Certainty factors are multiplied with the result of the AND-operation.

Typically, the AND-operator is the multiplication or minimum function. If DIs are used, it could for example be defined as:

$$AND(c_1, \ldots, c_k) = \prod_{i=1}^{k} c_i^{DI_i} \tag{5}$$

(Step 3) Defuzzification: Generate the intermediate output o_j by applying r_j's firing strength w_j obtained in (Step 2) to the inverse of the membership function of the rule consequent's Fuzzy predicate. To make this work, of course, this function has to be bijective. In particular, consequent predicates with bell-shaped membership functions cannot be used!

For rules of type (1) or (2), we treat "is false" and "is true" as predicates as explained in (Step 1). In type-(4) rules, o_j depends on the input vector instead of w_j as explained above in the description of those rules.

Having calculated w_j and o_j for each rule r_j, the final output needs to be generated. Either, a winner-takes-all strategy is used, setting

$$y = o_k \text{ with } k = \arg\min_{j=1}^{m} w_j, \tag{6}$$

or, paying more homage to the fuzzy spirit, no rule's work is neglected and y is the weighted average of all o_j's:

$$y = \frac{\sum_{j=1}^{m} w_j o_j}{\sum_{j=1}^{m} w_j} \tag{7}$$

3 Radial Basis Function Networks (RBFNs)

RBFNs were proposed by Moody and Darken and are derived from observations of biological receptive fields, e.g. the visual cortex [1]. Fig. 1 gives a diagram of an RBFN with five receptive units. Each unit u_j is (possibly multiply) connected to some of the *Real*-valued input nodes x_1, ..., x_n (fixed, predefined connections) and sends its output $w_j \in \mathbb{R}$ to a single node u_{out} that computes the final output $y \in \mathbb{R}$ of the network.

The output w_j (also called "strength") of u_j is obtained by applying the incoming vector $x^j = (x_{j_1}, ..., x_{j_{in(j)}})$ to the local *receptive field response* R_j ($j_i \in \{1, ..., n\}$, $in(j)$ denotes the number of incoming connections to unit u_j). If the network undergoes a learning process, the R_j's can change according to a learning rule. Usually, R_j is chosen as bell-shaped function

$$R_j(x) = \exp(-\frac{||x^j - c^j||^2}{\sigma_j^2}), \tag{8}$$

where $c^j \in \mathbb{R}^{in(j)}$ and $\sigma_j \in \mathbb{R}$ are parameters adaptable during learning. For the special case $n = 1$, this function was already stated in equation (4). The center of the bell is at point $x = c^j$. There, R_j yields 1, the greatest possible value. The parameter σ_j characterises the thickness of the bell. The larger $|\sigma_j|$ is chosen, the slower R_j decreases when x is moved away from the center c^j.

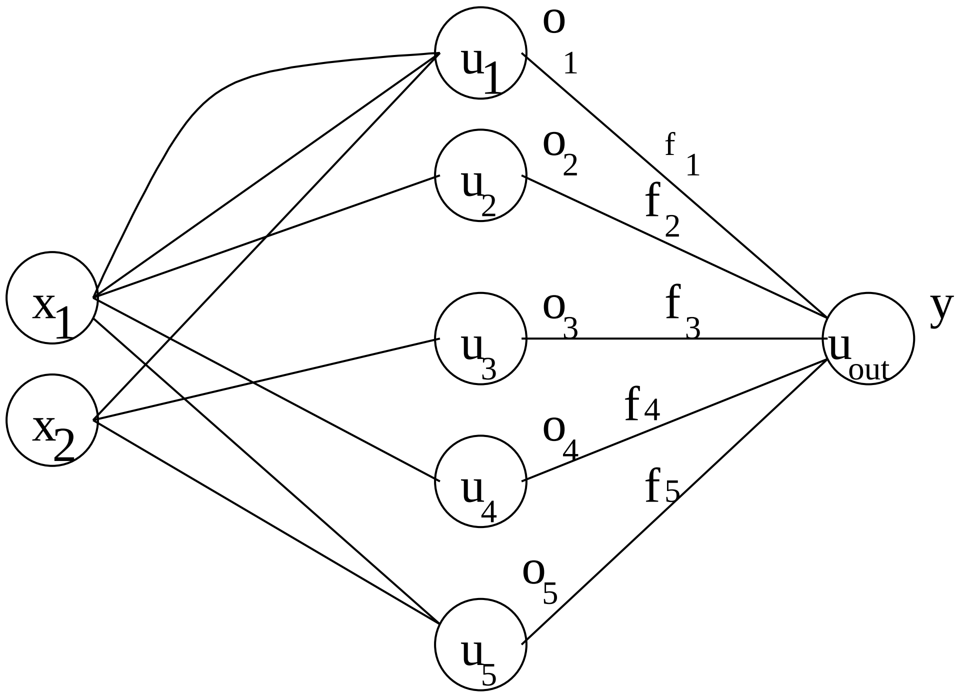

Figure 1: Example RBFN

For obtaining the final output y, a parameter f_j is associated with each connection from u_j to u_{out}. Now, u_{out} either returns the weighted sum

$$y = \sum_{j=1}^{m} w_j f_j \tag{9}$$

of those parameters, where m is the number of receptive units, or a weighted average (referred to as *normalized response function* in [1]):

$$y = \frac{\sum_{j=1}^{m} w_j f_j}{\sum_{j=1}^{m} w_j} \tag{10}$$

3.1 Equivalence to Fuzzy Inference Systems

As shown in [1], inference systems using rules of type (4), where the parameters a and b are zero (i.e., rule r_j's output o_j is just a constant), can be modelled by RBFNs in a very straigtforward way. The receptive units correspond to the inference rules, the network's parameters f_j to the (constant) rule outputs o_j, and the receptive field responses R_j have to be chosen as:

$$R_j = (M_{j,1} \times \cdots \times M_{j,in(j)}); \mathrm{And}_j \tag{11}$$

Here, ";" means function composition, $M_{j,i}$ is the membership function for the (i)th fuzzy predicate in the condition of the inference system's rule r_j, and And_j is the And-Operator of the rule. For practical reasons, the membership functions should be kept uniform (i.e., Gaussian functions as in equation (4) with varying parameters c and σ) in order to get simple response functions.

To obtain the And-operators and membership functions out of the response functions, the R_j's are decomposed arbitrarily according to equation (11).

Example. Assume, the fuzzy inference system has the following two rules

R1: IF x1 is SMALL AND x1 is MEDIUM AND x3 is LARGE, THEN y = 3
R2: IF x3 is SMALL, THEN y = 1

Let us use the membership functions

$$M_{1,1}(x) = M_{2,1}(x) = \exp\left[-\left(\frac{x-(-1)}{3}\right)^2\right] \quad \text{(SMALL)} \tag{12}$$

$$M_{1,2}(x) = \exp\left[-\left(\frac{x}{3}\right)^2\right] \quad \text{(MEDIUM)} \tag{13}$$

$$M_{1,3}(x) = \exp\left[-\left(\frac{x-1}{3}\right)^2\right] \quad \text{(LARGE)} \tag{14}$$

and multiplication as And-operator. Then the receptive field responses of the corresponding RBFN are

$$R_1(x^1) = e^{-\frac{(x_1^1-(-1))^2}{3^2}} e^{-\frac{(x_2^1-0)^2}{3^2}} e^{-\frac{(x_3^1-1)^2}{3^2}} = e^{-\frac{||x^1-(-101)||^2}{3^2}} \tag{15}$$

$$R_2(x^2) = e^{-\frac{(x_1^2-(-1))^2}{3^2}} = e^{-\frac{||x^2-(-1)||^2}{3^2}}, \tag{16}$$

where $x^1 = (x_1 x_1 x_3)$ (i.e., we have a double-connection from x_1 to u_1) and $x^2 = (x_3)$. The obtained field responses are equations of form (8).

The equivalence between RBFNs and certain types of inference systems enables us to apply results obtained in one field of research to the other one. For example, learning rules for inference systems can be carried over to RBFNs and vice versa.

3.2 Adaptive-Network-Based Fuzzy Inference System (ANFIS)

For modelling general type-(4) inference systems, extended RBFNs called Adaptive-Network-Based Fuzzy Inference Systems (ANFIS) are used. Essentially, the architecture contains two further layers, which obtain the weighted averages of the linear factors of the rule consequents and multiply those factors with the input components. A detailed description of ANFIS and its learning mechanism can be found in [2].

4 Evolving Fuzzy Neural Networks (EFuNNs)

EFuNNs were introduced by Kasabov [3]. Unlike common neural networks, they have a local one-pass learning strategy and are thus claimed to be "several orders of magnitudes faster than ... traditional models". Kasabov proposes them for online-learning tasks like financial time series or biological process control, where real-time data steadily arrives and quickly has to be aggregated into the network in order to guess future data and act accordingly. To keep the size of the network under control, nodes with a small amount of traffic can be pruned and similarly behaving nodes are merged together. The inference rules modelled by EFuNNs are those of type (6) (s. a. Section 2).

4.1 Architecture

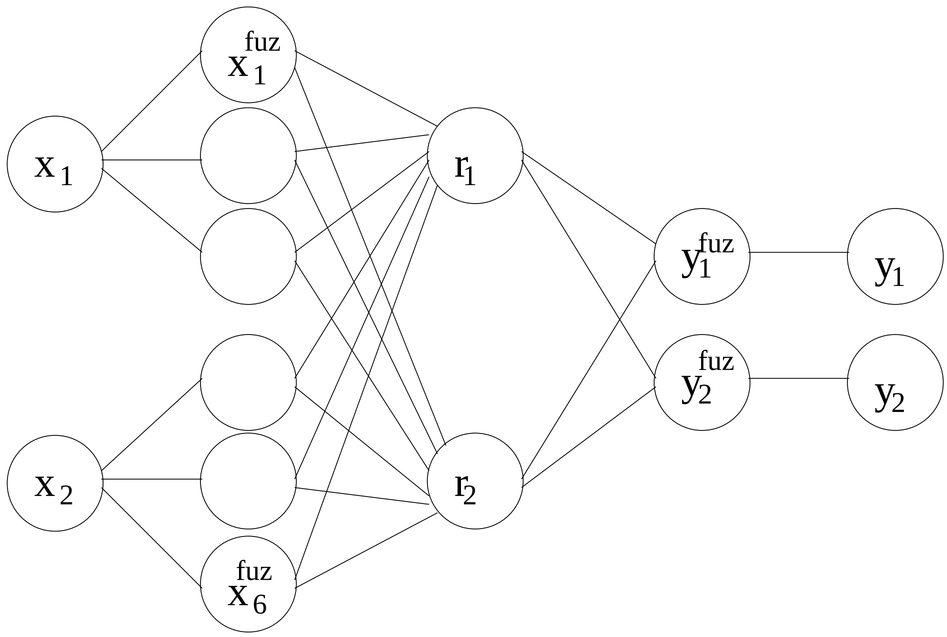

Figure 2: Example RBFN

EFuNNs consist of five layers as indicated in Fig. 2. Similarly to RBFNs, the second layer represents the fuzzified *Real* input values coming in from the first layer. For each pair of Fuzzy predicates and input components, a unit is set up in the second layer, which computes the degree of membership of that input component to the predicate. The connections from the fourth to the last (output) layer create *Real*-values out of the fuzzy outputs. The number of units in both layers is the same, and each unit in one layer is connected to exactly one unit in the other layer. In the middle, we have the rule layer. Initially, the rule layer can be either empty or contain nodes with predefined standard connections.

4.2 Basic Principles

Let N_j denote the number of rule conditions, i.e., connections to r_j from the Fuzzy input layer, and N_{out} the number of components of the output vector y. Each node r_j of the rule layer is defined by two vectors of connection weights, $W1(r_j) \in [0;1]^{N_j}$ and $W2(r_j) \in [0;1]^{N_{out}}$, and a sensitivity threshold parameter S_j.

$W1(r_j)$ represents the center of a sphere in which the fuzzified inputs stated in the rule condition have to lie to be considered by the rule. The sphere's radius is given by $R_j = 1 - S_j$. The metric we use, however, is not the usual Euklidean one, but the so called *normalised fuzzy distance*

$$D(x^1, x^2) = \frac{||x^1 - x^2||}{||x^1 + x^2||}, \text{ where } x^1, x^2 \in [0;1]^{N_j}. \tag{17}$$

Above, $||v||$ denotes the sum of the absolute values of v's components. Note that $D(x^1, x^2) \leq 1$, since $x^1, x^2 \geq 0$.

$W2(r_j)$ represents the fuzzified output vector suggested by rule r_j for inputs within its reach. Now for all rule nodes r_j whose input cluster was activated by the fuzzified input vector x^{fuz}, the activation $A_j(x^{fuz})$ is calculated:

$$A_j(x^{fuz}) = 1 - D\left(x^{fuz}, W1(r_j)\right) \tag{18}$$

Either all activated nodes or just the m nodes with the highest activation (usually $m = 3$) are involved in the actual output process, in which the nodes of the fuzzy output layer are assigned the weighted average of the involved rule nodes' output suggestions:

$$(y^{fuz})_i = \frac{\sum_j A_j(x^{fuz}) \cdot [W2(r_j)]_i}{\sum_j A_j(x^{fuz})} \quad (i = 1, \ldots, N_{out}) \ , \tag{19}$$

where the summation is performed over the involved rule nodes r_j. If x^{fuz} did not reach into any rule's input cluster, an error flag is set.

As with Fuzzy Inference Systems, in order for the defuzzification process yielding the *Real* from the Fuzzy outputs to work, the membership functions from *Real* outputs to Fuzzy outputs have to be inversed and hence must be bijective.

4.3 Learning

Each time a new data pair becomes available, it can be taught to the network during runtime. Assume, the in- / output pair (x, z) with the respective fuzzified vectors (x^{fuz}, z^{fuz}) is to be accumulated. At first, x is propagated through the net as explained above and the fuzzified system output y^{fuz} is obtained. If the error flag was set, i.e., no rule got activated by x, or the *normalised Fuzzy output error*

$$Err(y^{fuz}, z^{fuz}) = \frac{||y^{fuz} - z^{fuz}||}{N_{out}} \tag{20}$$

is greater than a global Error threshold E, then a new rule is created with $W1 = x^{fuz}$, $W2 = z^{fuz}$ and its sensitivity threshold S initialised with a standard value. Otherwise, all rules r_j that were involved in the output process (s. a. previous subsection) change their parameters in the following way:

$$W1^{(new)}(r_j) = W1(r_j) + L_j \cdot (x^{fuz} - W1(r_j) \tag{21}$$

$$W2^{(new)}(r_j) = W2(r_j) + L_j A_j \cdot (z^{fuz} - W2(r_j)) \tag{22}$$

$$S_j^{(new)} = S_j - D(W1(r_j), W1^{(new)}(r_j)) \tag{23}$$

Above, $L_j \in [0; 1]$ is the learning rate for unit r_j. The greater L_j, the quicker the unit "forgets" previously learned examples and adjusts to new ones. If the number of learned examples $N_{ex}(r_j)$ is stored for each rule node, L_j could be defined as $1/N_{ex}(r_j)$, with the rationale that the more examples are represented by the unit, the more important is its current position and the less it should move. The update for S_j causes the cluster radius R_j to change as follows:

$$R_j^{(new)} = R_j + D(W1(r_j), W1^{(new)}(r_j)) \tag{24}$$

This ensures that input vectors that previously lay within the rule's sphere remain inside.

4.4 Pruning and Aggregation

To keep the size of the EFuNN under a certain limit, neurons and connections can be pruned or aggregated. A *pruning* rule could for example look as follows [3]:

> IF Age(r_j) > OLD AND [number of times r_j was involved in an output process] < Pr $\cdot$ Age(r_j) THEN [remove rule node r_j and its connections]

Here, Age(r_j) is the number of examples presented to the EFuNN since the creation of r_j, OLD is a predefined age limit, and Pr $\in [0; 1]$ is a pruning parameter.

Aggregation is the process of merging several rule nodes that are close to each other in the problem space into one. This enables generalisation. The aggregation system selects from time to time an arbitrary rule node r_j and now forms two subsets of nodes: Pos(r_j) containing all rule nodes r_p with $Err(W2(r_j), W2(r_p)) \leq E$ and Neg(r_j) containing the other rule nodes. All rule nodes from Pos(r_j) that are closer to r_j in the input space than the closest node to r_j from Neg(r_j), i.e., all $r_p \in Pos(r_j)$ satisfying

$$\forall_{r_n \in Neg(r_j)}.D(W1(r_j), W1(r_p)) < D(W1(r_j), W1(r_n)),$$

are aggregated, if the resulting radius of the new rule r_{agg} does not exceed a pre-defined limit R_{max} for a receptive field. The parameters for r_{agg} are set as follows:

$$N_{ex}(r_{agg}) = \sum_i N_{ex}(r_i) \tag{25}$$

$$W1(r_{agg}) = \frac{\sum_i N_{ex}(r_i) \cdot W1(r_i)}{N_{ex}(r_{agg})} \tag{26}$$

$$W2(r_{agg}) = \frac{\sum_i N_{ex}(r_i) \cdot W2(r_i)}{N_{ex}(r_{agg})} \tag{27}$$

$$R_{agg} = \max_i \{D(W1(r_{agg}), W1(r_j)) + R_j\}, \tag{28}$$

where the indices i run over the rules r_i to be aggregated.

4.5 EFuNNs in practise

Kasabov and his research team from the University of Otago, New Zealand, have programmed EFuNNs in *MathLab*. The source code is freely available at:

http://www.commerce.otago.ac.nz/infosci/KEL/CBIIS.html

On this site, also a Java Applet simulating an EFuNN can be found.

5 Conclusions

Inference systems and networks that make use of Fuzzy logic are a powerful tool for representing limited knowledge in a functional way and for modelling functions of which only a small finite data-set is known.

While non-fuzzy inference systems cannot interpolate between Boolean truth values and thus are unable to derive answers that only hold to a certain extent or with a certain probability, but might still be more satisfying than no answer at all, neural network approaches are better suited for these kind of tasks. However, typical neural networks, like multi-layer perceptrons and also standard RBFNs, use global learning strategies and hence suffer from long training times and high forgetting of formerly-trained data. EFuNNs can overcome these problems by their flexible architecture, high locality and a one-pass (and thus fast) learning mechanism.

References

[1] J.-S. Roger Jang, C.-T. Sun. Functional Equivalence between Radial Basis Function Networks and Fuzzy Inference Systems. *IEEE Transactions on Neural Networks.* Vol. 4, No. 1, pp. 156-159, 1993.

[2] J.-S. Roger Jang. ANFIS: Adaptive-Network-Based Fuzzy Inference Systems. *IEEE Transactions on Systems, Man, and Cybernetics.* Vol. 23, No. 3, pp. 665-685, 1993.

[3] N. K. Kasabov. On-line learning, reasoning, rule extraction and aggregation in locally optimized evolving fuzzy neural networks. *Neurocomputing.* Vol. 41, pp. 25-45, 2001.