# Neuro–Fuzzy Systems for Function Approximation

Detlef Nauck and Rudolf Kruse
Faculty of Computer Science, Neural and Fuzzy Systems
Otto–von–Guericke–University of Magdeburg
Universitaetsplatz 2, D-39106 Magdeburg, Germany
Tel.: +49.391.67.12700, Fax: +49.391.67.12018
Email: nauck@iik.cs.uni-magdeburg.de

**Abstract**

We propose a neuro–fuzzy architecture for function approximation based on supervised learning. The learning algorithm is able to determine the structure and the parameters of a fuzzy system. The approach is an extension to our already published NEFCON and NEFCLASS models which are used for control or classification purposes. The proposed extended model, which we call NEFPROX, is more general and can be used for any application based on function approximation.

**Keywords:** neuro–fuzzy system, function approximation, structure learning, parameter learning

## 1 Introduction

Certain fuzzy systems are universal function approximators [1, 4]. In order to identify a suitable fuzzy system for a given problem, membership functions (parameters) and a rule base (structure) must be specified. This can be done by prior knowledge, by learning, or by a combination of both. If a learning algorithm is applied that uses local information and causes local modifications in a fuzzy system, this approach is usually called neuro–fuzzy system [7].

We have already presented two neuro–fuzzy approaches NEFCON [5] and NEFCLASS [6, 8]. The first one is used for control applications, and is trained by reinforcement learning based on a fuzzy error measure. The second one is used for classification of data, and is based on supervised learning. Both models can do structure and parameter learning by using a learning procedure called fuzzy error backpropagation. The term "backpropagation" denotes that learning is done by determining error signals, and propagating them backwards through the system architecture to compute local parameter modifications. This is not a gradient descent method like in neural networks, but a simple heuristic procedure, because the functions involved in the system are usually not differentiable.

The term "fuzzy error" denotes that the error measure guiding the learning process is either specified by a fuzzy rule base (NEFCON) or by a fuzzy set over the differences between actual and desired outputs (NEFCLASS). If the error is specified by a rule base describing a desired state, e.g. of a controlled task, then we have a special case of supervised learning, i.e. reinforcement learning. On the other hand, if there is information about the correct output value, then we use plain supervised learning.

In this paper we propose a general approach to function approximation by a neuro–fuzzy model based on plain supervised learning. This approach has a similar structure as the NEFCON model, but it is an extension, because it does not need reinforcement learning. On the other hand is also extends the NEFCLASS model, that can only be used for crisp classification tasks. We call this approach NEFPROX (NEuro Fuzzy function apPROXimator).

# 2 Architecture

Function approximation based on local learning strategies is a domain of neural networks [2] and neuro–fuzzy systems [7]. Neuro–fuzzy systems have the advantage that they can use prior knowledge, whereas neural networks have to learn from scratch. In addition neural networks are black boxes, and they can usually not be interpreted in form of rules. A well known neuro–fuzzy system for function approximation is the ANFIS model [3]. However there is no algorithm given for structure learning, and it is used to implement Sugeno models with differentiable functions (e.g. product as $t$–norm). We propose a more general approach that can also implement Mamdani type fuzzy systems. The model is like NEFCON and NEFCLASS based on a generic fuzzy perceptron [7].

**Definition 1** *A* **3–layer generic fuzzy perceptron** *is a 3–layer feedforward neural network* $(U, W, \mathrm{NET}, A, O, \mathrm{ex})$ *with the following specifications:*

*(i)* $U = \bigcup_{i \in M} U_i$ *is a non–empty set of units (neurons) and* $M = \{1, 2, 3\}$ *is the index set of* $U$*. For all* $i, j \in M, U_i \neq \emptyset$ *and* $U_i \cap U_j = \emptyset$ *with* $i \neq j$ *holds.* $U_1$ *is called input layer,* $U_2$ *rule layer (hidden layer), and* $U_3$ *output layer.*

*(ii) The structure of the network (connections) is defined as* $W : U \times U \to \mathcal{F}(\mathbb{R})$*, such that there are only connections* $W(u, v)$ *with* $u \in U_i$, $v \in U_{i+1} (i \in \{1, 2\})$ *(*$\mathcal{F}(\mathbb{R})$ *is the set of all fuzzy subsets of* $\mathbb{R}$*).*

*(iii)* $A$ *defines an activation function* $A_u$ *for each* $u \in U$ *to calculate the activation* $a_u$

  *(a) for input and rule units* $u \in U_1 \cup U_2$*:* $A_u : \mathbb{R} \to \mathbb{R}$, $a_u = A_u(\mathrm{net}_u) = \mathrm{net}_u$,

  *(b) for output units* $u \in U_3$*:* $A_u : \mathcal{F}(\mathbb{R}) \to \mathcal{F}(\mathbb{R})$, $a_u = A_u(\mathrm{net}_u) = \mathrm{net}_u$.

*(iv)* $O$ *defines for each* $u \in U$ *an output function* $O_u$ *to calculate the output* $o_u$

  *(a) for input and rule units* $u \in U_1 \cup U_2$*:* $O_u : \mathbb{R} \to \mathbb{R}$, $o_u = O_u(a_u) = a_u$,

  *(b) for output units* $u \in U_3$*:* $O_u : \mathcal{F}(\mathbb{R}) \to \mathbb{R}$, $o_u = O_u(a_u) = \mathrm{DEFUZZ}_u(a_u)$, *where* $\mathrm{DEFUZZ}_u$ *is a suitable defuzzification function.*

(v) NET *defines for each unit* $u \in U$ *a propagation function* $\mathrm{NET}_u$ *to calculate the net input* $\mathrm{net}_u$

(a) *for input units* $u \in U_1$*:* $\mathrm{NET}_u : \mathbb{R} \to \mathbb{R}$, $\mathrm{net}_u = ex_u$,

(b) *for rule units* $u \in U_2$*:* $\mathrm{NET}_u : (\mathbb{R} \times \mathcal{F}(\mathbb{R}))^{U_1} \to [0,1]$, $\mathrm{net}_u = \underset{u' \in U_1}{\top} \{W(u',u)(o_{u'})\}$, *where* $\top$ *is a t–norm,*

(c) *for output units* $u \in U_3$*:* $\mathrm{NET}_u : ([0,1] \times \mathcal{F}(\mathbb{R}))^{U_2} \to \mathcal{F}(\mathbb{R})$, $\mathrm{net}_u : \mathbb{R} \to [0,1]$, $\mathrm{net}_u(x) = \underset{u' \in U_2}{\perp} \{\top(o_{u'}, W(u',u)(x))\}$, *where* $\perp$ *is a t–conorm.*

(vi) $\mathrm{ex} : U_1 \to \mathbb{R}$, *defines for each input unit* $u \in U_1$ *its external input* $\mathrm{ex}(u) = \mathrm{ex}_u$. *For all other units* ex *is not defined.*

A generic fuzzy perceptron can be viewed as a 3–layer neural network with special activation and propagation functions, and fuzzy sets as weights. On the other hand it can also be viewed as a fuzzy system represented in a neural–network–like architecture. To obtain NEFCON or NEFCLASS systems from the definition of the generic fuzzy perceptron, certain restrictions must be specified. The restrictions for a neuro–fuzzy model for function approximation are similar to the NEFCON model.

**Definition 2** *A NEFPROX system is a special 3–layer fuzzy perceptron with the following specifications:*

(i) *The input units are denoted as* $x_1, \ldots, x_n$*, the hidden rule units are denoted as* $R_1, \ldots, R_k$*, and the output units are denoted as* $y_1, \ldots, y_m$.

(ii) *Each connection between units* $x_i$ *and* $R_r$ *is labeled with a linguistic term* $A_{k_r}^{(i)}$.

(iii) *Each connection between units* $R_r$ *and* $y_j$ *is labeled with a linguistic term* $B_{k_r}^{(j)}$.

(iv) *Connections coming from the same input unit* $x_i$ *and having identical labels, bear the same fuzzy weight at all times. These connections are called* **linked connections**, *and their weight is called a* **shared weight**. *An analogous condition holds for the connections leading to the same output unit* $y_j$.

(v) *Let* $L_{x,R}$ *denote the label of the connection between an input unit* $x$ *and a rule unit* $R$. *For all rule units* $R, R'$ $(\forall x\ L_{x,R} = L_{x,R'}) \Longrightarrow R = R'$ *holds.*

This definition makes it possible to interpret a NEFRPOX system as a plain fuzzy system. Each hidden unit represents a fuzzy if–then rule. Condition (iv) specifies that there have to be *shared* or *linked weights*. If this feature is missing, it would be possible for fuzzy weights representing identical linguistic terms to evolve differently during the learning process. If this is allowed to happen, each rule can have its individual membership functions for its antecedent and conclusions variables. This would inhibit proper interpretation of the rule base, and is highly undesirable. Condition (v) determines that there are no rules with identical antecedents.

These conditions are similar to the definitions for NEFCON or NEFCLASS systems. But note that NEFCON systems have only a single output node, and NEFCLASS systems do not use membership functions on the conclusion side.

# 3 Structure and Parameter Learning

In a function approximation problem we can use plain supervised learning, because we know for each given input vector the correct output vector (fixed learning problem). If we use a system of fuzzy rules to approximate the function, we can use prior knowledge. This means if we already know suitable rules for certain areas, we can initialize the neuro–fuzzy system with them. The remaining rules have to be found by learning. If there is no prior knowledge we start with a NEFPROX system without hidden units, and incrementally learn all rules.

The learning algorithm for NEFPROX is described in the following definition. We assume that triangular membership functions are used that are described by three parameters:

$$\mu : \mathbb{R} \to [0,1], \quad \mu(x) = \begin{cases} \frac{x-a}{b-a} & \text{if } x \in [a,b), \\ \frac{c-x}{c-b} & \text{if } x \in [b,c], \\ 0 & \text{otherwise.} \end{cases}$$

In addition, the leftmost and rightmost membership functions for each variable can be shouldered. We use triangular fuzzy sets for reasons of simplicity. However, the learning algorithm can be applied to other forms of membership functions as well. As defuzzyification procedure in the output nodes we can use for example COG or MOM.

To start the learning process, we must specifiy initial fuzzy partitions for each input variable. This is not necessary for output variables, for them fuzzy sets can be created during learning. However, if the learning algorithm shall start with specific fuzzy sets, they can be defined. If no fuzzy sets are given, it is necessary to specify the initial width (here: $|a-c|$) of a membership function that is created during learning.

**Definition 3 (NEFPROX learning algorithm)** *Consider a NEFPROX system with $n$ input units $x_1, \ldots, x_n$, $k$ rule units $R_1, \ldots, R_k$, and $m$ output units $y_1, \ldots, y_m$. Also given is a learning set $\tilde{\mathcal{L}} = \{(\mathbf{s}_1, \mathbf{t}_1), \ldots, (\mathbf{s}_r, \mathbf{t}_r)\}$ of $r$ patterns, each consisting of an input pattern $\mathbf{s} \in \mathbb{R}^n$, and a target pattern $\mathbf{t} \in \mathbb{R}^m$ (desired ouput). The learning algorithm that is used to create the $k$ rule units of the NEFPROX system consists of the following steps* **(structure learning algorithm)***:*

*(i) Select the next pattern $(\mathbf{s}, \mathbf{t})$ from $\tilde{\mathcal{L}}$*

*(ii) For each input unit $x_i \in U_1$ find the membership function $\mu_{j_i}^{(i)}$ such that $\mu_{j_i}^{(i)}(s_i) = \max\limits_{j \in \{1,\ldots,p_i\}} \{\mu_j^{(i)}(s_i)\}$*

*(iii) If there is no rule node $R$ with $W(x_1, R) = \mu_{j_1}^{(1)}, \ldots, W(x_n, R) = \mu_{j_n}^{(n)}$, then create such a node, and connect it to all output nodes.*

*(iv) For each connection from the new rule node to the output nodes find a suitable fuzzy weight by the following procedure:*
*From the membership functions assigned to an output unit $y_i$ find a membership function $\nu_{j_i}^{(i)}$ such that $\nu_{j_i}^{(i)}(t_i) = \max\limits_{j \in \{1,\ldots,q_i\}} \{\nu_j^{(i)}(t_i)\}$, and $\nu_{j_i}^{(i)}(t_y) \geq 0.5$. If there is*

*no such fuzzy set, then create a fuzzy set* $\nu_{new}^{(i)}$ *such that* $\nu_{new}^{(i)}(t_i) = 1$ *holds, add it to the fuzzy sets assigned to output variable* $y_i$*, and set* $W(R, y_i) = \nu_{new}^{(i)}$.

*(v) If there are still unprocessed patterns in* $\tilde{\mathcal{L}}$*, then proceed with step (i), otherwise stop the rule creation process.*

*(vi) Finally, evaluate the rule base: Determine the mean output for each output variable of each rule given by those patterns for which the respective rule has a degree of fulfillment greater than* 0*. If there is an output fuzzy set to which the mean output has a higher degree of membership than to the current fuzzy set used by the respective rule, then change the rule conclusion accordingly.*

*The supervised learning algorithm of a NEFPROX system to adapt its fuzzy sets runs cyclically through the learning set* $\tilde{\mathcal{L}}$ *by repeating the following steps until a given end criterion is met* **(parameter learning algorithm)***:*

*(i) Select the next pattern* $(\mathbf{s}, \mathbf{t})$ *from* $\tilde{\mathcal{L}}$*, propagate it through the NEFPROX system, and determine the output vector.*

*(ii) For each output unit* $y_i$*: Determine the delta value* $\delta_{y_i} = t_i - o_{y_i}$

*(iii) For each rule unit* $R$ *with* $o_R > 0$*:*

*(a) For all* $y_i \in U_3$ *determine the delta values for the parameters* $a, b, c$ *of the fuzzy set* $W(R, y_i)$ *using the learning rate* $\sigma > 0$*:*

$$\begin{aligned} \delta_{b_i} &= \sigma \cdot \delta_{y_i} \cdot (c - a) \cdot o_R \cdot (1 - W(R, y_i)(t_i)), \\ \delta_{a_i} &= \sigma \cdot (c - a) \cdot o_R + \delta_b, \\ \delta_{c_i} &= -\sigma \cdot (c - a) \cdot o_R + \delta_b, \end{aligned}$$

*and apply the changes to* $W(R, y_i)$ *if this does not violate against a given set of constraints* $\Phi$*. (Note: the weight* $W(R, y_i)$ *might be shared by other connections, and in this case it might be changed more than once)*

*(b) Determine the delta value* $\delta_R = o_R(1 - o_R) \cdot \sum_{y \in U_3} (2W(R, y)(t_i) - 1) \cdot |\delta_y|$.

*(c) For all fuzzy sets* $W(x, R)$ *with* $W(x, R)(o_x) > 0$ *determine the delta values for its parameters* $a, b, c$ *using the learning rate* $\sigma > 0$*:*

$$\begin{aligned} \delta_b &= \sigma \cdot \delta_R \cdot (c - a) \cdot (1 - W(x, R)(o_x))) \cdot \operatorname{sgn}(o_x - b), \\ \delta_a &= -\sigma \cdot \delta_R \cdot (c - a) \cdot (1 - W(x, R)(o_x))) + \delta_b, \\ \delta_c &= \sigma \cdot \delta_R \cdot (c - a) \cdot (1 - W(x, R)(o_x))) + \delta_b. \end{aligned}$$

*and apply the changes to* $W(x, R)$ *if this does not violate against a given set of constraints* $\Phi$*. (Note: the weight* $W(x, R)$ *might be shared by other connections, and in this case it might be changed more than once)*

*(iv) If an epoch was completed, and the end criterion is met, then stop; otherwise proceed with step (i).*

The structure learning algorithm selects fuzzy rules based on a predefined grid over the input space. This grid is given by the initial fuzzy partitions. If the algorithm creates too many rules, it is possible to evaluate them by determining individual rule errors, and keeping only the best rules.

In this case, however, the approximation performance will suffer. Each rule represents a number of samples of the (unknown) function in form of a fuzzy sample. If rules are deleted, this means that some samples are not considered anymore. If parameter learning cannot compensate for this, then the approximation performance must decrease.

The learning procedure for the fuzzy sets is a simple heuristic. It results in shifting the membership functions, and in making their supports larger or smaller. It is easy to define constraints $\Phi$ for the learning procedure, e.g. that fuzzy sets must not pass each other, or that they must intersect at 0.5, etc. As a stop criterion usually the error over an additional validation set is observed. Training is continued until the error over the validation set does not further decrease. This technique is well known from neural network learning, and is used to avoid over–fitting to the training data.

# 4 Results and Conclusions

As an example for the learning capabilities of the NEFPROX algorithms, we consider a chaotic time series given by the Mackey–Glass differential equation:

$$\dot{x}(t) = \frac{0.2x(t-\tau)}{1+x^{10}(t-\tau)} - 0.1x(t).$$

We use the values $x(t-18), x(t-12), x(t-6)$ and $x(t)$ to predict $x(t+6)$. The training data was created using a Runge–Kutta procedure with step width 0.1. As initial conditions for the time series we used $x(0) = 1.2$ und $\tau = 17$. We created 1000 values between $t = 118$ and 1117, where the first 500 samples were used as training data, and the second half was used as a validation set.

The NEFPROX system that was used to approximate the time series has four input and one output variable. Each variable was initially partitioned by 7 equally distributed triangular fuzzy sets, where the leftmost and rightmost membership functions were shouldered. Neighboring membership functions intersected at degree 0.5. The range of the output variable was extended for 10% in both directions, to better obtain extreme output values. We used max–min inference and mean–of–maximum defuzzification, i.e. the NEFPROX system represents a common Mamadani–type of fuzzy system with MOM defuzzification. We used MOM defuzzification because it is more than two times faster compared to center of gravity defuzzification, and produces almost the same results after learning.

This NEFPROX system has $105 = (4+1) * 7 * 3$ adjustable parameters. The learning procedure was carried out in batch mode, and parameter learning was constraint by not allowing a membership function to pass one of its neighbors. We also used an adaptive learning rate. Beginnig with $\sigma = 0.01$ the learning rate is multiplied by 1.1, if the error on the validation set decreases for 4 consecutive steps. If the error oscillates or increases,

the learning rate is multiplied by 0.9. Learning is stopped, if the error on the validation set cannot be further reduced for 100 epochs. The NEFPROX system with the lowest error is saved during the learning process, and it is restored after learning.

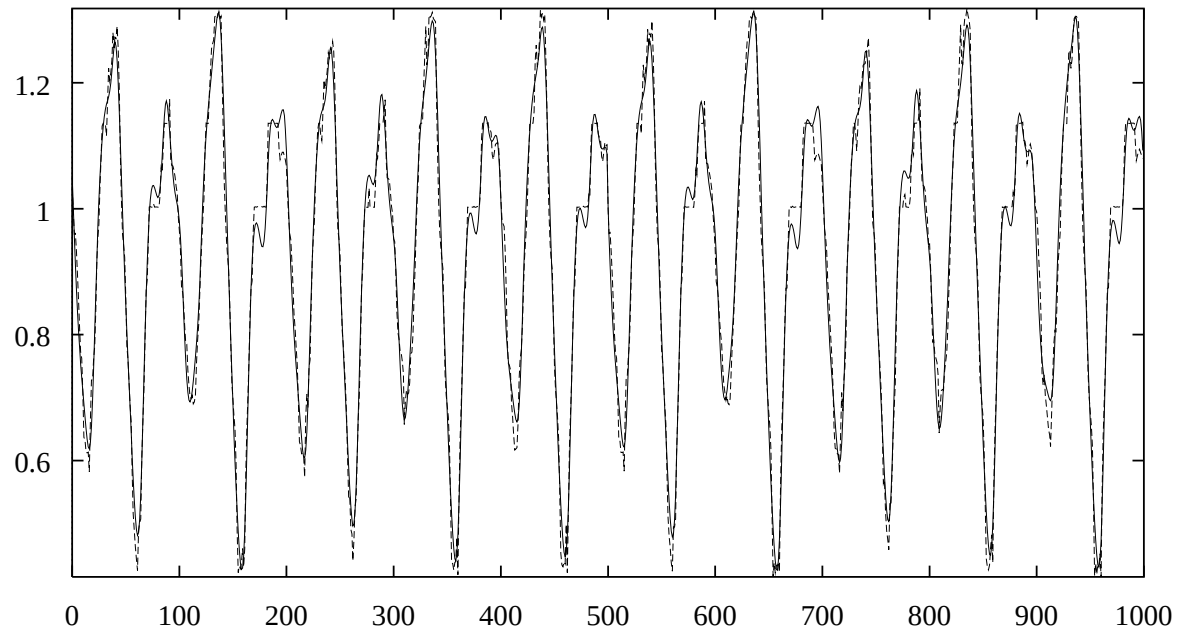


Figure 1: Approximation of the Mackey–Glass time series by NEFPROX

In Figure 1 the approximation performance of NEFPROX after 216 epochs can be seen (solid line = original data). The values 1 – 500 are the training data, and the values 501– 1000 are from the validation set. The structure learning procedure created 129 fuzzy rules (in this configuration there could be a maximum of $7^4 = 2401$ different rules out of possible $7^5 = 16807$ rules). The number of rules does not influence the number of free parameters, but only the run time of the simulation. The root mean square error (RMSE) on the training set results in 0.0315, and in 0.0332 for the validation set. On a SUN UltraSparc the training takes 75 seconds.

Compared to an ANFIS model with two bell–shaped fuzzy sets per input variable and 16 rules (i.e. $4 * 2 * 3 + 16 * 5 = 104$ free parameters) a better aproximation can be obtained (RMSE of 0.0016 and 0.0015)[3]. However, the trainig time is about 15 times longer using the same hardware platform (18 minutes on a SUN UltraSparc using software distributed by Jang at ftp.cs.cmu.edu in user/ai/areas/fuzzy/systems/anfis). An ANFIS model represents a Sugeno–type of fuzzy system using sum–prod inference. Because the conclusions of ANFIS rules consist of linear combinations of the input variables, the number of free parameters in an ANFIS systems depends also on the number of rules.

# 5 Conclusions

We have presented learning algorithms to find the structure and the parameters of a fuzzy system to approximate a function given by a supervised learning problem. The resulting NEFPROX model is an extension to the NEFCON and NEFCLASS approaches. The first

results on the learning capabilities of the NEFPROX model are quite promising. NEFPROX can learn a common Mamdani–type of fuzzy system from data. It uses a restricted learning algorithm, such that the semantics and interpretability of the represented fuzzy system are retained.

Compared to ANFIS, NEFPROX is much faster, but ANFIS yields better approximation results. There is a trade–off between fast learning using simple algorithms, and exact approximations using complex learning algorithms. In addition, NEFPROX offers a method for structure learning, where the ANFIS model does not. The rule base for ANFIS has to be given in advance, and it can only represent Sugeno–type fuzzy models, which are not as easy to interpret as Mamdani–type models.

To increase the interpretability, the number of rules which are created by the learning procedure must be reduced. Currently a large number of rules, and many fuzzy sets are necessary to obtain acceptable approximation results. Algorithms for rule reduction are currently under examination. The NEFPROX software can be obtained by WWW from fuzzy.cs.uni-magdeburg.de.

# References


[1] J. J. Buckley. Sugeno type controllers are universal controllers. *Fuzzy Sets and Systems*, 53:299–303, 1993.

[2] Simon Haykin. *Neural Networks. A Comprehensive Foundation*. Macmillan College Publishing Company, New York, 1994.

[3] J.-S. Roger Jang. ANFIS: Adaptive-network-based fuzzy inference systems. *IEEE Trans. Systems, Man & Cybernetics*, 23:665–685, 1993.

[4] Bart Kosko. Fuzzy systems as universal approximators. In *Proc. IEEE Int. Conf. on Fuzzy Systems 1992*, pages 1153–1162, San Diego, March 1992.

[5] Detlef Nauck. Building neural fuzzy controllers with NEFCON–I. In Rudolf Kruse, Jörg Gebhardt, and Rainer Palm, editors, *Fuzzy Systems in Computer Science*, pages 141–151. Vieweg, Braunschweig, 1994.

[6] Detlef Nauck and Rudolf Kruse. NEFCLASS – a neuro–fuzzy approach for the classification of data. In K.M. George, Janice H. Carrol, Ed Deaton, Dave Oppenheim, and Jim Hightower, editors, *Applied Computing 1995. Proc. of the 1995 ACM Symposium on Applied Computing, Nashville, Feb. 26–28*, pages 461–465. ACM Press, New York, February 1995.

[7] Detlef Nauck and Rudolf Kruse. Designing neuro–fuzzy systems through backpropagation. In Witold Pedrycz, editor, *Fuzzy Modelling: Paradigms and Practice*, pages 203–228. Kluwer, Boston, 1996.

[8] Detlef Nauck, Ulrike Nauck, and Rudolf Kruse. Generating classification rules with the neuro–fuzzy system NEFCLASS. In *Proc. Biennial Conference of the North American Fuzzy Information Processing Society NAFIPS'96*, pages 466–470, Berkeley, June 1996. IEEE.