A Neuro-Fuzzy Approach to Obtain Interpretable Fuzzy Systems for Function Approximation

Detlef Nauck and Rudolf Kruse
University of Magdeburg, Faculty of Computer Science,
Universitaetsplatz 2, D-39106 Magdeburg, Germany
Tel: +49.391.67.12700, Fax: +49.391.67.12018
E-Mail: Detlef.Nauck@cs.uni-magdeburg.de, WWW: fuzzy.cs.uni-magdeburg.de/~nauck

Abstract

Fuzzy systems can be used for function approximation based on a set of linguistic rules. We present a method to obtain the necessary parameters for such a fuzzy system by a neuro-fuzzy training method. The learning algorithm is able to determine the structure and the parameters of a fuzzy system from sample data. The approach is an extension to our already published NEFCON and NEFCLASS models which are used for control or classification purposes. The NEFPROX model, which is discussed in this paper is more general, and it can be used for any problem based on function approximation. We especially consider the problem to obtain interpretable fuzzy systems by learning.

1. Introduction

Certain fuzzy systems are universal function approximators [3], [5]. 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. In this paper we only want to consider neuro-fuzzy systems which display the following properties [7]:
1. A neuro-fuzzy system is based on a fuzzy system which is trained by a learning algorithm derived from neural network theory. The (heuristical) learning procedure operates on local information, and causes only local modifications in the underlying fuzzy system.
2. A neuro-fuzzy system can be viewed as a 3-layer feedforward neural network. The first layer represents input variables, the middle (hidden) layer represents fuzzy rules and the third layer represents output variables. Fuzzy sets are encoded as (fuzzy) connection weights. (*Remark:* Sometimes a 5-layer architecture is used, where the fuzzy sets are represented in the units of the second and fourth layer).
3. A neuro-fuzzy system can be always (i.e. before, during and after learning) interpreted as a system of fuzzy rules. It is also possible to create the system out of training data from scratch, as it is possible to initialize it by prior knowledge in form of fuzzy rules. (*Remark:* Not all neuro-fuzzy models support fuzzy rule creation).
4. The learning procedure of a neuro-fuzzy system takes the semantical properties of the underlying fuzzy system into account. This results in constraints on the possible modifications applicable to the system parameters. (*Remark:* Not all neuro-fuzzy approaches have this property).
5. A neuro-fuzzy system approximates an n-dimensional (unknown) function that is partially defined by the training data. The fuzzy rules encoded within the system represent vague samples, and can be viewed as prototypes of the training data. A neuro-fuzzy system should not be seen as a kind of (fuzzy) expert system, and it has nothing to do with fuzzy logic in the narrow sense [6].

We have already presented two neuro-fuzzy approaches NEFCON [8], [11] and NEFCLASS [9]. 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 discuss 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. The approach is called NEFPROX (NEuro Fuzzy function apPROXimator). After introducing the architecture of the NEFPROX model we discuss some problems of obtaining interpretable fuzzy systems by learning. Then we present the learning algorithms of NEFPROX and illustrate their capabilities on a small example.

2. The Architecture of NEFPROX

Function approximation based on local learning strategies is a domain of neural networks 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 [4]. 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:

1. $U = \bigcup_{i \in M} U_i$ 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.
2. 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 $\mathbb{R}$).
3. A defines an activation function A_u for each $u \in U$ to calculate the activation a_u
   (a) for input and rule units $u \in U_1 \cup U_2$:
   $A_u : \mathbb{R} \to \mathbb{R}, \quad a_u = A_u(\mathrm{net}_u) = \mathrm{net}_u$,
   (b) for output units $u \in U_3$:
   $A_u : \mathcal{F}(\mathbb{R}) \to \mathcal{F}(\mathbb{R}),\ a_u = A_u(\mathrm{net}_u) = \mathrm{net}_u$.
4. O defines for each $u \in U$ an output function O_u to calculate the output o_u
   (a) for input and rule units $u \in U_1 \cup U_2$:
   $O_u : \mathbb{R} \to \mathbb{R}, \quad o_u = O_u(a_u) = a_u$,
   (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 DEFUZZ_u is a suitable defuzzification function.
5. NET defines for each unit $u \in U$ a propagation function NET_u to calculate the net input net_u
   (a) for input units $u \in U_1$:
   $\mathrm{NET}_u : \mathbb{R} \to \mathbb{R}, \quad \mathrm{net}_u = ex_u$,
   (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.
6. $\mathrm{ex} : U_1 \to \mathbb{R}$, 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 [7]. 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 (see Fig. 1) with the following specifications:

1. 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 $y_1, \ldots, y_m$.
2. Each connection between units x_i and R_r is labeled with a linguistic term $A^{(i)}_{k_r}$.
3. Each connection between units R_r and y_j is labeled with a linguistic term $B^{(j)}_{k_r}$.
4. 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**.

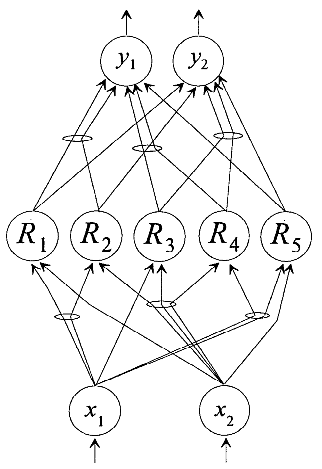

Fig. 1. Architecture of a NEFPROX system

An analogous condition holds for the connections leading to the same output unit y_j.
5. 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. Learning Interpretable Fuzzy Systems

Compared to approaches like ANFIS our NEFPROX model has the advantage that it can also represent Mamdani-type fuzzy systems, which are more easy to interpret than Sugeno-type systems. Our interest with NEFPROX is to obtain simple fuzzy systems that can be interpreted well. For this it is important to have a small number of rule an a small number of meaningful membership functions.

However there is a trade-off between readability and precision. If we are interested in a very precise function approximation, then we are not so much interested in the interpretability of the solution. In this case we want to use another feature of fuzzy system: The convenient combination of local models to an overall solution. For this Sugeno-type models are more suited than Mamdani-type models. However, if we are interested in a precise solution we should consider whether a fuzzy system is the most suitable approach. Very similar to Sugeno-type fuzzy systems are radial basis function networks, kernel regression, B-spline networks etc. [1], [2].

If we are more interested in an interpretable solution, then a Mamdani-type fuzzy system should be preferred. However, we must consider that we will probably not obtain a very precise solution by learning, because we cannot allow a learning algorithm to apply any possible modification to the parameters of a fuzzy systems that may be possible. For the sake of interpretability we must constrain the learning procedure. This idea can be found in all our neuro-fuzzy approaches [7].

A neuro-fuzzy learning procedure should in this case be very simple and fast to allow a user to understand what it does and to experiment with it. We prefer a tool-oriented view on neuro-fuzzy systems. We consider a neuro–fuzzy method to be a tool for creating fuzzy systems from data. The learning algorithm should take the semantics of the desired fuzzy system into account, and adhere to certain constraints. The learning result should also be interpreted, and the insights gained by this should be used to restart the learning procedure to obtain better results if necessary. A neuro-fuzzy system supports the user to find a desired fuzzy system based on training data, but it cannot do all the work

Semantical problems will occur if neuro-fuzzy systems do not have mechanisms to make sure that all changes caused by the learning procedure are interpretable in terms of a fuzzy system. The learning algorithms should be constrained such that adjacent membership functions do not exchange positions, do not move from positive to negative parts of the domains or vice versa, have a certain degree of overlapping, etc. An interpretation in terms of a Mamdani-type fuzzy system is also in danger if the evaluation of antecedents is not done by t-norms, but by some special functions. This is sometimes done to allow gradient descent learning to be applied. We refrain from this in NEFPROX and use simple heuristics for learning instead.

For applying a neuro-fuzzy learning strategy one additional aspect should be considered: for whatever reason we choose a fuzzy system to solve a problem it cannot be because we need an *exact* solution. Fuzzy systems are used to exploit the tolerance for imprecise solutions. So it does not make much sense to select a very sophisticated and expensive training procedure to squeeze the last bit of information from the training data or error measure. To do this we usually must leave the standard fuzzy system architectures behind and get semantical problems in exchange. From our point of view fuzzy systems are used because they are easy to implement, easy to handle and easy to understand. A learning algorithm to create a fuzzy system from data also should have these features.

Our view of neuro-fuzzy models as a way heuristically to find parameters of fuzzy systems by processing training data with a learning algorithm, is expressed by the list of five points given above. We think that neuro-fuzzy systems should be seen as development tools that can help to construct a fuzzy system. They are not automatic "fuzzy system creators". The user should always supervise and interpret the learning process. We have also to keep in mind that, as in neural networks, a successful learning outcome cannot be guaranteed for the learning process of a neuro-fuzzy system. The same guidelines for selection and preprocessing of training data that are known from neural networks apply to neuro-fuzzy systems.

4. Structure and Parameter Learning

In a function approximation problem we can use plain supervised learning, because the correct output is known for the training data. 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 given in Def. 3. We assume that triangular membership functions are used that are described by three parameters:

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

The leftmost and rightmost membership functions for each variable may be shouldered. We use triangular fuzzy sets for the sake of simplicity, but the learning algorithm can be applied to other forms of membership functions as well. We can use, for example, either the center-of-gravity or the mean-of-maximum method as the defuzzification procedure in the output nodes.

To start the learning process, we must specify initial fuzzy partitions for each input variable. This is not necessary for output variables, for which fuzzy sets can be created during learning. However, if the learning algorithm is to 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.

For the following definition remember that $W(.,.)$ denotes a fuzzy weight (membership function) and that o represents the output of a NEFPROX unit.

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 problem $\tilde{\mathcal{L}} = \{(\mathbf{s}_1, \mathbf{t}_1), \ldots, (\mathbf{s}_r, \mathbf{t}_r)\}$ of r patterns, each consisting of an input pattern $\mathbf{s} \in \mathbb{R}^n$, and a target pattern $\mathbf{t} \in \mathbb{R}^m$. The learning algorithm that is used to create the k rule units of the NEFPROX system consists of the following steps (*rule learning algorithm*):

1. Select the next pattern $(\mathbf{s}, \mathbf{t})$ from $\tilde{\mathcal{L}}$.
2. For each input unit $x_i \in U_1$ find the membership function $\mu_{j_i}^{(i)}$ such that

$$\mu_{j_i}^{(i)}(s_i) = \max_{j \in \{1,\ldots,p_i\}} \{\mu_j^{(i)}(s_i)\}.$$

3. 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.
4. 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

$$\nu_{j_i}^{(i)}(t_i) = \max_{j \in \{1,\ldots,q_i\}} \{\nu_j^{(i)}(t_i)\}, \ \textit{and}\ \nu_{j_i}^{(i)}(t_y) \geq 0.5.$$

If there is no such fuzzy set, then create $\nu_{\text{new}}^{(i)}$ such that $\nu_{\text{new}}^{(i)}(t_i) = 1$, add it to the fuzzy sets assigned to output variable y_i, and set $W(R, y_i) = \nu_{\text{new}}^{(i)}$.
5. If there are still unprocessed patterns in $\tilde{\mathcal{L}}$, then go to with step (i), otherwise stop creating rules.
6. 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 fulfilment 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 consequent accordingly.

The supervised learning algorithm for the fuzzy sets of a NEFPROX system runs cyclically through the learning set $\tilde{\mathcal{L}}$ repeating the following steps until some stop criterion is met (*fuzzy set learning algorithm*):

1. Select the next pattern $(\mathbf{s}, \mathbf{t})$ from $\tilde{\mathcal{L}}$, propagate it through the NEFPROX system, and determine the output vector.
2. For each output unit y_i, determine the difference between desired and actual output value $\delta_{y_i} = t_i - o_{y_i}$.
3. For each rule unit R with $o_R > 0$:

(a) For all $y_i \in U_3$ determine the modifications for the parameters a, b, c of the fuzzy set $W(R, y_i)$ using the learning rate $\sigma > 0$.
If $W(R, y_i)(t_i) > 0$

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

If $W(R, y_i)(t_i) = 0$

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

Apply the changes to $W(R, y_i)$ if this does not violate a given set of constraints Φ. (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 rule error

$$E_R = o_R(1 - o_R) \cdot \sum_{y \in U_3} (2W(R, y)(t_i) - 1) \cdot |\delta_y|.$$

(c) For each fuzzy set $W(x, R)$ with $W(x, R)(o_x) > 0$ determine the modifications for its parameters a, b, c using the learning rate $\sigma > 0$:

$$\begin{aligned}\Delta_b &= \sigma \cdot E_R \cdot (c-a) \cdot (1 - W(x, R)(o_x))\\ &\quad \cdot \operatorname{sgn}(o_x - b),\\ \Delta_a &= -\sigma \cdot E_R \cdot (c-a) \cdot (1 - W(x, R)(o_x)) + \Delta_b,\\ \Delta_c &= \sigma \cdot E_R \cdot (c-a) \cdot (1 - W(x, R)(o_x)) + \Delta_b,\end{aligned}$$

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

4. If an epoch has been completed, and the stop criterion is met, then stop; otherwise go to step (i).

As it is the case in our NEFCLASS model [10], the rule learning algorithm selects fuzzy rules based on a predefined partitioning of the input space. This partitioning is given by the initial fuzzy sets. If the algorithm creates too many rules, it is possible to evaluate them by determining individual rule errors, keeping only the best rules.

In this case, however, the approximation performance can suffer. Each rule represents a number of samples of the (unknown) function in the form of a fuzzy sample. If rules are deleted, this means that some samples are no longer considered. If parameter learning cannot compensate for this, then the approximation performance must decrease. For classification problems as they are handled by NEFCLASS [10], rule pruning is not such a problem. This is due to the winner-takes-all interpretation which is not much influenced by small changes in the output units. In contrast, the output of NEFPROX is taken as a function result such that changes in the output units have a stronger influence.

As in our other neuro-fuzzy models NEFCON and NEFCLASS [7], 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. As before, the learning procedure must meet certain constraints Φ. As a stop criterion the error over an additional validation set can be chosen. Training is continued until the error is no longer decreasing. This technique is well known from neural network learning, and is used to avoid over-fitting to the training data.

5. An Example: Time Series Prediction

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 $x(0) = 1.2$ and $\tau = 17$. 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

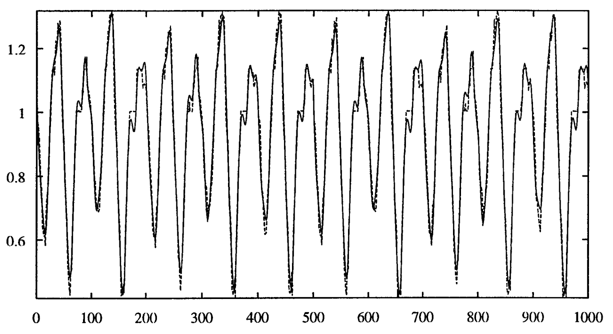

Fig. 2. Approximation of the Mackey-Glass time series by NEFPROX

used MOM defuzzification because it is more than two times than center of gravity defuzzification, and produces almost the same results after learning.

This NEFPROX system has $105 = (4+1) \cdot 7 \cdot 3$ 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. Beginning 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.

Fig. 2 shows the approximation performance of NEFPROX after 216 epochs (solid line = original data). Values 1 - 500 are the training data, and 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 errors (RMSE) are 0.0315 on the training and 0.0332 on the validation set. On a SUN UltraSparc training takes 75 seconds.

Compared to an ANFIS model with two bell-shaped fuzzy sets per input variable and 16 rules (i.e. $4 \cdot 2 \cdot 3 + 16 \cdot 5 = 104$ free parameters) a better aproximation can be obtained (RMSE of 0.0016 and 0.0015)[4]. However, the trainig time is about 15 times longer (18 minutes on a SUN UltraSparc using software distributed by Jang at ftp.cs.cmu.edu in user/ai/areas/fuzzy/systems/anfis). 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.

6. 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. 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. NEFPROX can do structure learning, but for ANFIS Sugeno-type rule must be given whoose consequents cannot be interpreted linguistically.

NEFPROX is a first step towards learning interpretable fuzzy systems for function approximation. We are currently working on using some of our results from NEFCLASS [9], [10] to improve NEFPROX. To increase interpretability, the number of rules created during learning must be reduced. Often a large number of rules and fuzzy sets are needed to obtain acceptable approximation results. Algorithms for rule reduction are under examination. The NEFPROX software is available at http://fuzzy.cs.uni-magdeburg.de.

References

[1] Hugues Bersini and Gianluca Bontempi. Fuzzy models viewed as multi-expert networks. In *Proc. Seventh International Fuzzy Systems Association World Congress IFSA'97*, volume II, pages 354–359, Prague, 1997.

[2] Martin Brown and Chris Harris. *Neurofuzzy Adaptive Modelling and Control.* Prentice Hall, New York, 1994.

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

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

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

[6] Rudolf Kruse, Jörg Gebhardt, and Frank Klawonn. *Foundations of Fuzzy Systems.* Wiley, Chichester, 1994.

[7] Detlef Nauck, Frank Klawonn, and Rudolf Kruse. *Foundations of Neuro-Fuzzy Systems.* Wiley, Chichester, 1997.

[8] Detlef Nauck and Rudolf Kruse. NEFCON-I: An X-Window based simulator for neural fuzzy controllers. In *Proc. IEEE Int. Conf. Neural Networks 1994 at IEEE WCCI'94*, pages 1638–1643, Orlando, FL, June 1994.

[9] Detlef Nauck and Rudolf Kruse. A neuro-fuzzy method to learn fuzzy classification rules from data. *Fuzzy Sets and Systems*, 89:277–288, 1997.

[10] Detlef Nauck and Rudolf Kruse. New learning strategies for NEFCLASS. In *Proc. Seventh International Fuzzy Systems Association World Congress IFSA'97*, volume IV, pages 50–55, Prague, 1997.

[11] Andreas Nürnberger, Detlef Nauck, and Rudolf Kruse. Neuro-fuzzy control based on the NEFCON model under MATLAB/SIMULINK. In Pravir Chawdry, Rajkumar Roy, and R.K. Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, London, 1997. Springer-Verlag.