NEURO-FUZZY SYSTEMS: REVIEW AND PROSPECTS

Detlef Nauck
Faculty of Computer Science (FIN-IIK), University of Magdeburg
Universitaetsplatz, D-39106 Magdeburg, Germany
Tel. +49.391.67-12700, Fax: +49.391.67-12018
E-Mail: nauck@iik.cs.uni-magdeburg.de, URL:fuzzy.cs.uni-magdeburg.de/∼nauck

Abstract

This paper reviews *neuro-fuzzy systems*, which combine methods from neural network theory with fuzzy systems. Such combinations have been considered for several years already. However, the term *neuro-fuzzy* still lacks proper definition, and still has the flavour of a buzzword to it. Surprisingly few neuro-fuzzy approaches do actually employ neural networks, even though they are very often depicted in form of some kind of neural network structure. However, all approaches display some kind of learning capability, as it is known from neural networks. This means, they use algorithms which enable them to determine their parameters from training data in an iterative process.

In this paper we review some of our neuro-fuzzy approaches to illustrate our view of neuro-fuzzy techniques and our understanding on how these approaches should be used. From our point of view *neuro-fuzzy* means using heuristic learning strategies derived from the domain of neural network theory to support the development of a fuzzy system.

1 Introduction

Ever since fuzzy systems were applied in industrial applications, developers know that the construction of a well performing fuzzy system is not always easy. The problem of finding appropriate membership functions and fuzzy rules is often a tiring process of trial and error. Therefore the idea of applying learning algorithms to fuzzy systems was considered early Approaches to so called *adaptive* or *self–organizing fuzzy controllers* can be found e.g. in [22, 23, 24]. An overview of this area is presented in [5]. These kind of adaptive models usually use knowledge based methods. However, another possibility of learning parameters of fuzzy systems is given by neural networks.

The learning capabilities of neural networks made them a prime target for a combination with fuzzy systems in order to automate or support the process of devloping a fuzzy system for a given task. The first so–called neuro–fuzzy approaches were considered mainly in the domain of (neuro–) fuzzy control, but today the approach is more general. Neuro–fuzzy systems are applied in various domains, e.g. control, data analysis, decision support, etc.

Modern neuro–fuzzy–systems are usually represented as a multilayer feedforward neural network [1, 3, 4, 6, 11, 14, 15, 25], but fuzzifications of other neural network architectures are also considered, for example self–organizing feature maps [2, 26]. In neuro–fuzzy models, connection weights and propagation and activation functions differ from common neural networks. Although there are a lot of different approaches [3, 15], we want to restrict the term "neuro–fuzzy" to systems which display the following properties:

1. A neuro-fuzzy system is a fuzzy system that is trained by a learning algorithm (usually) derived from neural network theory. The (heuristic) learning procedure operates on local information, and causes only local modifications in the underlying fuzzy system. The learning process is not knowledge based, but data driven.
2. A neuro-fuzzy system can be viewed as a special 3-layer feedforward neural network. The units in this network use t-norms or t-conorms instead of the activation functions usually used in neural networks. 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.

This paper appeared in Proc. Fifth European Congress on Intelligent Techniques and Soft Computing (EUFIT'97), Aachen, Sep. 8–11, 1997, pp. 1044–1053.

Some neuro-fuzzy models use more than 3 layers, and encode fuzzy sets as activation functions. In this case, it is usually possible to transform them into a 3-layer architecture.

This view of a fuzzy system illustrates the data flow within the system and its parallel nature. However this neural network view is not a prerequisite for applying a learning procedure, it is merely a convenience.

3. A neuro-fuzzy system can always (i.e. before, during and after learning) be interpreted as a system of fuzzy rules. It is both possible to create the system out of training data from scratch, and it is possible to initialise it by prior knowledge in form of fuzzy rules.
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 of the system's parameters.
5. A neuro-fuzzy system approximates an n-dimensional (unknown) function that is partially given by the training data. The fuzzy rules encoded within the system represent vague samples, and can be viewed as vague 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 [9].

In this paper *neuro-fuzzy* has to be understood as stated by the five points above. Therefore we consider *neuro-fuzzy* as a technique to derive a fuzzy system from data, or to enhance it by learning from examples. The exact implementation of the *neuro-fuzzy model* does not matter. It is possible to use a neural network to learn certain parameters of a fuzzy system, like using a self-organising feature map to find fuzzy rules [21] (cooperative models), or to view a fuzzy system as a special neural network and to apply a learning algorithm directly [14] (hybrid models).

Approaches, where neural networks are used to provide inputs for a fuzzy system, or to change the output of a fuzzy system, we prefer to call *neural (network)/fuzzy (system) combinations* or *concurrent neural/fuzzy models* to stress the difference that in these approaches parameters of a fuzzy system are not changed by a learning process. If the creation of a neural network is the main target, it is possible to apply fuzzy techniques to speed up the learning process, or to fuzzify a neural network by the extension principle to be able to process fuzzy inputs. These approaches could be called *fuzzy neural networks* to stress that fuzzy techniques are used to create or enhance neural networks.

2 A Fuzzy System in a Neural Network Structure

A lot of neuro-fuzzy approaches represent their models as a neural network. This is of course not necessary to apply a learning algorithm to a fuzzy system. However, it can be convenient because it visualises the data flow through the system, as well for the input data, as for the error signals that are used to update the system parameters. An additional benefit is that different models can easily be compared, and structural differences are clearly visible.

There can be also some applicational advantages. If a fuzzy system is represented in form of a network, and a neural network development tool is available that is flexible enough to let us define special activation and propagation function, then it may be possible to use it. Fuzzy system development tools usually implement only very restrictive learning capabilities like, for instance, rule weights.

To describe the network structure of a neuro-fuzzy model in general, it is useful to have a generic model. In [14] we have presented a *generic 3-layer fuzzy perceptron*. The name refers to the structure of the model, that is similar to the perceptrons as they are known from the domain of neural networks. The term *fuzzy (multi-layer) perceptron* has also been used by other authors for their approaches [8, 10, 20]. We use our interpretation of this notion here to describe the structure of our generic model. Other definitions of the term "fuzzy perceptron" are also possible, of course.

By using a generic fuzzy perceptron to derive neuro-fuzzy systems for special domains, it would be possible to evaluate these different neuro-fuzzy approaches by means of the same underlying model. The fuzzy perceptron was used to derive some of the models which we discuss in the following two sections. A generic fuzzy perceptron has the architecture of a usual multilayer perceptron, but the weights are modeled as fuzzy sets and the activation, output, and propagation functions are changed accordingly, to implement a common fuzzy inference path. The intention of this model is to provide a framework for learning algorithms, to be interpretable as a system of linguistic rules, and to be able to use prior rule based knowledge, so that the learning need not start from scratch.

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

1. $U = U_1 \cup U_2 \cup U_3$ *is a non-empty set of units (neurons). For all* $i, j \in \{1,2,3\}, U_i \neq \emptyset$, *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. *By* A *an activation function* A_u *for each* $u \in U$ *is given 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* $u \in U_3$*:*

$$\begin{aligned} A_u &: \mathcal{F}(\mathbb{R}) \to \mathcal{F}(\mathbb{R}), \\ a_u &= A_u(\mathrm{net}_u) = \mathrm{net}_u. \end{aligned}$$

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* $u \in U_3$*:*

$$\begin{aligned} O_u &: \mathcal{F}(\mathbb{R}) \to \mathbb{R}, \\ o_u &= O_u(a_u) = \mathrm{DEFUZZ}_u(a_u), \end{aligned}$$

   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* $u \in U_2$*:*

$$\begin{aligned} \mathrm{NET}_u &: (\mathbb{R} \times \mathcal{F}(\mathbb{R}))^{U_1} \to [0, 1], \\ \mathrm{net}_u &= \mathop{\top}_{u' \in U_1} \{W(u', u)(o_{u'})\}, \end{aligned}$$

   where $\top$ *is a t-norm,*

   (c) *for output units* $u \in U_3$*:*

$$\begin{aligned} \mathrm{NET}_u &: ([0, 1] \times \mathcal{F}(\mathbb{R}))^{U_2} \to \mathcal{F}(\mathbb{R}), \\ \mathrm{net}_u &: \mathbb{R} \to [0, 1], \end{aligned}$$

$$\mathrm{net}_u(x) = \mathop{\perp}_{u' \in U_2} \{\top(o_{u'}, W(u', u)(x))\},$$

   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 fuzzy perceptron can be viewed as a fuzzy system represented as a graph or as a usual 3-layer perceptron that is *fuzzified to a certain extent*. Only the weights, the net inputs, and the activations of the output units are modeled as fuzzy sets. A fuzzy perceptron is like a usual perceptron used for function approximation. The advantage is the interpretation of its structure in the form of linguistic rules, because the fuzzy weights can be associated with linguistic terms. The network can also be created partly, or in the whole, out of linguistic (fuzzy if-then) rules.

In the following two sections we consider neuro-fuzzy models in the domain of function approximation. This means we want to describe a function by means of a fuzzy rule base. First we consider general function approximation, where a function is given by (noisy) data samples. In this case, we can use plain supervised learning. Then we discuss two special cases of function approximation by learning: neuro-fuzzy control and neuro-fuzzy classification.

We encounter a special case of function approximation in neuro-fuzzy control. Here, learning is usually done indirectly by reinforcement, because the outputs (control actions) for given input values (system states) are unknown, if no other controller exists for the considered problem. The sought-after (control) function is not given by data samples in this case, but by the behaviour of the process to be controlled and by an external performance or reinforcement signal that guides the learning process.

In neuro-fuzzy classification plain supervised learning is used again, because we use labelled training data. In this case, we are not looking for a continuous function like in the other two cases, but for a discrete one by which we can classify input vectors.

3 Neuro-Fuzzy Function Approximation

In this section we consider the problem of approximating an unknown continuous function by a fuzzy system, where the function is partly specified by a set of data samples. This is a supervised learning problem, because the error of the approximation is defined by the difference between the actual output of the fuzzy system, and the target output given in the training data.

The NEFPROX model (neuro fuzzy function approximation) is a neuro-fuzzy approach based on a Mamdani-type of fuzzy system. It can be derived from the generic fuzzy perceptron [17].

A NEFPROX system (see Fig. 1) is a special 3-layer fuzzy perceptron 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 is weighted with a fuzzy set, and is labelled with a linguistic term.

3. Connections that come from the same input unit and have 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 that lead to the same output unit.

4. 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 NEFPROX system in terms of a fuzzy system; each hidden unit represents a fuzzy if-then rule. Condition 3 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, the architecture of the NEFPROX system cannot be understood as a fuzzy rule base. Shared weights make sure that for each linguistic value (e.g. "x_1 is *positive big*") there is only one representation as a fuzzy set, i.e. the linguistic value has only one interpretation for all rule units (e.g. R_1 and R_2 in Fig. 1). It cannot happen that two fuzzy sets that are identical at the beginning of the learning process develop differently, and so the semantics of the rule base encoded in the structure of the network is not affected [11]. Connections that share a weight always come from the same input unit or lead to the same output unit. Condition 4 determines that there are no rules with identical antecedents.

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 fuzzy set learning algorithm for NEFPROX is a simple, computationally inexpensive heuristic procedure, and not a gradient descent method, which would not be applicable, because the functions (min and max) used for evalutating the fuzzy rules are not differentiable. Based on the error measure at the output layer the fuzzy sets of the conclusions are shifted to higher or lower values, and the width of their support is modified. Then the error is propagated back to the rule nodes. Each rule node computes its individual error value and uses it to correct the spread and position of the antecedent membership functions. It is easy to define constraints 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 stopping criterion usually the error on an additional validation set is observed. Training is continued until the error on 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.

To start the learning process, we must specify initial fuzzy partitions for each input variable. This is not necessary for output variables. For them, fuzzy sets can be created during learning, by creating a fuzzy set of a given shape at the current output value, if there is no suitable fuzzy set so far.

The structure (rule) learning algorithm selects fuzzy rules based on a predefined partitioning of the input space (see also Fig. 2) given by the initial fuzzy sets. If the algorithm creates too many rules, it is possible to evaluate them by determining individual rule errors and to keep only the best rules.

In this case, however, the approximation performance will suffer. Each rule represents a number of crisp samples of the (unknown) function by a fuzzy sample. If rules are deleted, some samples are not considered anymore. If parameter learning cannot compensate for this, then the approximation performance must decrease. For classification problems as they are handled by NEFCLASS (see next section) 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.

We often encounter a trade-off in neuro-fuzzy approaches. To obtain a high performance, we need complex training algorithms based on gradient descent, algorithms which demand Sugeno-type fuzzy systems. If we use a Mamdani-type fuzzy system, which is easier to interpret, we can use fast heuristics for training, but usually achieve a lower performance.ANFIS [7], for example, on the one hand sometimes provides better approximation than NEFPROX, which on the other hand learns much faster [17, 16].

Another problem in neuro-fuzzy systems is rule learning. Either no rule learning procedure is defined for a given neuro-fuzzy model (e.g. ANFIS), or simple heuristics are used. However, those simple strategies are not always powerful enough to yield good (i.e. small and interpretable) rule bases. In this case, it can be useful to consider pruning techniques from neural networks to reduce the number of rules and variables in a neuro-fuzzy system [27]. It is also possible to use e.g. fuzzy clustering methods to find fuzzy rules, and initialize a neuro-fuzzy system with them.

Learning fuzzy rule is especially hard in fuzzy control. Neuro-fuzzy control is a special case of function approximation, where we want to approximate a control function (we restrict ourselves to static aspects). We assume that we want to find a fuzzy controller for some process by learning. We further assume that we do not have any previously recorded training data, i.e. we cannot use plain supervised learning. A solution is to use reinforcement learning, if either a model of the considered process is available, or training can be done online using the real process.

NEFCON [12] is a model for neural fuzzy controllers developed by our group, and it is also based on the architecture of the generic fuzzy perceptron described above. A NEFCON system is in fact a NEFPROX system with just one output variable, and which is trained by reinforcement learning. The NEFCON learning algorithm uses a rule based fuzzy error measure as reinforcement signal. Thus it is possible to define a reinforcement type learning algorithm without using an adaptive critic element. The algorithm enables NEFCON to learn fuzzy sets as well as fuzzy rules. Learning a rule base is done by deleting (decremental rule learning) or inserting rules (incremental rule learning). Hence the learning process can work online and does not need previously recorded sample data.

One possibility to learn rules is to start with a NEFCON system that contains all fuzzy rules that can be defined due to the partitioning of the variables. Thus the system begins with an inconsistent initial rule base, which must be made consistent by learning. The idea of decremental rule learning is to try out existing rules and to evaluate them. Rule units that do not pass this test are eliminated from the network. This rule learning algorithm becomes very expensive, if there are a lot of fuzzy sets defined for a lot of variables. For this reason

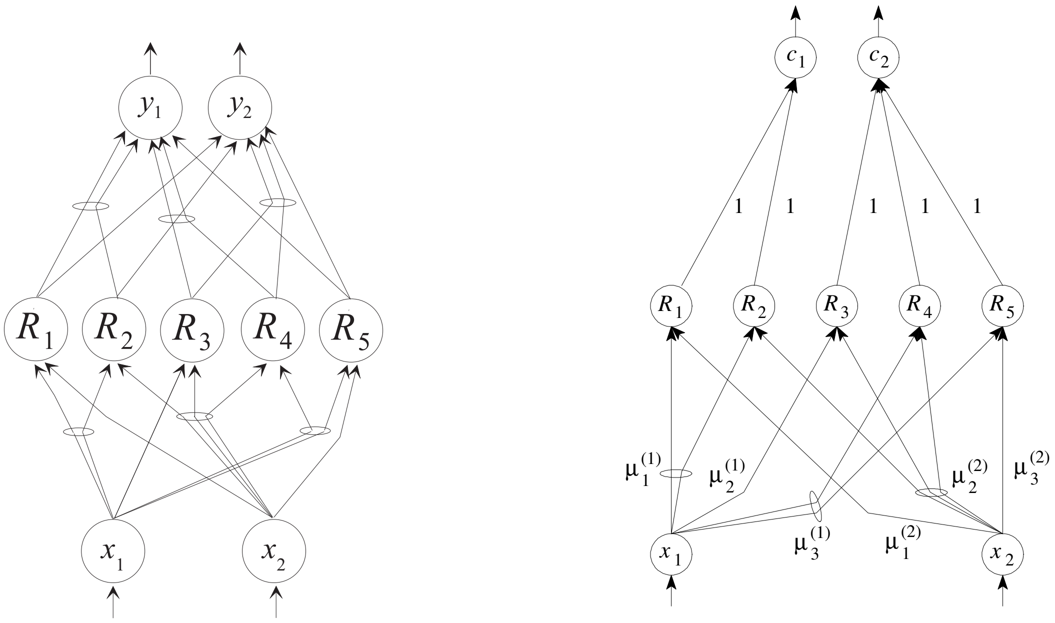

Figure 1: The structure of the NEFPROX model (right): Some of the connections are linked – they always have the same (fuzzy) weight. On the left a NEFCLASS system with two inputs, five rules and two output classes is shown

one should always try to use partial knowledge to avoid that all possible rules have to be created. If there are no known rules for certain input states, then only for these particular states all possible rules have to be created. This way the number of initial rule units can be reduced.

Incremental rule learning goes the opposite way, and creates a rule base from scratch by adding rule by rule. It does this by first classifying an input vector, i.e. finding that membership function for each variable that yields the highest membership value for the respective input value. By this a rule antecedent is formed. Then the algorithm tries to guess the output value by deriving it from the current fuzzy error. In a second phase the rule base is optimised by changing the conclusion to an adjacent membership function if necessary.

This idea provides a rule learning algorithm that is less expensive than the one previously described. It is not necessary to handle all possible rules at once, something that soon becomes impossible, especially if there are a lot of variables, or a lot of membership functions. After the rule base has been learned, the learning procedure for the fuzzy sets is invoked to tune the membership functions.

NEFCON has been implemented in several software tools under Unix, Windows and MATLAB/SIMULINK. On this conference there is another paper from our group that describes this implementation [19].

4 Neuro-Fuzzy Classification

In this section we discuss classification as another special case of function approximation. An input vector $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ is mapped to class indicator C which represents a crisp subset of $\mathbb{R}^n$. We assume the intersection of two different classes to be empty. The training data consists of a set of labelled data, i.e. for each training sample its correct class is known. A classification problem can be represented by a function

$$\varphi : \mathbb{R}^n \to \{0,1\}^m,$$

where $\varphi(\mathbf{x}) = \mathbf{c} = (c_1, \ldots, c_m)$ such that $c_i = 1$ and $c_j = 0$ $(j \in \{1, \ldots, m\}, j \neq i)$, i.e. $\mathbf{x}$ belongs to class C_i. This way, the class information is given by a 1-of-n code. If a fuzzy system is used to perform this task, the fuzzy rules look like this:

R : **if** x_1 is μ_1 **and** x_2 is μ_2 **and** $\ldots$ **and** x_n is μ_n
then pattern $(x_1, x_2, \ldots, x_n)$ belongs to class C,

where $\mu_1, \ldots, \mu_n$ are fuzzy sets which describe the patterns' features.

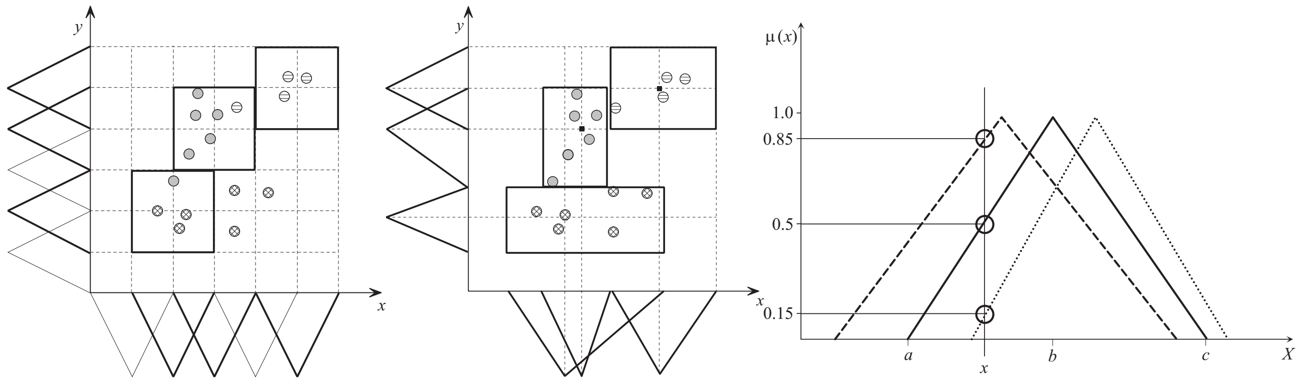

Figure 2: Visualization of a possible NEFCLASS learning process after rule creation (left), and after fuzzy set tuning (middle). The adaption of fuzzy sets (right) is carried out by simply changing the parameters of its membership function in a way that the membership degree for the current feature value is increased or decreased (middle fuzzy set: initial situation, left fuzzy set: increase situation, right fuzzy set: decrease situation)

Because of the mathematics involved in the rule evaluation process, the rule base actually does not approximate the above mentioned function φ but the function $\varphi' : \mathbb{R}^n \to [0,1]^m$. We will get $\varphi(\mathbf{x})$ by $\varphi(\mathbf{x}) = \psi(\varphi'(\mathbf{x}))$, where ψ reflects the interpretation of the classification result obtained from the fuzzy system. Usually, we will map the highest component of each vector $\mathbf{c}$ to 1 and its other components to 0 (winner takes all).

NEFCLASS (neuro fuzzy classification) [13] is a neuro-fuzzy model that is derived from the generic fuzzy perceptron presented above. An example for a NEFCLASS system is presented in Fig. 1. It shows a NEFCLASS system that classifies input patterns with two features into two distinct classes by using five linguistic rules. There are again shared weights on the connections from the input to the hidden layer. Compared to NEFCON and NEFPROX, the connections to the output layer are quite different, however. Each hidden (rule) unit is connected to exactly one output (class) unit. The weights on the connections from rule units to output units are fixed at 1 for semantical reasons (to avoid weighted rules).

The output value for an output unit (class), can either be computed as the mean of the activation values of all rule units it is connected to, or as the maximum of those values. The activation of a rule is the minimum of the membership values from its antecedent.

A NEFCLASS system can be built from partial knowledge about the patterns, and can then be refined by learning, or it can be created from scratch by learning. A user has to define a number of initial fuzzy sets partitioning the domains of the input features, and must specify the maximum number of rule nodes that may be created in the hidden layer.

The idea of the learning algorithm is to create a rulebase first, and then to refine it by modifying the initially given membership functions (usually fuzzy partitions where the membership degrees of each value add up to 1.0). The rulebase will be created by finding for each pattern in the training set a rule that best classifies it. If a rule with an identical antecedent is not already in the rule base, it will be added. After all patterns are processed once, the rule base is complete. It is then possible to evaluate the performance of each rule, and delete some of them (if there are too many) to keep only the best rules.

The learning algorithm of the membership functions uses the output error that tells, whether the degree of fulfillment of a rule has to be higher or lower. This information is used to change the input fuzzy sets by shifting the membership functions, and making their supports larger or smaller (see Fig. 2). By changing only the fuzzy set that delivered the smallest membership degree for the current pattern, the changes are kept as small as possible. It is easy to define constraints for the learning procedure, e.g. that fuzzy sets must not pass each other, or that they must intersect at 0.5, etc. Constraints like these help to obtain an interpretable rule base, but may cause a loss of performance in classification.

The learning process is visualized in Fig. 2. The left part shows the situation after the rule learning algorithm has terminated. The predefined fuzzy partitioning on both input variables defines a partitioning of the input space, created by overlapping hyper-boxes, where each hyper-box is formed by the Cartesian product of the supports of n fuzzy sets. Each hyper-box represents the support of an n-dimensional fuzzy set, i.e. the antecedent of a fuzzy rule. During rule learning, hyper-boxes are selected due to the distribution of the patterns. Each hyper-box is mapped to the class of the pattern which caused its selection (i.e. the rule conclusion is determined). After all patterns are processed, the mapping of hyper-boxes to classes is re-evaluated and changed, where necessary. After this only the "best" hyper-boxes (fuzzy rules) are kept.

After rule learning there are usually some patterns which are not classified, because their hyper-box (rule) was not included in the set of $k_{\max}$ best rules. There are usually also some misclassifications. It is the task of the fuzzy set learning algorithm to improve this situation. By modifying the membership functions, the predefined partitioning is distorted. This results in the situation shown in the right part of Fig. 2. Because the learning algorithm for the fuzzy sets is constrained (e.g. a fuzzy set must not pass a neighbor), it is possible, that some changes to the form of the hyper-boxes are not applicable. This is one reason that some classification errors can remain. Another reason can be a too small number of fuzzy rules. This can also lead to undesired forms of membership functions (e.g. too much overlapping). Considering the resulting situation in the right part of Fig. 2, it is probably better to accept four instead of three rules to avoid the extremely wide support of the leftmost fuzzy set over feature x.

From the viewpoint of the NEFCLASS architecture and the flow of data, the fuzzy sets are trained by a backpropagation-like algorithm: the error is propagated from the output units towards the input units and is used to change the membership function parameters, but there is no gradient information involved. The adaptivity of a NEFCLASS system is restricted, because of the initially given input fuzzy partions, which define the form and maximal number of clusters, and by the constraints that do not admit certain changes in the fuzzy sets.

To interpret a fuzzy rule base it is important that there are as few rules as possible, and that superfluous variables are not used in the rules. With the new pruning techniques of NEFCLASS [18] a rule base can be further improved. The rule editor of NEFCLASS supports the user with the rule simplification process. It checks the rule base for consistency after each pruning step, and tells the user which inconsistencies have occured. The user can decide what actions have to be taken in case of an inconsistency, for example which rule to delete, if there are two rules with identical antecedents, but different conclusions.

NEFCLASS offers five pruning strategies:

- Evaluate rules: The rule learning procedure of NEFCLASS is invoked again, and only the best k rules are kept. The user can specify a value for k. This strategy is useful, if the learning process was started with a large number of rules, and a many of them have a poor performance measure.
- Delete inputs: NEFCLASS computes the correlations of the input variables with the class information, and offers the possibility to delete input variables that have a low correlation to the output. The user can specify the absolute value for the minimum correlation. The deletion of inputs can lead to inconsistencies in the rule base. The rule editor will notify the user of this and support her or him in solving such problems.
- Delete rules: Rules that are only responsible for a few number of classifications can be deleted from the rule base, if they are not needed to cover assumptions in the data. The user can specify a percentage value that has to be reached by each rule to remain in the rule base.
- Delete antecedents: For each rule the antecedents that are not identical to the rule's degree of fulfillment in at least z percent, are deleted from the rule. The user can specify a value for z. This method can lead to an inconsistent rule base, which has to be repaired in the rule editor.
- Delete fuzzy sets: If a variable is partitioned by more than two fuzzy sets, sometimes the support for one or more of them can become quite large during learning. This can be seen as evidence, that such a fuzzy set is superfluous. The user can specify a percentage value, and all fuzzy sets that cover more percent of the domain are deleted, which leads to a reduction of variables in the antecedents. This procedure can also lead to an inconsistent rule base, which will be noticed by the rule editor.

Each of the five above mentioned pruning strategies is applied interactively by the user. It is possible to view the evaluations of the pruning strategy which shall be applied prior to using it. By this the user can decide on the appropriate parameter for the pruning method. After the pruning method was applied the rule editor is invoked and displays the new rule base. If pruning resulted in inconsistencies or other problems like identical rules, the user can solve these problems step by step. If the new rule base is not desirable, the pruning step can be cancelled. If the classification performance decreases after a pruning step, the learning process can be invoked to improve it again, if possible. By saving and restoring NEFCLASS systems the user can try different strategies to improve the rule base until a set of rules is found that yields acceptable performance and interpretability.

5 Conclusions

We reviewed three of our neuro-fuzzy approaches which are derived from the same generic model and follow the same principle: Keep the learning algorithms simple and do not touch the semantics of the underlying fuzzy systems. Other researchers may prefer other models, for example more sophisticated learning algorithms. However, more powerful learning strategies can be more difficult to handle. From our point of view neuro-fuzzy techniques should be used as tools, and not as automatic solution generators. Therefore we think it is important that the models and learning algorithms are easy to handle and that a user can easily interpret them.

From an applicational point of view, one could say: why bother with interpretability and semantics? It is important that the system does its job. It is of course possible to omit all constraints from the learning procedures of a neuro-fuzzy system, to consider it only as a convenient tool that can be initialized by prior knowledge and trained with sample data, and never to analyse the final system, as long as it performs to the satisfaction of the user. However, interpretability and clear semantics provide us with advantages like simple ways to check the system for plausibility and to maintain it during its life cycle.

It is also possible to consider even simpler learning mechanisms like weighted fuzzy rules, as they can be found in several commercial fuzzy shells. However, they give rise to some semantical problems, which we addressed in [14]. Allowing the weights to be selected from $[0, 1]$ could be interpreted as something like a degree of support for a rule, i.e. a value less than 1 would then indicate an ill-defined rule, that supports a class only to some extent. Some approaches allow the weights to assume any value in $\mathbb{R}$, but this leaves the semantics of fuzzy rules behind. It is not clear how rules weighted by absolute values greater than 1 or by negative values should be interpreted (for rules with negative weights sometimes an interpretation as *if not* rules is suggested [6]).

Rule weights can always be replaced by changes in the fuzzy sets of a rule. However these changes can lead to non-normal fuzzy sets and to situations in which identical linguistic values are represented differently in different rules. Rule weights can destroy the interpretation of a fuzzy system completely. Therefore we always refrain from learning weights in our approaches.

Our view of neuro-fuzzy approaches as heuristics to determine parameters of fuzzy systems by processing training data with a learning algorithm, is expressed by the list of five points at the end of Section 1. 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 generators". The user should always supervise the learning process and try to interpret its results. We also have to keep in mind that – like in neural networks – the success of the learning process is not guaranteed. The same guidelines for selecting and preprocessing training data that are known from neural networks apply to neuro-fuzzy systems. But if the applicaton of neuro-fuzzy methods is well-considered, they can be a powerful tool in the development process of fuzzy systems.

The approaches discussed in this paper are implemented in several software tools which can be obtained via the Internet at http://fuzzy.cs.uni-magdeburg.de.

References

[1] Hamid R. Berenji and Pratap Khedkar. Learning and tuning fuzzy logic controllers through reinforcements. *IEEE Trans. Neural Networks*, 3:724–740, September 1992.

[2] James C. Bezdek, Eric Chen-Kuo Tsao, and Nikhil R. Pal. Fuzzy Kohonen clustering networks. In *Proc. IEEE Int. Conf. on Fuzzy Systems 1992*, pages 1035–1043, San Diego, CA, 1992.

[3] James J. Buckley and Yoichi Hayashi. Fuzzy neural networks: A survey. *Fuzzy Sets and Systems*, 66:1–13, 1994.

[4] James J. Buckley and Yoichi Hayashi. Neural networks for fuzzy systems. *Fuzzy Sets and Systems*, 71:265–276, 1995.

[5] Dimiter Driankov, Hans Hellendoorn, and Michael Reinfrank. *An Introduction to Fuzzy Control*. Springer-Verlag, Berlin, 1993.

[6] Saman K. Halgamuge and Manfred Glesner. Neural networks in designing fuzzy systems for real world applications. *Fuzzy Sets and Systems*, 65:1–12, 1994.

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

[8] James M. Keller and Hossein Tahani. Backpropagation neural networks for fuzzy logic. *Information Sciences*, 62:205–221, 1992.

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

[10] Sushmita Mitra and Ludmilla Kuncheva. Improving classification performance using fuzzy mlp and two-level selective partitioning of the feature space. *Fuzzy Sets and Systems*, 70:1–13, 1995.

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

[12] 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.

[13] 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. 1995 ACM Symposium on Applied Computing, Nashville, Feb. 26–28*, pages 461–465. ACM Press, New York, February 1995.

[14] 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.

[15] Detlef Nauck and Rudolf Kruse. Neuro-fuzzy systems research and applications outside of Japan (in japanese). In M. Umano, I. Hayashi, and T. Furuhashi, editors, *Fuzzy-Neural Networks* (in Japanese), Soft Computing Series, pages 108–134. Asakura Publ., Tokyo, 1996.

[16] Detlef Nauck and Rudolf Kruse. Function approximation by nefprox. In *Proc. Second European Workshop on Fuzzy Decision Analysis and Neural Networks for Management, Planning, and Optimization (EFDAN'97)*, pages 160–169, Dortmund, June 1997.

[17] Detlef Nauck and Rudolf Kruse. Neuro-fuzzy systems for function approximation. In Adolf Grauel, Wilhelm Becker, and Fevzi Belli, editors, *Fuzzy-Neuro-Systeme'97 – Computational Intelligence. Proc. 4th Int. Workshop Fuzzy-Neuro-Systeme '97 (FNS'97) in Soest, Germany*, Proceedings in Artificial Intelligence, pages 316–323, Sankt Augustin, 1997. infix.

[18] 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.

[19] Andreas Nürnberger, Detlef Nauck, Rudolf Kruse, and Ludger Merz. A neuro-fuzzy development tool for fuzzy controllers under MATLAB/SIMULINK. In *Proc. Fifth European Congress on Intelligent Techniques and Soft Computing (EUFIT97)*, pages 1029–1033, Aachen, September 1997.

[20] S. K. Pal and S. Mitra. Multi-layer perceptron, fuzzy sets and classification. *IEEE Trans. Neural Networks*, 3:683–697, 1992.

[21] Witold Pedrycz and H. C. Card. Linguistic interpretation of self-organizing maps. In *Proc. IEEE Int. Conf. on Fuzzy Systems 1992*, pages 371–378, San Diego, CA, 1992.

[22] T. J. Procyk and E. H. Mamdani. A linguistic self-organizing process controller. *Automatica*, 15:15–30, 1979.

[23] Wu Zhi Qiao, Wang Pei Zhuang, Teh Hoon Heng, and Song Shou Shan. A rule self-regulating fuzzy controller. *Fuzzy Sets and Systems*, 47:13–21, 1992.

[24] Shihuang Shao. Fuzzy self-organizing controller and its application for dynamic processes. *Fuzzy Sets and Systems*, 26:151–164, 1988.

[25] Nadine Tschichold-Gürman. Generation and improvement of fuzzy classifiers with incremental learning using fuzzy rulenet. In K. M. George, Janice H. Carrol, Ed Deaton, Dave Oppenheim, and Jim Hightower, editors, *Applied Computing 1995. Proc. 1995 ACM Symposium on Applied Computing, Nashville, Feb. 26–28*, pages 466–470. ACM Press, New York, February 1995.

[26] Petri Vuorimaa. Fuzzy self-organizing map. *Fuzzy Sets and Systems*, 66:223–231, 1994.

[27] Hans Georg Zimmermann, Ralph Neuneier, Hubert Dichtl, and Stefan Siekmann. Modeling the german stock index dax with neuro-fuzzy. In *Proc. Fourth European Congress on Intelligent Techniques and Soft Computing (EUFIT96)*, Aachen, September 1996.