Original paper

Soft Computing 2 (1999) 168–182

Neuro-fuzzy control based on the NEFCON-model: recent developments

A. Nürnberger, D. Nauck, R. Kruse

Abstract Fuzzy systems are currently being used in a wide field of industrial and scientific applications. Since the design and especially the optimization process of fuzzy systems can be very time consuming, it is convenient to have algorithms which construct and optimize them automatically. One popular approach is to combine fuzzy systems with learning techniques derived from neural networks. Such approaches are usually called neuro-fuzzy systems. In this paper we present our view of neuro-fuzzy systems and an implementation in the area of control theory: the NEFCON-Model. This model is able to learn and optimize the rule base of a Mamdani like fuzzy controller online by a reinforcement learning algorithm that uses a fuzzy error measure. Therefore, we also describe some methods to determine a fuzzy error measure for a dynamic system. In addition we present some implementations of the model and an application example. The presented implementations are available free of charge for non-commercial purposes.

Key words Hybrid methods; Neuro-fuzzy system; System control; Neural network; fuzzy system.

1 Introduction

Fuzzy systems are currently being used in a wide field of industrial and scientific applications [12, 46]. Since the design and especially the optimization process of fuzzy systems can be very time consuming [28], it is convenient to have algorithms which construct and optimize them automatically. One popular approach is to combine fuzzy systems with learning techniques derived from neural networks. Such approaches are usually called neuro-fuzzy systems.

In the following, we describe our notion of a neuro-fuzzy system. Then we examine how to represent a fuzzy system in a neural-network-like architecture and discuss an implementation of a neuro-fuzzy system in the area of control theory.

A. Nürnberger, D. Nauck, R. Kruse
Otto-von-Guericke-University of Magdeburg, Faculty of Computer Science, Universitätsplatz 2, D-39106 Magdeburg, Germany
Tel.: +49.391.67.11358, fax: +49.391.67.12018,
E-mail: andreas.nuernberger@cs.uni-magdeburg.de
WWW: http://fuzzy.cs.uni-magdeburg.de.

Correspndence to: A. Nürnberger

The term *neuro-fuzzy systems* refers to combinations of neural networks and fuzzy systems. However, it is not completely clear what kind of models this term applies to. If we take a look into the relevant literature, we can find several different approaches which are called *neuro-fuzzy*, *neural fuzzy*, or *fuzzy-neuro* systems. In the following, we discuss the general idea of such approaches,and give a meaning to *neuro-fuzzy*, currently the most popular of the three aforementioned terms.

Before we present our interpretation of neuro-fuzzy systems we must discuss three other terms: *neural network*, *learning* and *fuzzy system*.

Neural networks

Neural networks – they are also called *connectionist systems* – are designed to (very roughly) model certain aspects of the human brain. They consist of simple processing elements (neurons) that exchange signals along connections (network structure). The signals are changed when they travel along the connections: They are combined (usually multiplied) with the (connection) weights. A neuron gathers the input from all connections leading to it, and computes an activation value by using a (usually non-linear) activation function. The units of a neural network are usually organized in layers which are called input layer, hidden layer(s) or output layer, depending on their functionality. A neural network with n units in its input layer and m units in its output layer, implements a mapping $f: I^n \rightarrow O^m$. The sets $I \subseteq \mathbb{R}$ and $O \subseteq \mathbb{R}$ are the input and output domain. The hidden layer(s) add to the complexity of a neural network, and they are important, if arbitrary mappings have to be represented.

The most interesting feature of neural networks is their ability to *learn*. There are so-called *learning algorithms* which can be used to determine the connection weights by processing a set of examples (the learning problem). The weights are learned in such a way that two similar input vectors (patterns) result in outputs which are similar to each other. The neural network must not memorize the examples, but it is supposed to generalize.

The goal of a learning algorithm is to minimize an error function that is defined over the difference between desired and actual output. However, the algorithm does not perform some kind of global optimization, but it computes only local weight modifications. The weights are modified according to local information, by distributing the global error inside the network. We can view a weight (or a connection) as an active

entity that changes itself due to the local conditions in its environment, independently of other weights. The sum of all modifications in the network is supposed to lead to a global error reduction.

Certain types of neural networks, like multilayer perceptrons or radial basis function networks, are universal approximators, i.e. they can approximate any continuous function on a compact domain to any given degree of accuracy [15, 40]. However, the desired solution cannot be found analytically. It has to be obtained by iteratively processing the training data with the learning algorithm. Success is not guaranteed: the algorithm can get stuck in local minima, for example. Even if the learning problem is solved, it is not obvious whether the network has generalized. The reaction of the network to new, unknown inputs must be tested on a set of test patterns. It is usually not possible to prove analytically that the network has learned the desired mapping. *A neural network is a black box – the networks learns, but the user does not learn anything from the network*. It is usually not possible to express the knowledge hidden in the network by some kind of rules, or to interpret the network in some other sense. This can be a severe drawback, if a neural network is to be used in an application, where questions of reliability and safety play a role. For an introduction to neural networks see for example [1, 14, 47].

Learning

As mentioned above neural networks are able to learn from data. Learning in this context means to find good values for parameters (weights) by an iterative procedure which processes sample data. The procedure is guided by an error measure that rates the current degree of performance. This is a very simple form of learning, and depending on the scientific background (e.g. philosophy, psychology, cognitive science, artificial intelligence, etc.) one may be reluctant to call this kind of iterative parameter determination a learning method. Learning as it is described has only marginal connections to learning as it occurs in human beings, or as it is examined in areas like machine learning. To be more precise we should speak of *statistical learning* or *learning from data*. It is also known as *supervised learning*, because there is a virtual supervisor or trainer, who can determine an output error.

There are other terms for this kind of parameter estimation. It may be more suitable to say that a neural network is *trained*, and therefore a neural network learning algorithm is also called a *training algorithm*. From the viewpoint of a statistician, a multilayer perceptron is a non-linear regression or discriminant model. In this case, learning (or training) means to iteratively update estimates.

In the following, we use the term learning or training in the sense described above: an iterative learning (training) algorithm tries to adjust parameters of a model by processing sample data (a learning problem). Two forms of supervised learning can be distinguished:

- (*Plain*) *Supervised learning* tries to reduce the difference between actual and desired output, and needs a fixed learning problem, i.e. a data set, where for each input pattern an output pattern is given.
- *Reinforcement learning* tries to produce outputs that have a certain observable effect to an environment. It needs a free learning problem, where there is no known output for a given input pattern, and an external reinforcement signal indicating whether the desired effect occured or not.

Finally, we have to define the term fuzzy system.

Fuzzy systems

We assume the reader to be familiar with terms like fuzzy set, fuzzy logic, fuzzy rule, t-norm and t-conorm. For an introduction see [21]. The kind of fuzzy system we discuss has nothing to do with fuzzy logic in the narrow sense, i.e. we do not consider systems of generalized logical rules. Certain types of fuzzy systems are – like neural networks – universal approximators [8, 20].

A fuzzy system consists of a set of fuzzy rules R_j like

$$R_j: \textbf{if } x_1 \text{ is } \mu_j^1 \textbf{ and } \ \dots \ \textbf{and } x_n \text{ is } \mu_j^n \textbf{ then } y \text{ is } \nu_j,$$

where $x_1, \dots, x_2 \in \mathbb{R}$ are input variables, and $y \in \mathbb{R}$ is an output variable. $\mu_j^i: \mathbb{R} \to [0, 1]$ and $\nu_j: \mathbb{R} \to [0, 1]$ are fuzzy sets which are labeled with linguistic terms like *small* or *approximately zero*. The fuzzy sets are usually represented by parameterized membership functions like triangular, trapezoidal or bell shaped functions. The **and** connective in the antecedent of the rule is evaluated by a t-norm, usually the minimum function. A complete set of rules is evaluated by max–min inference, resulting in an output fuzzy set ν:

$$\nu(y) = \max_{R_j} \{\min\{\mu_{j_1}^1(x_1), \dots, \mu_{j_n}^n(x_n), \nu_j(y)\}\}.$$

This evaluation procedure is well known from Mamdani controllers [26]. The output fuzzy set ν is then transformed into a crisp value by a defuzzification procedure, like, e.g. center of gravity or mean of maximum. There are various other ways to evaluate a set of fuzzy rules, see e.g. [23, 24] for an overview.

A fuzzy system like this approximates an unknown function $f: \mathbb{R}^n \to \mathbb{R}$. Mappings to $\mathbb{R}^m$ can be obtained straightforwardly by just adding variables to the conclusions of the rules. Each rule can be interpreted as a *fuzzy sample*, and the inference method results in an interpolation in a fuzzy environment [18]. This means fuzzy systems can be used for the same tasks as neural networks. The difference is that fuzzy systems are not created by a learning algorithm. They are built from explicit knowledge which is expressed in the form of linguistic (fuzzy) rules. However, it is sometimes difficult to specify all parameters of a fuzzy system (rules and membership functions). If the performance of the fuzzy system is not satisfactory, the parameters must be tuned manually. This tuning process is usually error prone and time consuming, e.g., small changes applied to the fuzzy set could result in a large change to the control behavior (see, for example [28], pp. 121).

So the idea of applying some kind of learning algorithm to a fuzzy system is not surprising. From the number of possible ways to accomplish this, the combination with neural network methods is very popular.

2 Neuro-fuzzy systems

A lot of approaches describing the combination of neural network and fuzzy systems are discussed in the literature. The preference of these so-called *neuro-fuzzy systems* is probably

due to the fact that neural networks and fuzzy systems – especially fuzzy controllers – became popular roughly at the same time, at the end of the 1980s. Applicants of fuzzy controllers, who had trouble tuning them, perhaps have admired the ostensible ease by which neural networks learned their parameters. On the other hand, neural network users may have admired the transparency and interpretability of a rule-based fuzzy system, while a neural network is only a black box.

Typical situations in which a fuzzy system or a neural network would be used can be described as follows:

Fuzzy system
There is (at least some) knowledge about the relation between input and output, i.e. for some input situations one can (vaguely) specify the outputs. This knowledge can be described by fuzzy rules.

Neural network
There is a lot of training data that describes the input/output relation of the problem, and there is little or nothing known about this relation.

If we decide to use a fuzzy system, and find out that we cannot derive all parameters from our knowledge about the problem, then we may think of learning them. However, if we plan to use neural network techniques to do this, we must have a lot of training data. This means we are in a classical situation to apply a neural network. Consider Table 1: using a fuzzy system has obviously some benefits over using a neural network. We can interpret a fuzzy system as a system of linguistic rules. This means we can at least to some extent check our solution for plausibility. A neural network is a black box to the user. If we have at least some prior knowledge we can use it to initialize a fuzzy system. A neural network always learns from scratch. So it makes sense to use a fuzzy system, and instead of tuning it manually, using some kind of learning algorithm to optimize its parameters.

A common way to apply a learning algorithm to a fuzzy system is to represent it in a special neural-network-like architecture, which is quite easy as we show in the next section. Then a learning algorithm – like, for example backpropagation – is used to train the system. There are some problems, however. Neural network learning algorithms are usually gradient descent methods. They cannot be applied directly to a fuzzy system, because the functions used to realize the inference process are usually not differentiable. There are two solutions to this problem:

(a) the functions used in the fuzzy system (like min and max) are replaced by differentiable functions, or
(b) instead of a standard neural learning algorithm a better suited procedure is used.

Table 1. Comparison between fuzzy systems and neural networks

Fuzzy system	Neural network
Interpretable	Black box
Making use of linguistic knowledge	Learning from scratch

Modern neuro-fuzzy systems are usually represented as a multilayer feedforward neural network [5, 7, 9, 10, 13, 16, 28, 32, 33, 43]. The well-known ANFIS model by Jang [16], for example, implements a Sugeno-like fuzzy system [42] in a network structure, and applies a mixture of backpropagation and least-mean-square procedure to train the system. A Sugeno-like fuzzy system uses only differentiable functions, so ANFIS goes for solution (a). The GARIC model [5] also chooses solution (a) by using a special "soft minimum" function which is differentiable. The problem with solution (a) is that the models are sometimes not as, easy to interpret as e.g. Mamdani-type fuzzy systems. Other models, that we discuss in the following sections, try solution (b) – they are Mamdani-type fuzzy systems and use special learning algorithms.

To stress the common feature of all these approaches, and to give the term *neuro-fuzzy system* a suitable meaning, we want to restrict it to systems which posses the following properties:

(i) A neuro-fuzzy system is a fuzzy system that is trained by a learning algorithm (usually) derived from neural network theory. The (heuristical) 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.
(ii) 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.

    Some neuro-fuzzy models use more than three 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.
(iii) 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 initialize it by prior knowledge in the form of fuzzy rules.
(iv) 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.
(v) 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 [21].

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-organizing feature map to find fuzzy rules [39](cooperative models), or to view a fuzzy system as a special neural network and to apply a learning algorithm directly [32] (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.

3
A fuzzy system in a neural network structure: The generic fuzzy perceptron

A lot of neuro-fuzzy approaches represent their models as a neural network. This is, of course, not necessary to be able to apply a learning algorithm to a fuzzy system. However, it can be convenient because it visualizes 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 the form of a network, and a neural network development tool is available that is flexible enough to let the user 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.

It is very easy to represent a fuzzy system as a neural network. In Fig. 1 a multilayer perceptron that solves the XOR-problem is presented. The network uses sigmoid activation functions in the hidden and output units. The weights and bias values were found by backpropagation. The other network is, in fact, a fuzzy system with two rules. It is also an acceptable solution to the XOR-problem. The activation functions of the units are min and max, respectively, and the connection weights are fuzzy sets. To the right of the network the fuzzy sets *b* (*big*) and *s* (*small*), and the two fuzzy rules represented in the network are shown.

To describe the network structure of a neuro-fuzzy model in general, it is useful to have a generic model. In [nauck95b] a *generic* 3-*layer fuzzy perceptron* is presented. 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 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 (see Definition 1). 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.

The term *fuzzy* (*multi-layer*) *perceptron* has also been used by other authors for their approaches [17, 27, 38]. 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.

Definition 1 (*Generic* 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:

(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

$$W : U \times U \to \mathscr{F}(\mathbb{R}),$$

such that there are only connections $W(u,v)$ with $u \in U_i$, $v \in U_{i+1} (i \in \{1, 2\})$. $\mathscr{F}(\mathbb{R})$ is the set of all fuzzy subsets of $\mathbb{R}$.

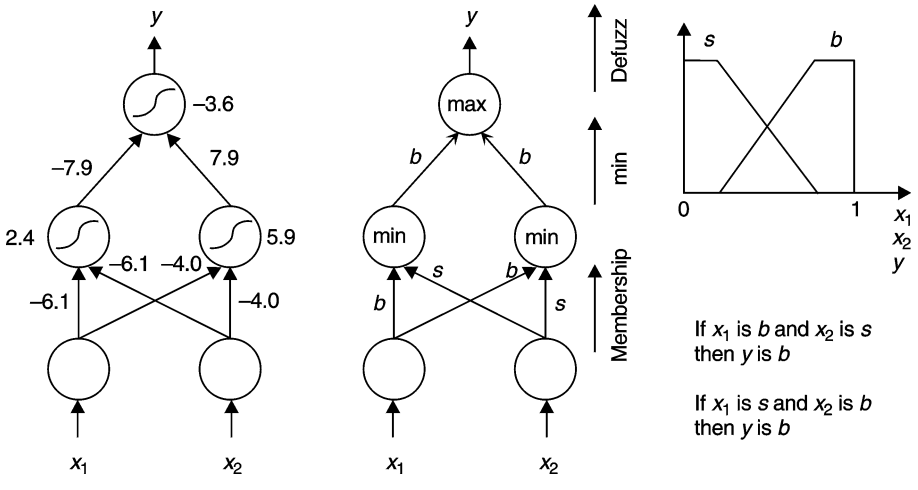


Fig. 1. A multilayer perceptron with sigmoid activation functions that solves the XOR problem, and a fuzzy system for the same problem represented as a feedforward multilayer neural network with special activation and propagation functions

(iii) 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 $u \in U_1 \cup U_2$:

$$A_u : \mathbb{R} \to \mathbb{R}, \quad a_u = A_u(\text{net}_u) = \text{net}_u,$$

(b) for output units $u \in U_3$:

$$A_u : \mathscr{F}(\mathbb{R}) \to \mathscr{F}(\mathbb{R}),$$

$$a_u = A_u(\text{net}_u) = \text{net}_u.$$

(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 $u \in U_1 \cup U_2$:

$$O_u : \mathbb{R} \to \mathbb{R}, \quad o_u = O_u(a_u) = a_u,$$

(b) for output units $u \in U_3$:

$$O_u : \mathscr{F}(\mathbb{R}) \to \mathbb{R},$$

$$o_u = O_u(a_u) = \text{DEFUZZ}_u(a_u),$$

where DEFUZZ_u is a suitable defuzzification function.
(v) NET defines for each unit $u \in U$ a propagation function NET_u to calculate the net input net_u
(a) for input units $u \in U_1$:

$$\text{NET}_u : \mathbb{R} \to \mathbb{R}, \quad \text{net}_u = \text{ex}_u,$$

(b) for rule units $u \in U_2$:

$$\text{NET}_u : (\mathbb{R} \times \mathscr{F}(\mathbb{R}))^{U_1} \to [0, 1],$$

$$\text{net}_u = \top_{u' \in U_1} \{W(u', u)(o_{u'})\},$$

where $\top$ is a t-norm,
(c) for output units $u \in U_3$:

$$\text{NET}_u : ([0, 1] \times \mathscr{F}(\mathbb{R}))^{U_2} \to \mathscr{F}(\mathbb{R}),$$

$$\text{net}_u : \mathbb{R} \to [0, 1],$$

$$\text{net}_u(x) = \bot_{u' \in U_2} \{\top(o_{u'}, W(u', u)(x))\},$$

where $\bot$ is a t-conorm.
(vi) $\text{ex} : U_1 \to \mathbb{R}$, defines for each input unit $u \in U_1$ its external input $\text{ex}(u) = \text{ex}_u$. For all other units ex is not defined.

A fuzzy perceptron can be viewed 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 on the whole, out of linguistic (fuzzy if–then) rules.

In the following, we present a realization of a neuro-fuzzy system based on this generic fuzzy perceptron in the area of control theory. This generic model was also used to derive a neuro-fuzzy system for classification [31] and function approximation [34].

4 The NEFCON model

NEFCON [30] is a model for neural fuzzy controllers developed by our group, and it is based on the architecture of the generic fuzzy perceptron described above. The learning algorithm for NEFCON is based on the idea of reinforcement learning. Figure 2 shows a NEFCON system with two input variables, one output variable and five rules. The input variables ξ_1 and ξ_2 are state variables of a technical system S which has to be controlled. NEFCON's output η is the control action applied to S. The units of the hidden layer represent fuzzy rules. Rule unit R_3 for instance stands for the rule

R_3: **If** ξ_1 **is** $A_2^{(1)}$ **and** ξ_2 **is** $A_2^{(2)}$ **then** η **is** B_2,

where $A_2^{(1)}$, $A_2^{(2)}$ and B_2 are linguistic terms represented by the fuzzy sets $\mu_2^{(1)}$, $\mu_2^{(2)}$ and ν_2. As specified in Definition 1 the connections in NEFCON are weighted with fuzzy sets instead of real numbers, and some connections always have the same weight (linked connections, shared weights). In Fig. 2 the connections from input unit ξ_1 to the rule units R_1 and R_2 share the weight $\mu_1^{(1)}$ and the connections from R_4 and R_5 to η have the common weight ν_3. This is illustrated by the ellipses drawn around the connections. The learning algorithm must take this weight sharing into account and carry out identical modifications on linked connections to ensure the integrity of the rule base.

Definition 2 Consider a process S with n state variables $\xi_1, \ldots, \xi_n$ and a control variable η. For each ξ_i there are p_i linguistic terms, and for η there are q linguistic terms. Let there be k linguistic rules which partially describe a control function for S. A NEFCON system (see Fig. 2) is a special

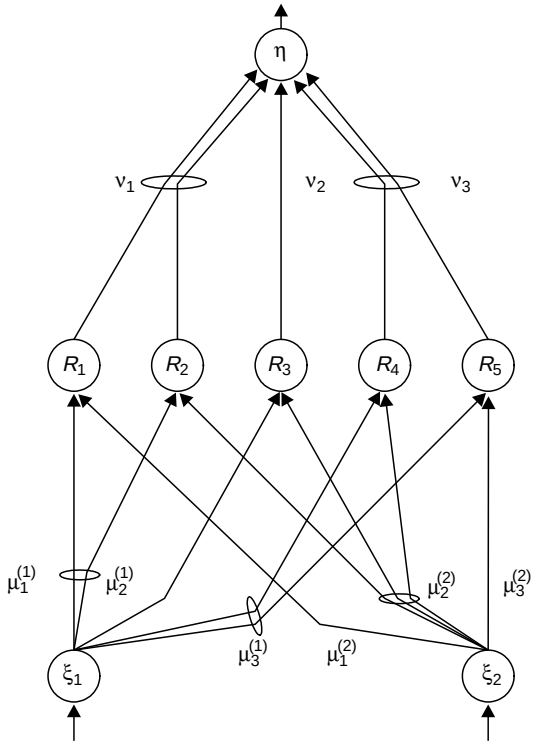


Fig. 2. A NEFCON system with two input variables and five rules

3-layer fuzzy perceptron with the following specifications:

(i) $U_1 = \{\xi_1, \ldots, \xi_n\}$, $U_2 = \{R_1, \ldots, R_k\}$, $U_3 = \{\eta\}$.
(ii) Each connection between units $\xi_i \in U_1$ and $R_r \in U_2$ is labelled with a linguistic term $A_{j_r}^{(i)} (j_r \in \{1, \ldots, p_i\})$.
(iii) Each connection between units $R_r \in U_2$ and the output unit η is labeled with a linguistic term $B_{j_r} (j_r \in \{1, \ldots, q\})$.
(iv) Connections that come from the same input unit ξ_i $(i \in 1, \ldots, n)$ and have identical labels, always carry the same fuzzy weight. These connections are called linked connections and their weight is called shared weight. An analogous condition holds for the connections leading to the output unit η.
(v) Let $L_{u,v}$ denote the label of the connection between a unit u and a unit v. For all units $v, v' \in U_2$

$$(\forall u \in U_1)(L_{u,v} = L_{u,v'}) \Rightarrow v = v'$$

holds.

This definition makes it possible to interpret a NEFCON system in terms of a fuzzy controller. Condition (iv) specifies that there have to be *shared weights*. If this feature is missing, it would be possible for fuzzy weights which represent identical linguistic terms to evolve differently during the learning process. If this is allowed to happen, the architecture of the NEFCON system can no longer be understood as a fuzzy rule base. Condition (v) determines that there are no rules with identical antecedents. A network that does not adhere to this condition is inconsistent. During decremental rule learning (see below) we allow this situation for some time, but after rule learning the NEFCON system must be consistent and condition (v) must hold [29, 30].

If k fuzzy rules are known to control a process S we can construct a NEFCON system as follows:

- For each input variable ξ_i an input unit with this name is entered into the input layer.
- The control variable is represented by a single output unit η.
- For each fuzzy rule R_r there is a hidden (rule) unit with this name.
- Each rule unit R_r is connected to the input units and to the output unit in accordance with the fuzzy rule it represents. The fuzzy set $\mu_{j_r}^{(i)}$ (v_{j_r}) that represents the linguistic term $A_{j_r}^{(i)}$ (B_{j_r}) in the antecedent (consequent) of the rule R_r is chosen as the connection weight. The linguistic term is used to label the connection.
- The t-norm and t-conorm to compute the network inputs for the units (Definition 1(v)) and the defuzzification procedure (Definition 1(iv)) must be chosen in accordance with the fuzzy system that will be represented by the NEFCON system.

The knowledge base of the fuzzy system is implicitly given by the network structure. The input units assume the task of the fuzzification interface, the inference logic is represented by the propagation functions, and the output unit is the defuzzification interface.

5
Learning algorithms for NEFCON

The learning process of the NEFCON model can be divided into two main phases. The first phase is designed to learn an initial rule base, if no prior knowledge about the system is available. Furthermore, it can be used to complete a manually defined rule base. The second phase optimizes the rules by shifting or modifying the fuzzy sets of the rules. Both phases use a fuzzy error e, which describes the quality of the current system state, to learn or to optimize the rule base. The fuzzy error plays the role of the critic element in reinforcement learning models (e.g. [2, 4]). In addition the sign of the optimal output value η_{opt} must be known. So the extended fuzzy error E is defined as

$$E(x_1, \ldots, x_n) = \text{sgn}(\eta_{\text{opt}}) e(x_1, \ldots, x_n)$$

with the crisp input $(x_1, \ldots, x_n)$.

The updated NEFCON learning algorithms learn and optimize the rule base of a Mamdani-like fuzzy controller (see Sect. 1). The fuzzy sets of the antecedents and consequents can be represented by any symmetric membership function.

5.1
Learning a rule base

Methods to learn an initial rule base can be divided into three classes: Methods starting with an empty rule-base [35, 44], methods starting with a "full" rule base (combination of every fuzzy set in the antecedents with every consequent) [29] and methods starting with a random rule base [25]. Up to now, we implemented algorithms of the first two classes.

Modified algorithm NEFCON I (decremental rule learning)

The modified algorithm NEFCON I starts with a "full" rule base. The first version of this algorithm was published in [29]. The algorithm can be divided into two phases which are executed during a fixed period of time or a fixed number of iteration steps. During the first phase, rules with an output sign different from that of the optimal output value η_{opt} are removed. During the second phase, a rule base is constructed for each control action by selecting randomly one rule from every group of rules with identical antecedents. The error of each rule (the output error of the whole network weighted by the activation of the individual rule) is accumulated. At the end of the second phase from each group of rule nodes with identical antecedents the rule with the least error value remains in the rule base. All other rule nodes are deleted. In addition, rules used very rarely are removed from the rule base. The original algorithm used triangular membership functions, while the improved implementation also supports trapezoidal and Gaussian membership functions. Besides, the algorithm was enhanced for dynamic systems which need a static offset.

The following definition specifies this decremental rule learning algorithm. Let $\text{Ant}(R_r)$ denote the antecedent and $\text{Con}(R_r)$ the consequent of a fuzzy rule corresponding to the rule unit R_r. $\mathscr{R}$ denotes the set of all rule units.

Definition 3 (*Decremental rule learning*) Let S be a process with n state variables $\xi_i \in X_i$ $(i \in \{1, \ldots, n\})$ which are partitioned by p_i fuzzy sets each and a control variable $\eta \in Y$ partitioned by q fuzzy sets. Let there also be an NEFCON system with $N = q \prod_{i=1}^{n} p_i$ initial rule units with

$$(\forall R, R' \in \mathscr{R})((\text{Ant}(R) = \text{Ant}(R') \wedge \text{Con}(R) = \text{Con}(R')) \Rightarrow R = R').$$

The decremental rule learning algorithm for NEFCON is given by the following steps.

(i) For each rule unit R_r a counter C_r (initialized to 0) is defined ($r \in \{1, \ldots, N\}$). For a fixed number m_1 of iterations the following steps are carried out:
  (a) Determine the current NEFCON output o_η using the current state of S.
  (b) For each rule R_r determine its contribution t_r to the overall output o_η ($r \in \{1, \ldots, N\}$).
  (c) Determine $\mathrm{sgn}(\eta_{\mathrm{opt}})$ for the current input values.
  (d) Delete each rule unit R_r with $\mathrm{sgn}(t_r) \neq \mathrm{sgn}(\eta_{\mathrm{opt}})$ and update the value of N.
  (e) Increment the counters C_r for all R_r with $o_{R_r} > 0$.
  (f) Apply o_η to S and get the new input values from S.

(ii) For each R_r a counter Z_r (initialized to 0) is defined. For a fixed number m_2 of iterations the following steps are carried out:
  (a) From all subsets

$$\mathscr{R}_j = \{R_r | Ant(R_r) = Ant(R_s)(r, s \in \{1, \ldots, N\})\} \subseteq \mathscr{R}$$

  one rule unit R_{r_j} is selected randomly.
  (b) Determine the NEFCON output o_η using only the rule units selected and the current state of S.
  (c) Apply o_η to S and get the new input values from S.
  (d) Determine the contribution t_{r_j} of each selected rule unit R_{r_j} to the overall output value o_η ($r_j \in \{1, \ldots, N\}$).
  (e) Determine $\mathrm{sgn}(\eta_{opt})$ using the new input values.
  (f) Add $|o_{R_{r_j}} \cdot E|$ to the counter Z_{r_j} of each selected rule unit R_{r_j}.
  (g) For all selected rule units R_{r_j} with $o_{R_{r_j}} > 0$, C_{r_j} is incremented.

(iii) Delete all rule units R_{s_j} for all subsets $\mathscr{R}_j$ from the network for which there is a rule unit $R_{r_j} \in \mathscr{R}_j$ with $Z_{r_j} < Z_{s_j}$, and delete all rule units R_r with $C_r \leqslant \beta \cdot (m_1 + m_2)$, $0 \leqslant \beta < 1$, from the network, and update the value of N.

Delete all rules that have not been used in more than in p% of all propagations.

The bottom-up-algorithm (incremental rule learning)

The bottom-up-algorithm starts with an empty rule base. An initial fuzzy partitioning of the input and output intervals must be given. The algorithm can be divided into two phases. During the first phase, the rules' antecedents are determined by classifying the input values, i.e. finding that membership function for each variable that yields the highest membership value for the respective input value. Then the algorithm tries to "guess" the output value by deriving it from the current fuzzy error. During the second phase the rule base is optimized by changing the consequent to an adjacent membership function, if this is necessary.

Definition 4 (*Incremental rule learning*) Let S be a process with n state variables $\xi_i \in X_i$ ($i \in \{1, \ldots, n\}$) which are partitioned by p_i fuzzy sets each and a control variable $\eta \in [y_{\min}, y_{\max}]$ partitioned by q fuzzy sets. Let there also be an initial NEFCON system with k' predefined rule units, where k' may be zero. The *incremental rule learning* algorithm for NEFCON is given by the following steps.

(i) For a fixed number m_1 of iterations the following steps are carried out:
  (a) For the current input vector $(x_n, \ldots, x_n)$ find those fuzzy sets $\hat{\mu}^{(1)}, \ldots, \hat{\mu}^{(n)}$ for which

$$(\forall i, j)(\hat{\mu}^{(i)}(x_i) \geqslant \mu_j^{(i)}(x_i))$$

  holds.
  (b) If there is no rule with the antecedent

  If ξ_1 is $\hat{\mu}^{(1)}$ and … and ξ_n is $\hat{\mu}^{(n)}$,

  then find the fuzzy set $\hat{v}$ such that $(\forall j)(\hat{v}(o) \geqslant v_j(o))$ holds, where the heuristic output value o is determined by

$$o = \begin{cases} m + |E| \cdot (y_{\max} - m) & \text{if } E \geqslant 0, \\ m - |E| \cdot (m - y_{\min}) & \text{if } E < 0 \end{cases} \qquad m = \frac{y_{\max} + y_{\min}}{2}.$$

  (c) Enter the rule

  If ξ_1 is $\hat{\mu}^{(1)}$ and … and ξ_n is $\hat{\mu}^{(n)}$ then η is $\hat{v}$

  into the NEFCON system.

(ii) For a fixed number m_2 of iterations the following steps are carried out:
  (a) Propagate the current input vector through the NEFCON system and estimate for each rule unit R_r the contribution t_r to the output value o_η. Compute the desired contribution to the system output value by

$$t_r^\star = t_r + \sigma \cdot o_r \cdot E,$$

  where E is the extended fuzzy error, and $\sigma > 0$ is a learning rate.
  (b) For each rule unit R_r determine the new output membership function $\hat{v}_r$ such that

$$(\forall j)(\hat{v}_r(t_r^\star) \geqslant v_j(t_r^\star))$$

  and change the consequent of the rule unit R_r accordingly.

(iii) Delete all rules that have not been used in more than in p% of all propagations.

This definition provides a rule learning algorithm that is less expensive than the decremental rule learning procedure. It is not necessary to handle all possible rules at once, which becomes impossible anyway when there are a lot of variables or a lot of membership functions. Nevertheless, the Bottom-Up-Algorithm can have problems with complex dynamic systems, because of the heuristic approach of finding the consequents.

Discussion

The decremental rule learning algorithm can only be used, if there are only a few input variables with not too many fuzzy sets. Time and memory needed to apply the algorithm rise exponentially with the number of variables. For NEFCON with four input variables and one output variable that are each partitioned by seven fuzzy sets there are $7^5 = 16\,807$ possible rules (if this number is not reduced by using prior knowledge). With such a large number of rules to be evaluated during rule learning NEFCON is probably not capable of running in real time. Therefore, it is advisable to use the incremental rule learning procedure for larger NEFCON systems.

It cannot be expected that the rule base found by one of the two algorithms will resemble a rule base given by a human expert who can control the process of interest. First, there may

be more than one rule base that is able to control the process sufficiently, and second, as in all neural-network-like learning procedures the learning outcome depends on heuristic factors like learning rate, duration of the learning procedure, and the error measure.

The rule base learned by NEFCON should not be seen as a final solution, but as supporting information towards a solution. It is conceivable that counterintuitive rules are manually deleted or replaced after learning, and adaptation of the fuzzy sets is continued. It is often not necessary to start the learning process from scratch. Two measures can make use of partial knowledge for the learning process and reduce the cost of the learning procedure, especially for decremental rule learning:

- If for a given antecedent a consequent is known, then an equivalent rule unit is put into the NEFCON system, and the learning algorithm is not allowed to remove it. Other rules with this antecedent are not created.
- If for some states only a subset of all possible rules need to be considered, then only these rules are put into the network.

Incremental rule learning also benefits from prior knowledge. If rules are known in advance and entered into the NEFCON system, then rule learning does not need to start from scratch. A user can determine that a given rule must not be deleted during rule learning, or the learning procedure can have the freedom to delete or modify rules entered in advance.

5.2 Optimization of a rule base

The algorithms presented in this section are designed to optimize a rule base of a fuzzy controller by shifting and/or modifying the support of the fuzzy sets. They do not modify the rules or the structure of a given network.

In the following, we assume that the fuzzy sets $\mu_j^{(i)}$ or v_j representing the linguistic terms of the state variables $\xi_i \in X_i \subseteq \mathbb{R}$ or the control variable $\eta \in Y \subseteq \mathbb{R}$, respectively, are defined as follows:

$$v_j(y) \stackrel{\text{def}}{=} \mu_j^{(i)}(x) \stackrel{\text{def}}{=} \begin{cases} \dfrac{x - a_j^{(i)}}{b_j^{(i)} - a_j^{(i)}} & \text{if } x \in [a_j^{(i)}, b_j^{(i)}], \\ \dfrac{c_j^{(i)} - x}{c_j^{(i)} - b_j^{(i)}} & \text{if } x \in [b_j^{(i)}, c_j^{(i)}], \\ 0 & \text{otherwise,} \end{cases}$$

with $a_j^{(i)}, b_j^{(i)}, c_j^{(i)} \in \mathbb{R}$, $a_j^{(i)} \leqslant b_j^{(i)} \leqslant c_j^{(i)}$. (1)

This means that $v_j(a_j) = \mu_j^{(i)}(a_j^{(i)}) = 0$, $v_j(b_j) = \mu_j^{(i)}(b_j^{(i)}) = 1$, and $v_j(c_j) = \mu_j^{(i)}(c_j^{(i)}) = 0$.

The algorithms presented in the following sections are defined by use of triangular membership functions, but it is easily possible to use symmetric fuzzy sets in the consequents and the antecedents (see, for example [36]).

Modified algorithm NEFCON-I

The algorithm NEFCON I is motivated by the backpropagation algorithm for the multilayer perceptron. The extended fuzzy error E is used to optimize the rule base by "reward and punishment". A rule is "rewarded" by shifting its consequent to a higher value and by widening the support of the antecedents, if its current output has the same sign as the optimal output η_{opt}. Otherwise the rule is "punished" by shifting its consequent to a lower value and by reducing the support of the antecedents.

Definition 5 (*NEFCON-I*) Let S be a process with n state variables $\xi_i \in X_i$ ($i \in \{1, \ldots, n\}$) and one control variable $\eta \in Y$. For each input variable ξ_i there are p_i triangular fuzzy sets $\mu_j^{(i)}$ ($j \in \{1, \ldots, p_i\}$), and for the output variable η there are q triangular fuzzy sets v_j ($j \in \{1, \ldots, q\}$) (see (1)). For the fuzzy sets there are constraints Ψ that determine whether a modification can be carried out. The fuzzy error backpropagation learning algorithm NEFCON I for adapting the membership functions of a NEFCON system with k rule units iterates the following steps until a stop criterion is met.

(i) Determine the output value o_η, and apply it to the process S to obtain a new state.

(ii) Determine the extended fuzzy error E according to the new state of S.

(iii) Estimate the contribution t_r of each rule node R_r to the output value o_η and calculate the fuzzy rule error E_{R_r} for each rule unit R_r ($r \in \{1, \ldots, k\}$):

$$E_{R_r} = o_{R_r} E \operatorname{sgn}(t_r).$$

(iv) Determine the modifications for the consequent fuzzy sets v_{j_r}:

$$\Delta b_{j_r} = \sigma o_{R_r} E,$$

$$\Delta a_{j_r} = \Delta b_{j_r},$$

$$\Delta c_{j_r} = \Delta b_{j_r}.$$

where $\sigma > 0$ is a learning rate. Modify v_{j_r} such that all constraints $\Psi(v_{j_r})$ are met.

(v) Determine the modifications for the antecedent fuzzy sets $\mu_{j_r}^{(i)}$:

$$\Delta a_{j_r}^{(i)} = -\sigma E_{R_r}(b_{j_r}^{(i)} - a_{j_r}^{(i)}),$$

$$\Delta c_{j_r}^{(i)} = \sigma E_{R_r}(c_{j_r}^{(i)} - b_{j_r}^{(i)}).$$

Modify $\mu_{j_r}^{(i)}$ such that all constraints $\Psi(\mu_{j_r}^{(i)})$ are met.

The constraints Ψ can be used, for example, to restrict the domains of the parameters, to ensure that fuzzy sets have to overlap for a certain amount, or that fuzzy sets should not "overtake" each other during optimization.

A further improvement to the algorithm described in Definition 5 is to shift the antecedents in addition to widening or reducing the support. This can be realized by modifying Definition 5(v):

Definition 6 (*NEFCON-Ia*) Based on the assumptions of Definition 5 the algorithm NEFCON-Ia is defined by replacing (v) of Definition 5 with (v'):

(v') Determine the modifications for the antecedent fuzzy sets $\mu_{j_r}^{(i)}$:

$$\Delta b_{j_r}^{(i)} = \sigma E_{R_r}(\xi_i - b_{j_r}^{(i)}) \cdot (c_{j_r}^{(i)} - a_{j_r}^{(i)}),$$

$$\Delta a_{j_r}^{(i)} = -\sigma E_{R_r}(b_{j_r}^{(i)} - a_{j_r}^{(i)}) + \Delta b_{j_r}^{(i)},$$

$$\Delta c_{j_r}^{(i)} = \sigma E_{R_r}(c_{j_r}^{(i)} - b_{j_r}^{(i)}) + \Delta b_{j_r}^{(i)}.$$

Modify $\mu_{j_r}^{(i)}$ such that all constraints $\Psi(\mu_{j_r}^{(i)})$ are met.

Modified algorithm NEFCON-II

In contrast to the algorithm NEFCON-I, which uses only the current fuzzy error E, the algorithm NEFCON-II also makes use of the change of the fuzzy error to optimize the rule base (see also Sect. 6). This is a heuristic approach to include the dynamics of the system into the optimization process.

Let E be the extended fuzzy error at time t and E' the extended fuzzy error at time $t+1$, then the error tendency τ is defined as

$$\tau = \begin{cases} 1 & \text{if } (|E'| \geqslant |E|) \wedge (E'E \geqslant 0), \\ 0 & \text{if } (|E'| < |E|) \wedge (E'E \geqslant 0), \\ -1 & \text{if } (E'E < 0). \end{cases}$$

If $\tau = 0$, the dynamic system moves to an optimal (or simply stable) state. In this case the rule base will not be modified. If $\tau = 1$, the error is rising without changing its sign. The output of each rule is increased by shifting its consequents. The antecedents of rules with consequents increasing the output will be "rewarded", while those with consequents decreasing the output will be "punished". If $\tau = -1$, the system has overshot. The output is decreased and the antecedents "punished" or "rewarded" accordingly.

Definition 7 (*NEFCON-II*) Let S be a dynamical system with n state variables $\xi_i \in X_i$, and one control variable $\eta \in Y$ (for the ease of computation we consider $Y = \{y_{\min}, \ldots, y_{\max}\}$ to be finite). For each input variable ξ_i there are triangular fuzzy sets $\mu_j^{(i)}, j \in \{1, \ldots, p_i\}$, and for the output variable η there are triangular fuzzy sets $v_j, j \in \{1, \ldots, q\}$. For the fuzzy sets there are constraints Ψ given that determine whether a modification can be carried out. The centers of the triangular membership function are denoted by c. To adapt the membership functions of a NEFCON system with k rule units, the following steps are iterated until an end criterion is met (e.g. when the fuzzy error stays small for an acceptable time).

(i) Determine E and $\operatorname{sgn}(\eta_{opt})$ from the current state.

(ii) Determine the absolute contribution ac_r and the relative contribution rc_r of each rule node r to the output value o_η (o_r is the output value (activation, degree of fulfillment) of rule node R_r):

$$ac_r := v_{j_r}^{-}(o_r),$$

$$rc_r := \frac{o_\eta - ac_r}{y_{\max} - y_{\min}}$$

where $v_{j_r}^{-}(o_r)$ is a pseudo-inverse of v_{j_r} at o_r, and it is calculated by a local defuzzification procedure:

$$v_{j_r}^{-}(o_r) = \frac{\sum_{\substack{y \in Y \\ v_{j_r}(y) > 0}} y \min(o_r, v_{j_r}(y))}{\sum_{\substack{y \in Y \\ v_{j_r}(y) > 0}} \min(o_r, v_{j_r}(y))}$$

(iii) Determine the output value o_η, and apply it to the dynamical system S to obtain a new state.

(iv) Determine the new extended fuzzy error E' and the error tendency τ according to the new state of S.

(v) Determine the shift for the conclusion fuzzy sets v_{j_r}:

$$\Delta c_{j_r} := \tau \cdot \operatorname{sgn}(\eta_{\text{opt}}) \, |E'| \, o_r \, |o_\eta - ac_r| \, \sigma,$$

where $\sigma > 0$ is a learning rate. Modify v_{j_r}, if all constraints $\Psi(v_{j_r})$ are met.

(vi) Determine the shift for the antecedent fuzzy sets $\mu_{j_r}^{(i)}$ (x_i is the current input value for ξ_i):

$$\Delta c_{j_r}^{(i)} := \tau \operatorname{sgn}(\eta_{\text{opt}}) \, (x_i - c_{j_r}^{(i)}) \, |E'| \, rc_r \, \sigma.$$

Modify $\mu_{j_r}^{(i)}$, if all constraints $\Psi(\mu_{j_r}^{(i)})$ are met.

Discussion

The ideas of the learning algorithms presented above are always the same: increase the influence of a rule if its action points into the right direction (rewarding), and decrease its influence if a rule behaves counterproductively (punishing). If the rule base is suitable, then a punishment situation will only occur if the process overshoots. Both learning algorithms presented are used solely to adapt the membership functions, and can be applied only if there is already a rule base of fuzzy rules. These rules can be directly transformed into a NEFCON system or learned by the algorithms presented in the prior section.

The algorithm NEFCON-II optimizes a rulebase more specific than the algorithm NEFCON-I, but it is more sensitive to a time delay between control action and system response (see also the credit assignment problem [4]). In such cases the algorithm NEFCON-I should be preferred.

All the algorithms described above are not designed to find an optimal solution for a given control problem, since one of the main objectives of our research is to develop algorithms that are able to determine online an appropriate and interpretable rule base within a small number of simulation runs. Besides it must be possible to use prior knowledge to initialize the learning process. This is a contrast to "pure" reinforcement strategies [4] or methods based on dynamic programming [3, 41], which try to find an optimal solution using neural network structures. These methods need many runs to find even an approximate solution for a given control problem. On the other hand, they have the advantage that they need less information about the error of the current system state. However, in many cases a simple error description can be achieved with little effort. We present some methods to determine a fuzzy error measure of a dynamic system in the following section.

6 Description of the system error

In case of a simple dynamic system the error can be described sufficiently well by simply using the difference between the reference signal and the system response. In case of more complex and sensitive systems the error must be described more exactly to obtain a satisfying rule base with the presented algorithms.

An optimal state of a system S can be described by a vector $x_{\text{opt}} = (x_1^{(\text{opt})}, \ldots, x_n^{(\text{opt})})$ of system state variable values [28]. The optimal state is reached if all state variables have assumed the values given by this vector. However, the system state is usually also considered to be acceptable or good if those values are only reached approximately.

Furthermore, we can distinguish two cases in evaluating the current system state. In the first case the system S is in a state

where all state variables have assumed approximately optimal values. In the second case they may have unacceptable values, but they compensate each other because S is going to assume optimal values in all state variables in the very near future. We call this a *compensatory state*. Example 1 explains the description of the *system error* based on the considerations mentioned above. It concerns the problem of the inverted pendulum, where a pole is mounted on a cart such that it can move back and forth. The pole must be kept upright by moving the cart.

Example 1 We consider an inverted pendulum. The current system error of the pendulum can be interpreted as acceptable or *zero*, if

(a) both the angle and the angular velocity are approximately zero,
(b) the angle is not zero, but the value of the angular velocity indicates a movement that will suitably reduce the angle.

Furthermore, the state can be described as *bad* if

(c) the angle is not zero, and the value of the angular velocity indicates a movement that will enlarge the angle.

The system error e can be described formally based on the considerations described above [28]. But, it can be difficult in some cases to describe all compensatory states in the presence of many state variables. Furthermore, the system state is usually also considered to be acceptable or *good* if the optimal system state is only reached approximately. It is therefore suitable to describe the goodness of the system state vaguely by using fuzzy rules, as described in the following.

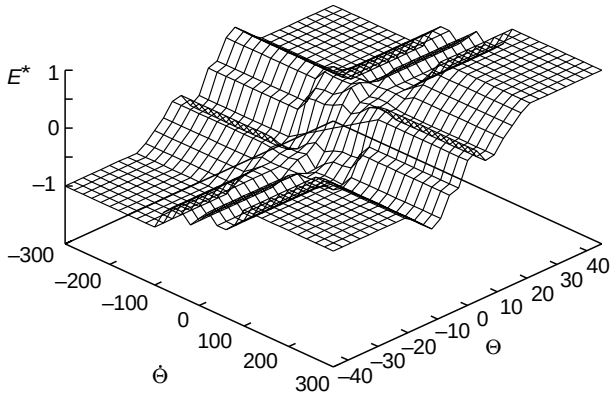


Fig. 3. Rule-based fuzzy error

6.1 Linguistic error description

Instead of specifying the system error in a formal mathematical way, it is usually more convenient to use linguistic (fuzzy) rules. By this, the system error can be called a *fuzzy error*. To do this, the state variables and the interval $[-1, 1]$ must be partitioned with fuzzy sets. The interval $[-1, 1]$ is the domain of the extended error measure E that already includes information concerning the direction of the (otherwise unknown) optimal control output η_{opt} (see Sect. 5). The fuzzy error rules are evaluated in the same way as the control rules.

An example of the extended fuzzy error of a NEFCON system is shown in Fig. 3 over the state space of an inverted pendulum. The representation is based on the fuzzy error rule base and the outline of the fuzzy partitions of the variables given in Fig. 4.

The short forms *po*, *pz*, *nz* and *ne* denote the linguistic terms *positive*, *positive zero*, *negative zero* and *negative*. As can be seen in the illustration the fuzzy sets for the variables θ, $\dot{\theta}$ and E are chosen such that an evenly distributed partition is obtained.

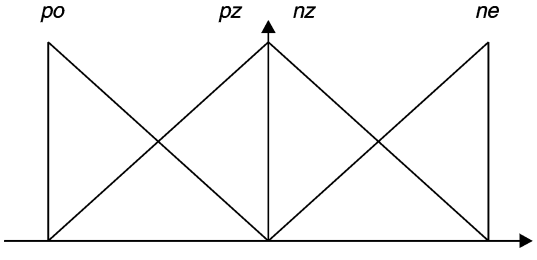
If the error is given by a fuzzy rule base, it is possible to use a second NEFCON system to compute the error value. This second NEFCON system can then also assume the role of an adaptive critic and can be trained. However, we do not consider this option further.

A refined possibility to define the fuzzy error is to use an error description with "fuzzy boundaries".

6.2 An error description with "fuzzy boundaries"

The error description with "fuzzy boundaries" makes it possible to describe a "soft" region for the system response which satisfies our request to the system behavior in a simple and intuitive way. Furthermore, it enables the definition of a time-dependent error measure. This error measure is defined by a interval (boundary) surrounding the required system response (reference signal) for every time step (see Fig. 5).

The calculated error signal remains zero, if the response signal of the dynamic system stays within the given interval surrounding the reference signal (see the thick lines in Figure Fig. 5 which presents an error description for a simple switch signal). If the signal leaves the bounds of the interval, an extended fuzzy error by use of a linguistic error description is computed, as described above. By this way a user can define areas where a certain deviation from the desired course is tolerated, which is even necessary in most real applications.



Θ̇ \ Θ	ne	nz	pz	po
ne	ne	ne	ne	pz
nz	ne	nz	pz	po
pz	ne	nz	pz	po
po	nz	po	po	po

Fig. 4. Sample of a rule base for the description of the extended fuzzy error for an inverted pendulum

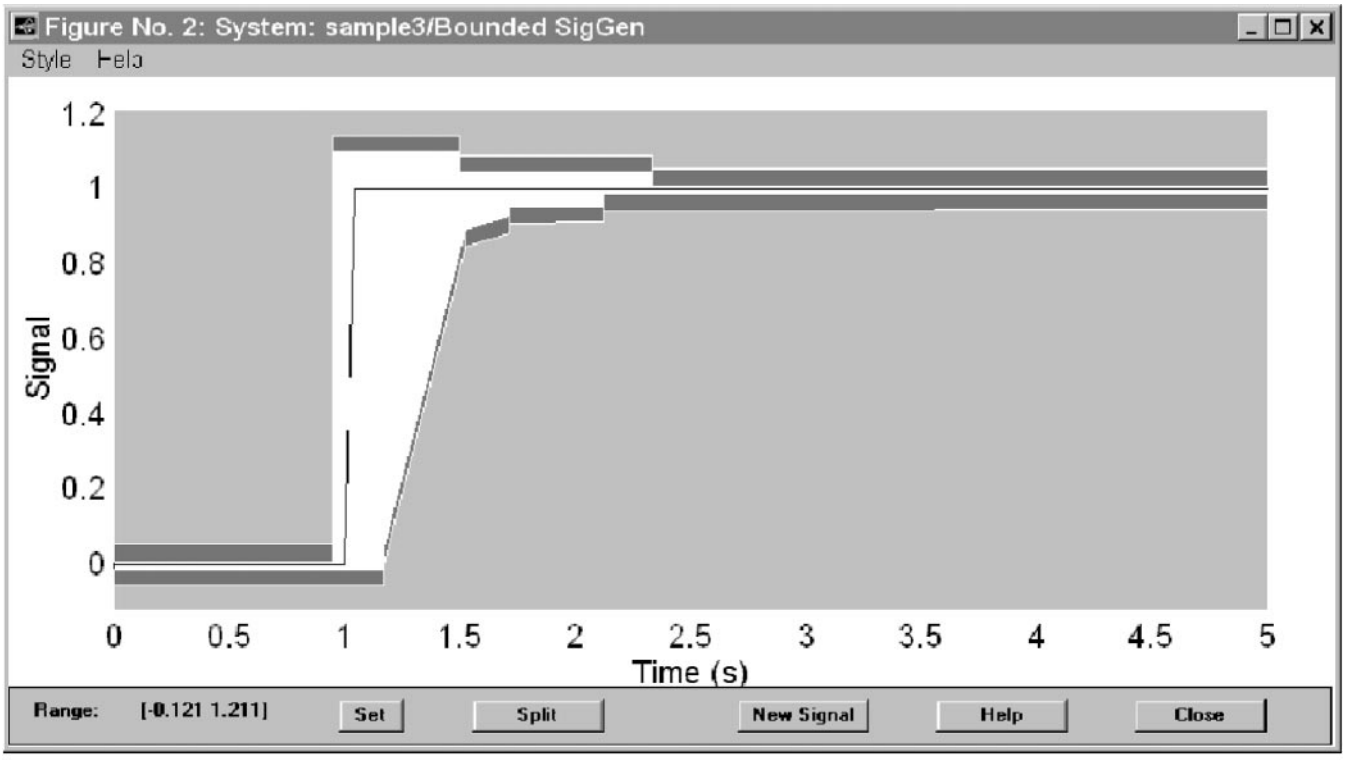


Fig. 5. The NEFCON bounded signal generator under MATLAB

7 Implementations and results

The NEFCON model has been implemented in several software tools under Unix, Microsoft Windows and MATLAB/SIMULINK. Most of the software can be obtained free of charge for non-commercial purposes from our WWW server (http://fuzzy.cs.uni-magdeburg.de). The implementation packages include simple tutorial applications like an inverted pendulum to try out the learning algorithms. In Fig. 6 the surface of the tool under Windows is shown displaying the solution (fuzzy rules and fuzzy sets) the tool has found to control an inverted pendulum.

The aim of the implementation under MATLAB/SIMULINK was to develop an interactive tool for the construction and optimization of a fuzzy controller in an industrial like simulation environment. By this, NEFCON can be applied easily to different plant models. Besides, the implementation enables the user to include prior knowledge into the system, to stop and to resume the learning process at any time, and to modify the rule base and the optimization parameters interactively. A graphical user interface was designed to support the user during the development process of the fuzzy controller. The optimized fuzzy controller can be detached from the development environment and can be used in real-time environments.

NEFCON for MATLAB is written in MATLAB's own programming language and is therefore platform-independent. In addition to MATLAB the NEFCON toolbox needs SIMULINK, the Fuzzy Toolbox, and the NCD Blockset Library. NEFCON can be conveniently operated via a graphical user interface and can use all features of MATLAB. The controlled process is implemented in the SIMULINK environment [37]. Fig. 7 presents the simulation environment of a sample application during the optimization phase of the algorithm. This example was created under Microsoft Windows NT 4.0.

An application example

In this section we consider NEFCON for MATLAB applied to a PT_2 system [37] that is given by the following differential equation:

$$\frac{1}{\omega_0^2}\ddot{x}+\frac{2D}{\omega_0}\dot{x}+x=Vy.$$

For the constants we chose $\omega_0=\sqrt{50}$, $D=\frac{2}{5}\sqrt{2}$ and $V=1$. The PT_2 system models the behavior of a two-mass system, for example, spring damper combinations or revolution controls for electric motors. In classical control theory such systems are controlled by a PI or PID controller. A comparison to fuzzy controllers for this problem is considered, for example, in [19].

As the desired course for y a reference signal $\hat{y}$ is provided by the NEFCON signal generator (see Fig. 5). The NEFCON signal generator also allows a variation of the fuzzy error to be used for the learning process. The desired course of a process state variable (here y) is defined by fuzzy boundaries as described in Sect. 6.

To control the PT_2 system a NEFCON system with three input variables $\xi_1=e=\hat{y}-y$, $\xi_2=\dot{e}$, $\xi_3=y$, and one output variable $\eta=x$ is created. Each input variable is partitioned by three triangular fuzzy sets and the output variable has five triangular fuzzy sets. Incremental rule learning and the fuzzy set learning algorithm NEFCON-II were used.

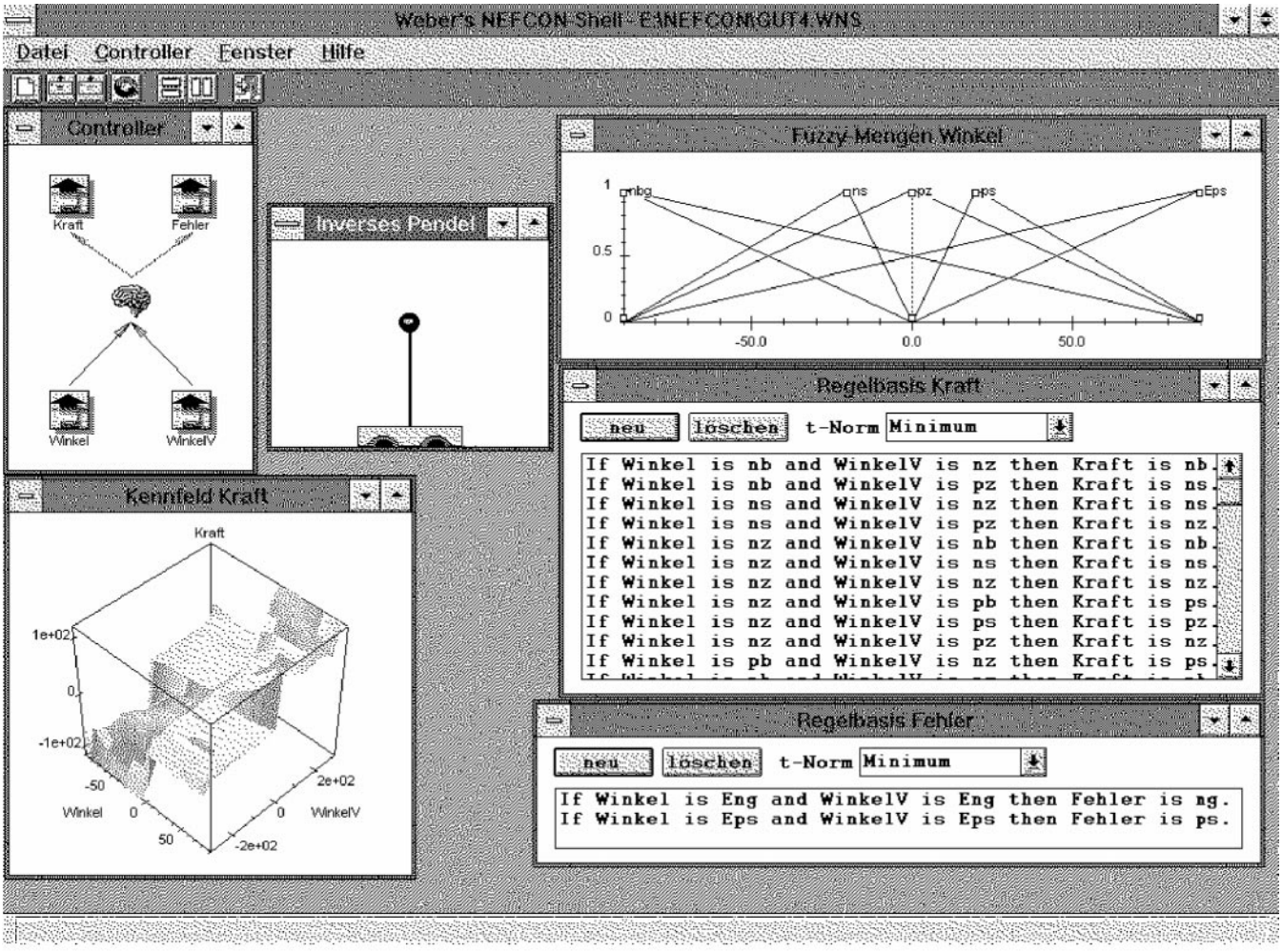


Fig. 6. An implementation of NEFCON under Windows

In Fig. 8 the performance of the controller can be seen. The whole learning process was applied for seven cycles, where each cycle is equivalent to 5 s of simulation (167 steps). Rule learning was done in the first three cycles in which Gaussian noise was added to the control output in order to cover the state space of the process better. Fuzzy set learning began in cycle 4. After cycle 7 the controller can drive the process quite nicely along the desired course. However, the control behavior is a little bit "uneasy" in the constant part after the jump. But this is implicitly tolerated by the error boundaries defined by the bounded signal generator (see Fig. 5). The rule base that was learned by NEFCON has 25 rules. The resulting fuzzy sets are shown in Fig. 9.

8 Conclusions

All learning methods presented in this paper 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 [6, 7]. 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 application 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 analyze 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 are addressed in [28, 32]. Therefore, we always refrain from learning weights in our approaches.

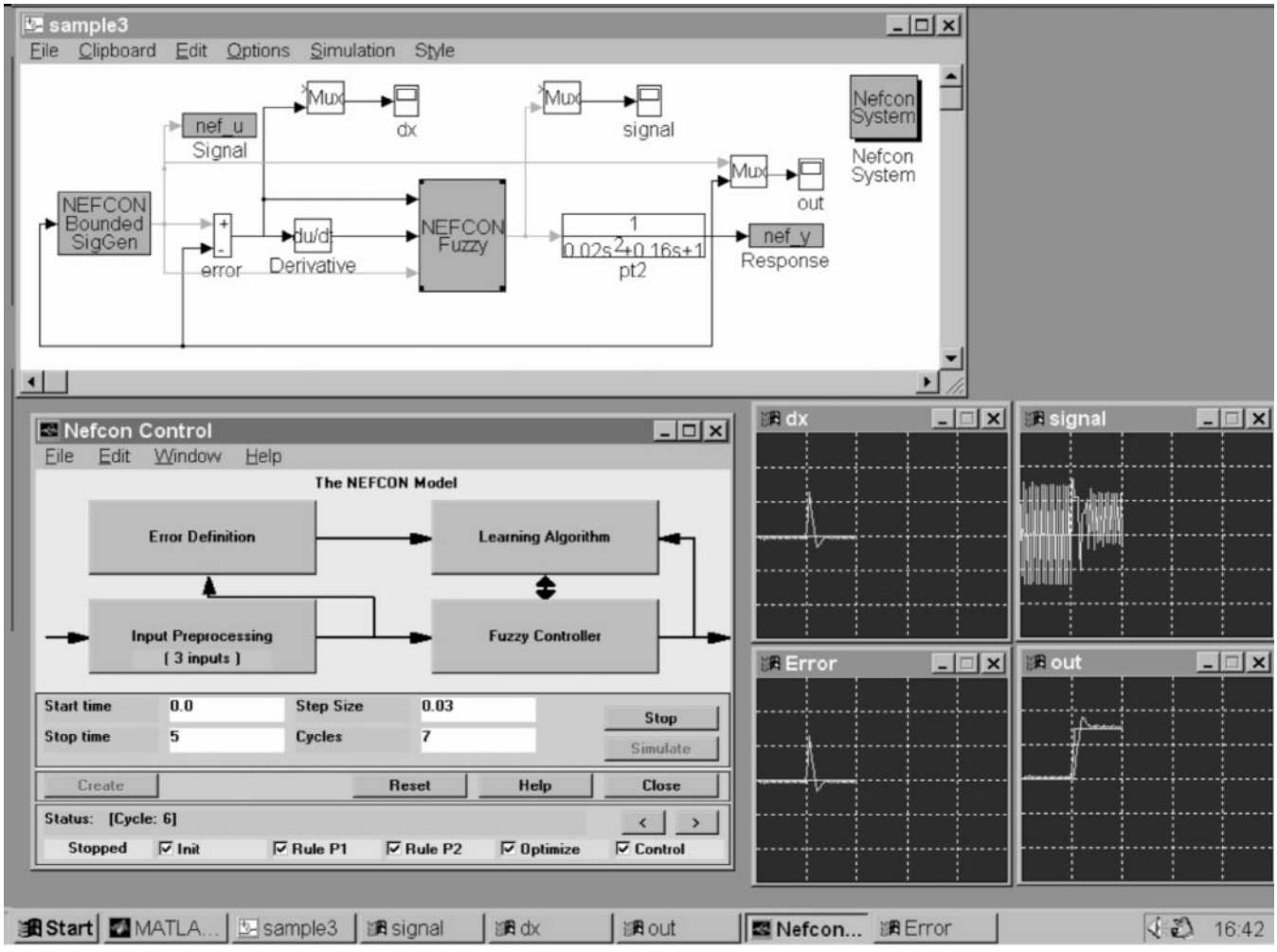


Fig. 7. Example of a simulation environment using NEFCON for MATLAB

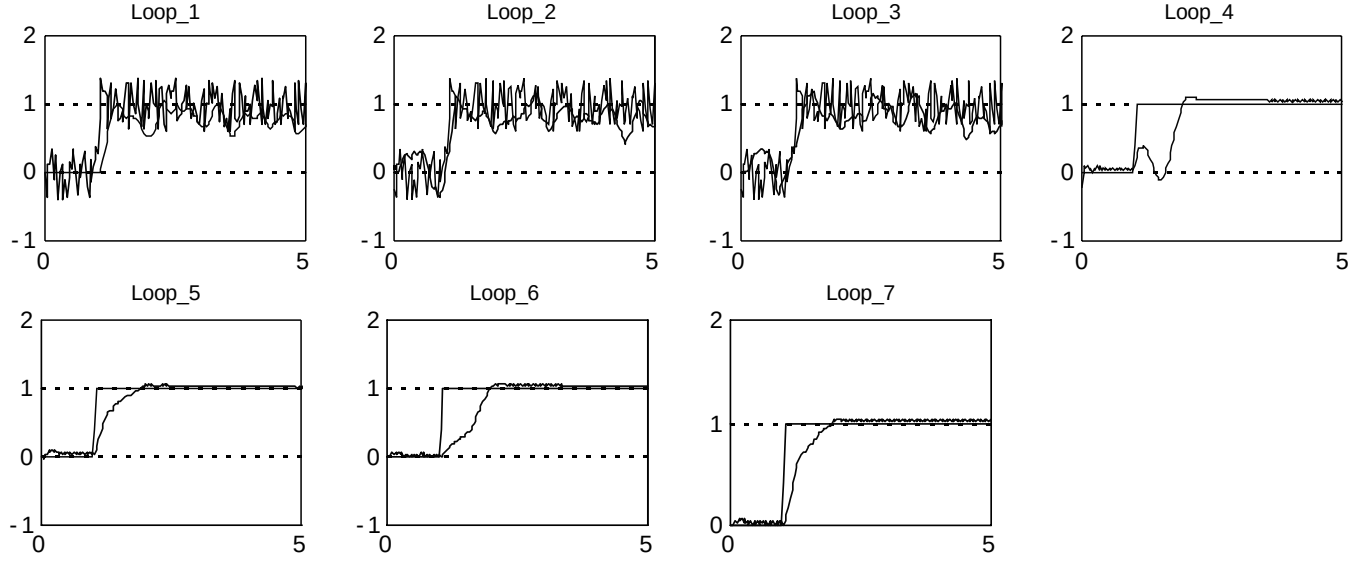


Fig. 8. The learning outcome of NEFCON for MATLAB applied to a PT_2 system

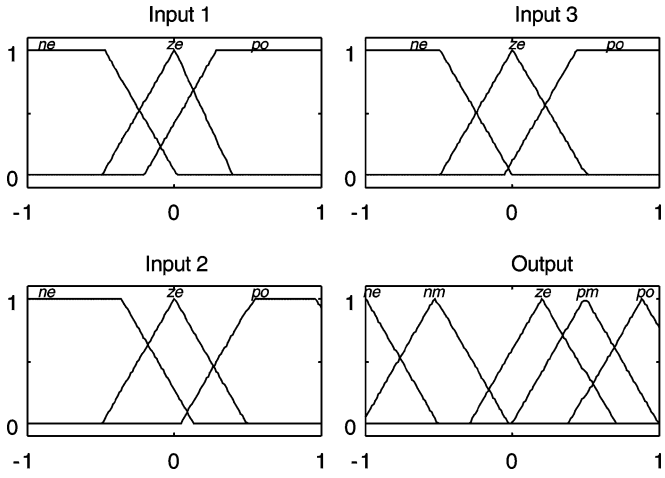


Fig. 9. The fuzzy sets learned by NEFCON for MATLAB applied to a PT_2 system

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 Sect. 2. 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 application of neuro-fuzzy methods is well considered, they can be a powerful tool in the development process of fuzzy systems.

References

1. **Aleksander I, Morton H** (1990) An Introduction to Neural Computing, Chapman & Hall, London
2. **Barto AG** (1992) Reinforcement learning and adaptive critic methods, In [45], pp 469–491. Van Nostrand, Reinhold
3. **Barto AG, Bradtke S, Singh S** (1995) Learning to act using real-time dynamic programming, Artificial Intell, Special Vol: Comput Res Interaction Agency, 1(72), 81–138
4. **Barto AG, Sutton RS, Anderson CW** (1983) Neuronlike adaptive elements that can solve difficult learing control problems, IEEE Trans. Man Cybernet 13, 834–846
5. **Berenji HR, Khedkar P** (1992) Learning and tuning fuzzy logic controllers through reinforcements, IEEE Trans Neural Networks, 3, 724–740
6. **Brown M, Bossley K, Harris C** (1995) An analysis of the application of B-spline neuro-fuzzy construction algorithms, Proc 3rd European Congress on Intelligent Techniques and Soft Computing (EUFIT95), vol 3, pp 1830–1834
7. **Brown M, Harris C** (1994) Neurofuzzy Adaptive Modelling and Control, New York: Prentice-Hall
8. **Buckley JJ** (1993) Sugeno type controllers are universal controllers, Fuzzy Sets and Systems, 53, 299–303
9. **Buckley JJ, Hayashi Y** (1994) Fuzzy neural networks: a survey, Fuzzy Sets and Systems, 66, 1–13
10. **Buckley JJ, Hayashi Y** (1995) Neural networks for fuzzy systems, Fuzzy Sets and Systems, 71, 265–276
11. **Chawdry P, Roy R, Pant RK** (Eds) (1997) Soft Computing in Engineering Design and Manufacturing, London UK: Springer
12. **de Silva CW** (1995) Intelligent Control: Fuzzy Logic Applications, Boca Raton, FL: CRC Press Inc.
13. **Halgamuge SK, Glesner M** (1994) Neural networks in designing fuzzy systems for real world applications, Fuzzy Sets and Systems, 65, 1–12
14. **Haykin S** (1994) Neural Networks. A Comprehensive Foundation, New York: Macmillan
15. **Hornik M, Stinchcombe M, White H** (1989) Multilayer feedforward networks are universal approximators. Neural Networks, 2, 359–366
16. **Jang JSR** (1993) ANFIS: Adaptive-network-based fuzzy inference systems, IEEE Trans Systems Man Cybernet 23, 665–685
17. **Keller JM, Tahani H** (1992) Backpropagation neural networks for fuzzy logic, Inform Sci, 62, 205–221
18. **Klawonn F, Kruse R** (1995) Fuzzy control on the basis of equality relations with an example from idle speed control, IEEE Trans Fuzzy Systems, 336–350
19. **Knappe H** (1994) Comparison of conventional and fuzzy-control of non-linear systems, In [22], pp 75–87, Braunschweig: Vieweg
20. **Kosko B** (1992) Fuzzy systems as universal approximators. Proc IEEE Int Conf on Fuzzy Systems San Diego, CA, pp 1153–1162
21. **Kruse R, Gebhardt J, Klawonn K** (1994) Foundations of Fuzzy Systems, Chichester: Wiley
22. **Kruse R, Gebhardt J, Palm R** (Eds) (1994) Fuzzy Systems in Computer Science, Braunschweig: Vieweg
23. **Lee CC** (1990) Fuzzy logic in control systems: Fuzzy logic controller, part i, IEEE Trans Systems Man Cybernet, 20, 404–418
24. **Lee CC** (1990) Fuzzy logic in control systems: fuzzy logic controller, part ii, IEEE Trans Systems Man Cybernet, 20, 419–435
25. **Lin C-T** (1994) Neural Fuzzy Control Systems with Structure and Parameter Learning, Singapore: World Scientific
26. **Mamdani EH, Assilian S** (1975) An experiment in linguistic synthesis with a fuzzy logic controller, Int J Man Mach Stud, 7, 1–13
27. **Mitra S, Kuncheva L** (1995) Improving classification performance using fuzzy mlp and two-level selective partitioning of the feature space, Fuzzy Sets and Systems, 70, 1-13
28. **Nauck D. Klawonn F, Kruse R** (1997) Foundations of Neuro-Fuzzy Systems, Chichester: Wiley
29. **Nauck D, Kruse R** (1993) A fuzzy neural network learning fuzzy control rules and membership functions by fuzzy error backpropagation, Proc IEEE Int Conf on Neural Networks San Franscisco pp 1022–1027
30. **Nauck D, Kruse R** (1994) NEFCON-1: An X-Window based simulator for neural fuzzy controllers, Proc IEEE Int Conf Neural Networks 1994 at IEEE WCCI'94, Orlando, FL, pp 1638–1643
31. **Nauck D, Kruse R** (1995) NEFCLASS – a neuro-fuzzy approach for the classification of data, In: George K, Carrol JH, Deaton E, Oppenheim D, Hightower J. (Eds.) Applied Computing 1995. Proc 1995 ACM Symp on Applied Computing, Nashville, New York: ACM Press, pp 461–465
32. **Nauck D, Kruse R** (1996) Designing neuro-fuzzy systems through backpropagation, In: Pedrycz W (Ed) Fuzzy Modelling: Paradigms and Practice, Boston: Kluwar, pp 203–228
33. **Nauck D, Kruse R** (1996) Neuro-fuzzy systems research and applications outside of Japan (in Japanese), In: Umano M, Hayashi I, Furuhashi T (Eds) Fuzzy-Neural Networks (in Japanese), Soft Computing Series, Tokyo: Asakura Publ. pp 108–134
34. **Nauck D, Kruse R** (1997) Neuro-fuzzy systems for function approximation, In: Grauel A, Becker W, Belli F. (Eds.) Fuzzy-Neuro-Systems'97 – Computational Intelligence, Proc 4th Int Workshop Fuzzy-Neuro-Systeme '97 (FNS'97) in Soest, Germany, Proceedings in Artificial Intelligence, Sankt Augustin, pp 316–323 infix
35. **Nauck D, Kruse R, Stellmach R** (1995) New Learning algorithms for the neuro-fuzzy environment NEFCON-1, Proc Neuro-Fuzzy-Systems'95, Darmstadt, pp 357–364
36. **Nürnberger A, Nauck D, Kruse R** (1997) Neuro-fuzzy control based on the NEFCON model under MATLAB/SIMULINK, In [11] London: Springer
37. **Nürnberger A, Nauck D, Kruse R, Merz L** (1997) A neuro-fuzzy development tool for fuzzy controllers under MATLAB/SIMULINK, Proc 5th European Congress on Intelligent Techniques and Soft Computing (EUFIT97), Aachen

38. **Pal S, Mitra S** (1992) Multi-layer perceptron, fuzzy sets and classification, IEEE Trans Neural Networks, 3, 638–697
39. **Pedrycz W, Card HC** (1992) Linguistic interpretation of self-organizing maps, Proc IEEE Int Conf on Fuzzy Systems 1992, San Diego, CA pp 371–378
40. **Poggio T, Girosi F** (1989) A theory of networks for approximation and learning, A.I. Memo 1140 Cambridge, MA: MIT
41. **Riedmiller M, Janusz B** (1995) Using eural reinforcement controllers in robotics, Proc 8th Australien Conf on Artificial Intelligence 1995, Canberra, Australia
42. **Sugeno M** (1985) An Introductory survey of fuzzy control, Inform Sci 36, 59–83
43. **Tschichold-Gürman N** (1995) Generation and improvement of fuzzy classifiers with incremental learning using fuzzy rulenet, In: George K, Carrol JH, Deaton E, Oppenheim D, Hightower J. (Eds) Applied Computing 1995, Proc 1995 ACM Sym on Applied Computing, Nashville, pp 466–470, New York: ACM Press
44. **Tschichold-Gürman N** (1996) Rule Net – A new knowledge-based artificial neural network model with application examples in robotics, PhD thesis, ETH Zürich
45. **White DA, Sofge DA** (Eds) (1992) Handbook of Intelligent Control. Neural, Fuzzy, and Adaptive Approaches, New York: Van Nostrand Reinhold
46. **Yen J, Langari R, Zadeh LA** (Eds) (1995) Industrial Applications of Fuzzy Logic and Intelligent Systems, Piscataway: IEEE Press
47. **Zurada JM** (1992) Introduction to Artificial Neural Systems, St. Paul MN: West Publishing Company