Combining Neural Networks and Fuzzy Controllers

Detlef Nauck, Frank Klawonn, and Rudolf Kruse

Dept. of Computer Science, Technical University of Braunschweig
Bueltenweg 74 – 75, D-38106 Braunschweig, Germany

Abstract. Fuzzy controllers are designed to work with knowledge in the form of linguistic control rules. But the translation of these linguistic rules into the framework of fuzzy set theory depends on the choice of certain parameters, for which no formal method is known. The optimization of these parameters can be carried out by neural networks, which are designed to learn from training data, but which are in general not able to profit from structural knowledge. In this paper we discuss approaches which combine fuzzy controllers and neural networks, and present our own hybrid architecture where principles from fuzzy control theory and from neural networks are integrated into one system.

1 Introduction

Classical control theory is based on mathematical models that describe the behavior of the plant under consideration. The main idea of fuzzy control [11, 14], which has proved to be a very successful method [7], is to build a model of a human control expert who is capable of controlling the plant without thinking in a mathematical model. The control expert specifies his control actions in the form of linguistic rules. These control rules are translated into the framework of fuzzy set theory providing a calculus which can simulate the behavior of the control expert.

The specification of good linguistic rules depends on the knowledge of the control expert. But the translation of these rules into fuzzy set theory is not formalized and arbitrary choices concerning for example the shape of membership functions have to be made. The quality of the fuzzy controller can be drastically influenced by changing shapes of membership functions. Thus methods for tuning fuzzy controllers are necessary.

Neural networks offer the possibility to solve this tuning problem. The combination of neural networks and fuzzy controllers assembles the advantages of both approaches and avoids the drawbacks of them. Although a neural networks is able to learn from given data, the trained neural network is generally understood as a black box. Neither is it possible to extract structural knowledge from the trained neural network, nor can we integrate special information about the problem into the neural network in order to simplify the learning procedure. On

Accepted for presentation at FLAI'93, Linz, Austria, Jun. 28 – Jul. 2

the other hand, a fuzzy controller is designed to work with knowledge in the form of rules. But there exists no formal framework for the choice of parameters and the optimization of parameters has to be done by hand.

In Sections 2 and 3 we discuss approaches which use ordinary neural networks to optimize certain parameters of an ordinary fuzzy controller, and describe how neural networks can preprocess data for a fuzzy controller and extract fuzzy control rules from data. These approaches strictly separate the tasks of the neural network from the task of the fuzzy controller. Section 4 and 5 are devoted to hybrid architectures integrating the concepts of a fuzzy controller into a neural network or understanding a fuzzy controller as a neural network.

2 Tuning Fuzzy Control Parameters by Neural Networks

For the implementation of a fuzzy controller it is necessary to define membership functions representing the linguistic terms of the linguistic inference rules, which is more or less arbitrary. For example, consider the linguistic term *approximately zero*. Obviously, the corresponding fuzzy set should be a unimodal function reaching its maximum at the value zero. Neither the shape, which could be triangular or Gaussian, nor the range, i.e. the support of the membership function is uniquely determined by *approximately zero*. Generally, a control expert has some idea about the range of the membership function, but he would not be able to argue about small changes of his specified range.

Therefore, the tuning of membership functions becomes an import issue in fuzzy control. Since this tuning task can be viewed as an optimization problem neural networks offer a possibility to solve this problem.

A straightforward approach is to assume a certain shape for the membership functions which depends on different parameters that can be learned by a neural network. This idea was carried out in [16] where the membership functions are assumed to be symmetrical triangular functions depending on two parameters, one of them determining where the function reaches its maximum, the other giving the width of the support. Gaussian membership functions were used in [8].

Both approaches require a set of training data in the form of correct input–output tuples and a specification of the rules including a preliminary definition of the corresponding membership functions.

A method which can cope with arbitrary membership functions for the input variables is proposed in [6, 19, 20]. The training data have to be divided into r disjoint clusters $R_1, \ldots, R_r$. Each cluster R_i corresponds to a control rule $\mathcal{R}_i$. Elements of the clusters are tuples of input–output values of the form (x, y) where x can be a vector $x = (x_1, \ldots, x_n)$ of n input variables.

This means that the rules are not specified in terms of linguistic variables, but in the form of crisp input–output tuples.

A multilayer perceptron with n input units, some hidden layers, and r output units can be used to learn these clusters. The input data for this learning task are the input vectors of all clusters, i.e. the set $\{x \mid \exists i \; \exists y : (x, y) \in R_i\}$. The

target output $t_{u_i}(x)$ for input x at output unit u_i is defined as

$$t_{u_i}(x) = \begin{cases} 1 & \text{if there exists } y \text{ s.t. } (x, y) \in R_i \\ 0 & \text{otherwise.} \end{cases}$$

After the network has learned its weights, arbitrary values for x can be taken as inputs. Then the output at output unit u_i can be interpreted as the degree to which x matches the antecedent of rule $\mathcal{R}_i$, i.e. the function

$$x \mapsto o_{u_i}$$

is the membership function for the fuzzy set representing the linguistic term on the left–hand side of rule $\mathcal{R}_i$.

In case of a Mamdani type fuzzy controller the same technique can be applied to the output variable, resulting in a neural network which determines the fuzzy sets for the right–hand sides of the rules.

For a Sugeno type fuzzy controller, where each rule yields a crisp output value together with a number, specifying the matching degree for the antecedent of the rule, another technique can be applied. For each rule $\mathcal{R}_i$ a neural network is trained with the input–output tuples of the set R_i. Thus these r neural networks determine the crisp output values for the rules $\mathcal{R}_1, \ldots, \mathcal{R}_r$.

These neural networks can also be used to eliminate unnecessary input variables in the input vector x for the rules $\mathcal{R}_1, \ldots, \mathcal{R}_r$ by neglecting one input variable in one of the rules and comparing the control result with the one, when the variable is not neglected. If the performance of the controller is not influenced by neglecting input variable x_j in rule $\mathcal{R}_i$, x_j is unnecessary for $\mathcal{R}_i$ and can be left out.

3 Neural Networks for Extracting Fuzzy Rules

In the previous section we described, how neural networks could be used to optimize certain parameters of a fuzzy controller. We assumed that the control rules where already specified in linguistic form or as a crisp clustering of a set of correct input–output tuples. If we are given such a set of input–output tuples we can try to extract fuzzy (control) rules from this set. This can either be done by fuzzy clustering methods [3] or by using neural networks.

The input vectors of the input–output tuples can be taken as inputs for a Kohonen self–organizing map, which can be interpreted in terms of linguistic variables [17]. The main idea for this interpretation is to refrain from the winner–take–all principle after the weights for the self–organizing map are learned. Thus each output unit corresponds to an antecedent of a fuzzy control rule. Depending on how far output unit u_i is from being the 'winner' given input vector x, a matching degree $\mu_i(x)$ can be specified, yielding the degree to which x satisfies the antecedent of the corresponding rule.

Finally, in order to obtain a Sugeno type controller, to each rule (output unit) a crisp control output value has to be associated. Following the idea of the Sugeno type controller, we could choose the value

$$\frac{\sum_{(x,y)\in S} \sum_{i} \mu_i(x) \cdot y}{\sum_{(x,y)\in S} \mu_i(x)}$$

where S is the set of known input–output tuples for the controller.

Another way to obtain directly a fuzzy clustering is to apply the modified Kohonen network proposed in [4].

Kosko uses another approach to generate fuzzy–if–then rules from existing data [10]. Kosko shows that fuzzy sets can be viewed as points in a multidimensional unit hypercube. This makes it possible to use *fuzzy associative memories (FAM)* to represent fuzzy rules. Special adaptive clustering algorithms allow to learn these representations (AFAM).

4 Fuzzy Control Oriented Neural Networks

Another approach to combine fuzzy control and neural networks is to create a special architecture out of standard feed–forward nets that can be interpreted as a fuzzy controller. In [1] such a system, the ARIC architecture (*approximate reasoning based intelligent control*) is described by Berenji. It is a neural network model of a fuzzy controller and learns by updating its prediction of the physical system's behavior and fine tunes a predefined control knowledge base.

This kind of architecture allows to combine the advantages of neural networks and fuzzy controllers. The system is able to learn, and the knowledge used within the system has the form of fuzzy–if–then rules. By predefining these rules the system has not to learn from scratch, so it learns faster than a standard neural control system.

ARIC consists of two special feed–forward neural networks, the *action–state evaluation network (AEN)* and the *action selection network (ASN).* The ASN is a multilayer neural network representation of a fuzzy controller. In fact, it consists of two separated nets, where the first one is the fuzzy inference part and the second one is a neural network that calculates a *measure of confidence associated with the fuzzy inference value* using the weights of time t and the system state of time $t + 1$. A *stochastic modifier* combines the recommended control value of the fuzzy inference part and the so called "probability" value and determines the final output value of the ASN.

The hidden units of the fuzzy inference network represent the fuzzy rules, the input units the rule antecedents, and the output unit represents the control action, that is the defuzzified combination of the conclusions of all rules (output of hidden units). In the input layer the system state variables are fuzzified. The membership values of the antecedents of a rule are then multiplied by weights

attached to the connection of the input unit to the hidden unit. The minimum of those values is its final input. In each hidden unit a special monotonic membership function (see also Section 5) representing the conclusion of the rule is stored. Because of the monotonicity of this function the crisp output value belonging to the minimum membership value can be easily calculated by the inverse function. This value is multiplied with the weight of the connection from the hidden unit to the output unit. The output value is then calculated as a weighted average of all rule conclusions.

The AEN tries to predict the system behavior. It is a feed–forward neural network with one hidden layer, that receives the system state as its input and an error signal r from the physical system as additional information. The output $v[t,t']$ of the network is viewed as a *prediction of future reinforcement,* that depends of the weights of time t and the system state of time t', where t' may be t or $t+1$. Better states are characterized by higher reinforcements.

The weight changes are determined by a reinforcement procedure that uses the outputs of the ASN and the AEN. The ARIC architecture was applied to cart–pole balancing and it was shown that the system is able to solve this task [1].

We call the Berenji's system *a fuzzy control oriented neural network.* It is a system that consists of more or less standard neural networks, where one network employs a fuzzy aspect in using membership functions within its units and a fuzzy inference scheme. This approach shows the following advantages: The system makes use of an existing rule base, it has not to learn from scratch. One main part of the system is build according to those fuzzy–if–then rules, and so its structure can be easily interpreted. Finally, the system is able to learn to improve its control actions by using an error signal from the controlled physical system.

From our point of view ARIC has a few disadvantages that we hope to overcome with our approach presented in the next section. The first point of our criticism is that the changes in the weights of the fuzzy inference part of the ASN are difficult to interpret. A change in the weight connecting an input unit and a hidden unit means a change in the respective membership function of the input unit. It is possible that the same input unit is connected to several hidden units, meaning there are several rules which share common linguistic values in their antecedents. The problem is that each connection has a *different* weight. This results in the fact, that two rules sharing a common fuzzy set may have different interpretations of the same variable state because of different membership functions. However, to retain the semantics of the fuzzy rule base, we want each rule to have the same interpretation, and a change in the membership functions during a learning process should be the same for all rules.

Finally we criticize the use of the so called "probability" used for the "stochastic action modification" and the concept of the AEN. It is not obvious why the same weights are on the one hand useful to refine the fuzzy rule base and can be used on the other hand to calculate a "probability" for changing the output of the fuzzy inference network. The semantics of this value and the modification

are not clear. The evaluation of the controller performed by the AEN is a computation based on the system state and the system error that cannot be traced. This value is used as an error signal to change the weights, but has also unclear semantics.

ARIC is a working neural network architecture making use of fuzzy control aspects to its advantage but still has the disadvantage of neural systems that do not allow the semantical interpretation of certain steps in the information processing.

In [2] Berenji and Khedkar present an enhancement of this approach called GARIC (generalized ARIC). There they have overcome the disadvantage of different interpretation of linguistic values by refraining from weighted connections in the ASN. The fuzzy sets are modelled as nodes in the network, and learning is achieved by changing parameters within these nodes that determine the shape of the membership functions. By defining a differentiable *softmin* function and a special defuzzification scheme (*localized mean of maximum)* they can use a different learning algorithm that allows to use any form of membership functions in the conclusions. The stochastic modification scheme is still in use for this new approach.

5 Neural Network Oriented Fuzzy Control

In this section we present our approach to a combination of neural networks and fuzzy controllers (see also [5]). Our main concern is to keep the structure of the fuzzy controller that is determined by the fuzzy rules. We think of these rules as a piece of structural knowledge that gives us a roughly correct representation of the system to be controlled. We consider it a natural approach to define a fuzzy error for our system, which, according to its structure, we may call neural fuzzy controller. This enables us to define a reinforcement learning algorithm that is described at length in [12]. Training a fuzzy controller with such a learning procedure allows us to keep track of the changes and to interpret the modified rules.

We consider a dynamical system S that can be controlled by one variable C and whose state can be described by n variables $X_1, \ldots, X_n$, i.e. we have a multiple input – single output system. The variables are modelled by membership functions, and the control action that drives the system S to a desired state is described by the well-known concept of fuzzy if-then rules [21].

To overcome the problem of defuzzification, and to determine the individual part that each rule contributes to the final output value, we use Tsukamoto's monotonic membership functions, where the defuzzification is reduced to an application of the inverse function [1, 11]. Such a membership function μ is characterized by two points a, b with $\mu(a) = 0$ and $\mu(b) = 1$, and it is defined as

$$\mu(x) = \begin{cases} \frac{-x+a}{a-b} & \text{if } (x \in [a,b] \wedge a \leq b) \vee (x \in [b,a] \wedge a > b) \\ 0 & \text{otherwise} \end{cases}$$

The defuzzification is carried out by

$$x = \mu^{-1}(y) = -y(a-b) + a$$

with $y \in [0,1]$.

For our purposes we only need to restrict ourselves to monotonic membership functions to represent the linguistic values of the output variable. For the input variables the usual triangular, trapezoidal etc. membership functions can be chosen.

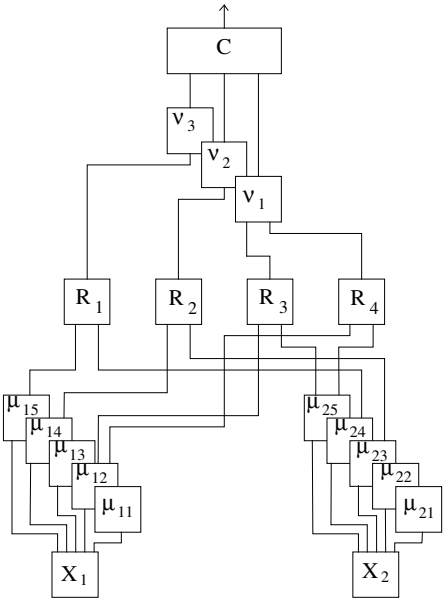

Fig. 1. The structure of the neural fuzzy controller

An example for the structure of our neural fuzzy controller is depicted in figure 1. The nodes X_1 and X_2 represent the input variables, and deliver their crisp values to their μ-modules which contain the respective membership functions. The μ-modules are connected to the R-modules which represent the fuzzy if-then rules. Each μ-module gives to all of its connected R-modules, the membership value $\mu_{ij}(x_i)$ of its input variable X_i. The R-modules use a t-norm (min-operation in this case) to calculate the conjunction of their inputs and pass this value forward to one of the ν-modules, which contain the membership functions representing the output variable. By passing through the ν-modules these values

are changed to the conclusion of the respective rule. This means the implication (min-implication in this case) is carried out to obtain the value of the conclusion, and because monotonic membership functions are used, the ν-modules pass pairs $(r_l, \nu_k^{-1}(r_l))$ to the C-module, where the final output value is calculated by

$$c = \frac{\sum_{i=1}^{n} r_i \nu_{R_i}^{-1}(r_i)}{\sum_{i=1}^{n} r_i},$$

where n is the number of rules, and r_i is the degree to which rule R_i has fired.

As one can easily see, the system in figure 1 resembles a feedforward neural network. The X-, R-, and C-modules can be viewed as the neurons and the μ- and ν-units as the adaptable weights of the network. The fact that one μ-module can be connected to more than one R-module is equivalent to connections in a neural network that share a common weight. This is very important, because we want each linguistic value to be represented by only one membership function that is valid for all rules. By this restriction we retain the structural knowledge that we put into the system by defining the rules. In other neural fuzzy systems this fact is not recognized [1, 9] and it is possible that one linguistic value is represented by different membership functions.

Our goal is to tune the membership functions of the controller by a learning algorithm according to a measure that adequately describes the state of the plant under consideration. The desired optimal state of the plant can be described by a vector of state variable values. But usually we are content with the current state if the variables have roughly taken these values. And so it is natural to define the goodness of the current state by a membership function from which we can derive a fuzzy error [12] that characterizes the performance of our neural fuzzy controller.

Consider a system with n state variables $X_1, \ldots, X_n$. We define the fuzzy-goodness G_1 as

$$G_1 = \min\left\{\mu_{\text{opt}}^{(1)}(x_1), \ldots, \mu_{\text{opt}}^{(n)}(x_n)\right\},$$

where the membership functions $\mu_{\text{opt}}^{(i)}$ have to be defined according to the requirements of the plant under consideration.

In addition of a near optimal state we also consider states as good, where the incorrect values of the state variables compensate each other in a way, that the plant is driven towards its optimal state. We define the fuzzy-goodness G_2 as

$$G_2 = \min\left\{\mu_{\text{comp}}^{(1)}(x_1, \ldots, x_n), \ldots, \mu_{\text{comp}}^{(k)}(x_1, \ldots, x_n)\right\}$$

where the membership functions $\mu_{\text{comp}}^{(j)}$ again have to be defined according to the requirements of the plant. There may be more than one $\mu_{\text{comp}}^{(j)}$ and they may depend on two or more of the state variables.

The overall fuzzy-goodness is defined as

$$G = g(G_1, G_2),$$

where the operation g has to be specified according to the actual application [12]. The fuzzy-error that is made by our neural fuzzy controller is defined as

$$E = 1 - G.$$

The learning algorithm depending on this fuzzy error works for each fuzzy rule in parallel. Each rule R_i knows the value r_i of the conjunction of its antecedents and the (crisp) value c_i of its conclusion. After the control action has been determined by the controller and the new state of the plant is known, we propagate the fuzzy error E and the current values of the state variables to each R-module. If the rule has contributed to the control output, i.e. $r_i \neq 0$, it has to evaluate its own conclusion. According to the current state of the plant the rule can decide, whether its conclusion would drive the system to a better or to a worse state. For the first case the rule has to be made more sensitive and has to produce a conclusion that increases the current control action, i.e. makes it more positive or negative respectively. For the second case the opposite action has to be taken.

Consider that we are using Tsukamoto's monotonic membership functions. Each membership function can be characterized by a pair (a,b) such that $\mu(a) = 0$ and $\mu(b) = 1$ hold. A rule is made more sensitive by increasing the difference between these two values in each of its antecedents. That is done by keeping the value of b and changing a. That means the membership functions are keeping their positions determined by their b-values, and their ranges determined by $|a-b|$ are made wider. To make a rule less sensitive the ranges have to be made smaller. In addition to the changes in its antecedents, each firing rule has to change the membership function of its conclusion. If a rule has produced a good control value, this value is made better by decreasing the difference $|a-b|$, and a bad control value is made less worse by increasing $|a-b|$.

The rules change the membership functions by propagating their own rule-error

$$e_{R_i} = \begin{cases} -r_i \cdot E & \text{if } \operatorname{sgn}(c_i) = \operatorname{sgn}(c_{\text{opt}}) \\ r_i \cdot E & \text{if } \operatorname{sgn}(c_i) \neq \operatorname{sgn}(c_{\text{opt}}) \end{cases}$$

to the connected μ- and ν-modules. The changes in the membership functions of the conclusions (ν-modules) are calculated according to

$$a_k^{\text{new}} = \begin{cases} a_k - \sigma \cdot e_{R_i} \cdot |a_k - b_k| & \text{if } (a_k < b_k) \\ a_k + \sigma \cdot e_{R_i} \cdot |a_k - b_k| & \text{otherwise,} \end{cases}$$

where σ is a learning factor and R-module R_i is connected through ν_k to the C-module. If ν_k is shared, it is changed by as much R-modules as are connected to the C-module through this membership function. For the membership functions of the antecedents (μ-modules) the following calculation is carried out:

$$a_{jk_j}^{\text{new}} = \begin{cases} a_{jk_j} + \sigma \cdot e_{R_i} \cdot |a_{jk_j} - b_{jk_j}| & \text{if } (a_{jk_j} < b_{jk_j}) \\ a_{jk_j} - \sigma \cdot e_{R_i} \cdot |a_{jk_j} - b_{jk_j}| & \text{otherwise,} \end{cases}$$

where the X-module X_j is connected to the R-module R_i through the membership function μ_{jk_j}, with $k_j \in \{1, \ldots, s_j\}$, and s_j is the number of linguistic values of X_j. If μ_{jk_j} is shared, it is changed by as much R-modules as X_j is connected to through this μ-module.

Our fuzzy error propagation learning algorithm is a reinforcement learning algorithm where each R-module learns according to the extent it has fired under the given circumstances and according to an error describing the performance of the control system. The neural fuzzy controller has not to learn from scratch, but knowledge in the form of fuzzy if-then rules is coded into the system. The learning procedure does not change this structural knowledge. It tunes the membership functions in an obvious way, and the semantics of the rules is not blurred by any semantically suspicious factors or weights attached to the rules.

The algorithm finds the most appropriate membership functions for the current control situation, while the meaning of each rule is not changed. If the learning is successful, this means that the rule base is suitable for the control task, and the changes in the fuzzy sets are due to an unsuitable modeling of the linguistic values by the a priori membership functions. It is not possible that two rules use different fuzzy sets describing the same linguistic value. That means we can interpret the new membership functions to give us the best fuzzy sets for the current task.

The neural fuzzy controller has been tested in a simulation using a simplified version of the inverted pendulum described by the differential equation

$$(m + \sin^2\theta)\ddot{\theta} + \frac{1}{2}\dot{\theta}^2 \sin(2\theta) - (m+1)\sin\theta = -F\cos\theta.$$

The controller was already able to balance the pendulum using the initial rule base and membership functions without learning. But it was not able to maintain an angle of roughly $\theta = 0$, and it was not able to cope with extreme starting positions (e.g. $\theta = 20°$ and $\dot{\theta} = 60°$ per second). After the learning process has been switched on the performance of the controller increased. It was able to maintain an angle of approximately zero and to cope with starting positions of e.g. $\theta = 30°$ and $\dot{\theta} = 60°$ per second [12].

An extension of this model is presented in [15]. There the system starts to learn from scratch, i.e. it determines the fuzzy rule base while it learns. The idea is to initialize the network with all rules that can be constructed out of all combinations of input and output membership functions. During the learning process all hidden nodes (rule nodes) that are not used or produce counterproductive results are removed from the network. We call this system fuzzy neural network because its initial state resembles more a neural network than a fuzzy controller. But after the learning process the system can of course still be interpreted in terms of a fuzzy controller.

The learning procedure is divided in three phases:

1. During phase I all rule nodes that have produced a counterproductive result, i.e. a negative value where a positive value is required and vice versa, are deleted instantly. Furthermore each rule node maintains a counter that is

decremented each time the rule node does not fire, i.e. $r_j = 0$. In the other case, i.e. $r_j > 0$, the counter is reset to a maximum value.

2. At the beginning of phase II there are no rule nodes left that have identical antecedents and consequences that produce output values of different directions. To obtain a sound rule base, from each group of rules with identical antecedents only one rule node must remain. During this phase the counters are evaluated and each time a counter reaches 0 the respective rule node is deleted. Furthermore each rule node now maintains an individual error value that accumulates a fuzzy transition error

$$E_t = 1 - min\left\{\tau_i(\Delta x_i) | i \in \{1, \ldots, n\}\right\},$$

where Δx_i is the change in variable X_i, and τ_i is a membership function giving a fuzzy representation of the desired change in the respective variable. If the rule produced a counterproductive result, E_t is added unscaled, and if not, E_t is weighted by the normalized difference between the rule output value and the control output value of the whole network. At the end of this phase from each group of rule nodes with identical antecedents, only the node with the least error value remains, all other rule nodes are deleted. This leaves the network with a minimal set of rules needed for the control task.

3. During phase III the performance of the fuzzy neural network is enhanced by tuning the membership functions as it is described above.

At the end of the learning process the remaining rule nodes identify the fuzzy if-then-rules that are necessary to control the dynamical system under consideration, and the fuzzy weights represent the membership functions that suitably describe the linguistic values of the input and output variables for this situation. A simulation of this model applied to the inverted pendulum gave first promising results. The system reduced the number of rules from 512 to a number between 20 and 40 depending on the internal parameters. In most of the simulation runs the resulting network was able to balance the pendulum. To present final results still more tests have to be carried out.

6 Conclusions

We have discussed four possible combinations of neural networks and fuzzy controllers. The first two approaches use stand–alone neural networks to tune membership functions or to create fuzzy–if–then rules. The remaining concepts both try to build hybrid systems that benefit from the advantages of both worlds. Berenji's ARIC architecture is a special neural network system that still has some semantical problems. We tried to overcome these problems with our neural fuzzy controller that uses a learning algorithm based on a fuzzy error measure. The structure of the controller resembles a neural network and the fuzzy error propagation is a reinforcement learning procedure as used for certain kinds of connectionistic systems. Simulations of the controller used to balance an inverted

pendulum have shown that the learning procedure improves the behavior of the fuzzy controller and is able to handle extreme situations where the non-learning controller fails [12, 13]. The learning algorithm starts from a predefined rule base, and it does not change the structural knowledge encoded in these rules. It leaves the semantics of each rule intended by the user unchanged, but removes the errors caused by an inaccurate modelling by changing the fuzzy sets. The results of the learning procedure can be interpreted easily.

References

1. H.R. Berenji: A Reinforcement Learning-Based Architecture for Fuzzy Logic Control. Int. J. Approximate Reasoning **6** (1992), 267-292.
2. H.R. Berenji, P. Khedkar: Learning and Tuning Fuzzy Logic Controllers Through Reinforcements. IEEE Trans. Neural Networks **3** (1992), 724-740.
3. J.C. Bezdek, S.K. Pal (eds.): Fuzzy Models for Pattern Recognition. IEEE Press, New York (1992).
4. J.C. Bezdek, E.C. Tsao, N.K. Pal: Fuzzy Kohonen Clustering Networks. Proc. IEEE Int. Conf. on Fuzzy Systems 1992, San Diego (1992), 1035-1043.
5. P. Eklund, F. Klawonn, D. Nauck: Distributing Errors in Neural Fuzzy Control. Proc. 2nd Int. Conf. on Fuzzy Logic and Neural Networks, IIZUKA'92, Iizuka (1992), 1139-1142.
6. I. Hayashi, H. Nomura, H. Yamasaki, N. Wakami: Construction of Fuzzy Inference Rules by NFD and NDFL. Int. J. Approximate Reasoning **6** (1992), 241-266.
7. K. Hirota: Survey of Industrial Applications of Fuzzy Control in Japan. Proc. IJCAI-91 Workshop on Fuzzy Control, Sydney (1992), 18-20.
8. H. Ichihashi: Iterative Fuzzy Modelling and a Hierarchical Network. In: R. Lowen, M. Roubens (eds.): Proc. 4th IFSA Congress, Engineering, Brussels (1991), 49-52.
9. J.-S.R. Jang: Fuzzy Modelling Using Generalized Neural Networks and Kalman Filter Algorithm. Proc. 9th Nat. Conf. on Artificial Intelligence, AAAI-91, MIT-Press, Menlo Park (1991), 762-767.
10. B. Kosko: Neural Networks and Fuzzy Systems. Prentice-Hall, Englewood Cliffs (1992).
11. C.C. Lee: Fuzzy Logic in Control Systems: Fuzzy Logic Controller. IEEE Trans. Syst. Man Cybern. **20** (1990), Part I: 404-418, Part II: 419-435.
12. D. Nauck, R. Kruse: A Neural Fuzzy Controller Learning by Fuzzy Error Propagation. Proc. NAFIPS'92, Puerto Vallarta, (1992), 388-397.
13. D. Nauck, R. Kruse: Interpreting Changes in the Fuzzy Sets of a Self-Adaptive Neural Fuzzy Controller. Proc. 2nd Int. Workshop in Industrial Fuzzy Control and Intelligent Systems IFIS'92, College Station (1992) 146-152.
14. D. Nauck, F. Klawonn, R. Kruse: Fuzzy Sets, Fuzzy Controllers, and Neural Networks. Wissenschaftliche Zeitschrift der Humboldt-Universität zu Berlin, R. Medizin **41** (4) (1992), 99-120.
15. D. Nauck, R. Kruse: A Fuzzy Neural Network Learning Fuzzy Control Rules and Membership Functions by Fuzzy Error Backpropagation. Proc. IEEE Int. Conf. on Neural Networks ICNN'93, San Francisco (1993).
16. H. Nomura, I. Hayashi, N. Wakami: A Learning Method of Fuzzy Inference Rules by Descent Method. Proc. IEEE Int. Conf. on Fuzzy Systems 1992, San Diego (1992), 203-210.

17. W. Pedrycz, H.C. Card: Linguistic Interpretation of Self-Organizing Maps. Proc. IEEE Int. Conf. on Fuzzy Systems 1992, San Diego (1992), 371-378.
18. S. Shao: Fuzzy Self-Organizing Controller and its Application for Dynamic Processes. Fuzzy Sets and Systems **26** (1988), 151-164.
19. H. Takagi, I. Hayashi: NN-Driven Fuzzy Reasoning. Int. J. Approximate Reasoning **5** (1991), 191-212.
20. H. Takagi, N. Suzuki, T. Koda, Y. Kojima: Neural Networks Designed on Approximate Reasoning Architecture and their Applications. IEEE Trans. Neural Networks **3** (1992), 752-760.
21. L.A. Zadeh: Outline of a New Approach to the Analysis of Complex Systems and Decision Processes. IEEE Trans. Syst. Man Cybern. **3** (1973), 28-44.