A Fuzzy Neural Network Learning Fuzzy Control Rules and Membership Functions by Fuzzy Error Backpropagation

Detlef Nauck Rudolf Kruse
Department of Computer Science
Technical University of Braunschweig
W-3300 Braunschweig, Germany

***Abstract*— In this paper we present a new kind of neural network architecture designed for control tasks, which we call fuzzy neural network. The structure of the network can be interpreted in terms of a fuzzy controller. It has a three-layered architecture and uses fuzzy sets as its weights. The fuzzy error backpropagation algorithm, a special learning algorithm inspired by the standard BP-procedure for multilayer neural networks, is able to learn the fuzzy sets. The extended version that is presented here is also able to learn fuzzy-if-then rules by reducing the number of nodes in the hidden layer of the network. The network does not learn from examples, but by evaluating a special fuzzy error measure.**

I. Introduction

Neural Networks and Fuzzy Controllers are both capable of controlling nonlinear dynamical systems. The disadvantage of neural control is that it is not obvious how the network solves the respective control task. It is not possible in general to retrieve any kind of structural knowledge from the network that could be formulated e.g. in some kind of rules, or to use prior knowledge to reduce the learning time. The network has to learn from scratch, and might have to do so again if substantial parameters of the dynamical system change for some reason.

The use of a fuzzy controller on the other hand allows to interpret the control behavior due to the explicit linguistic rules the controller consists of. The design problems of a fuzzy controller are the choice of appropriate fuzzy if-then-rules, and membership functions, and the tuning of both in order to improve the performance of the fuzzy controller. The disadvantage of this method is the lack of suitable learning algorithms retaining the semantics of the controller.

We propose to overcome these disadvantages by using a special feed-forward neural network architecture with fuzzy sets as its weights. The weights are membership functions and represent the linguistic values of the input variables and the output variables, respectively. Due to its structure the network can be interpreted as a fuzzy controller, where the nodes of the hidden layer represent fuzzy if-then-rules. The input layer consists of nodes for the variables describing the state of the dynamical system, and the output layer contains the node(s) representing the control action to drive the system towards a desired state. A learning algorithm utilizing a fuzzy error measure, valuating the output of the fuzzy neural network, is able to learn fuzzy rules by deleting hidden nodes, and to adapt the membership functions. We call this learning procedure fuzzy error backpropagation (FEBP).

We refrained from just inserting neural networks in fuzzy controllers to tune some weights scaling the rules or the fuzzy sets [3], or from using neural networks to identify fuzzy rules to build a fuzzy controller [9]. We use instead a new architecture which we call fuzzy neural network (FNN) that is able to learn both fuzzy if-then-rules and fuzzy sets by changing its structure and its weights. The network converges to a state that can be easily interpreted and has clear semantics. Our model is able to learn, to use prior knowledge, and has no black box behavior. Its structure can be interpreted in terms of membership functions and fuzzy-if-then rules. The learning procedure is an extension to our fuzzy error propagation algorithm for neural fuzzy controllers [6, 7]. A learning procedure that uses examples is considered in [2].

II. The Fuzzy Neural Network

We consider a dynamical system S that can be controlled by one variable C and whose state can be described by n variables $X_1, \ldots, X_n$. For each variable we consider measurements in a subinterval $H = [h_1, h_2]$ of the real line. The imprecision is modelled by mappings $\mu : [h_1, h_2] \to [0, 1]$ in the sense of membership functions with the obvious interpretation as representations of linguistic values.

In: Proc. IEEE Int. Conf. Neural Networks ICNN'93, San Francisco, Mar. 28 - Apr. 1 (1993), 1022–1027

The control action that drives the system S to a desired state is described by the well-known concept of fuzzy if-then rules [10] where a conjunction of input variables associated with their respective linguistic values determines a linguistic value associated with the output variable. All rules are evaluated in parallel, and their outputs are combined to a fuzzy set which has to be defuzzified to receive the crisp output value.

As the T-norm operator for the conjunction of the input values usually the min-operator is used, and as the S-norm for aggregating the output values of the rules the max-operator is used, as it is done by the well known Zadeh-Mamdani procedure [10, 5].

For the evaluation of fuzzy rules the defuzzification-operation constitutes a problem that cannot be neglected. It is not obvious which crisp value is best suited to characterize the output fuzzy set of the rule system. In most of the fuzzy control environments the center-of-gravity method is used [4]. Using this method, it is difficult to determine the individual part that each rule contributes to the final output value.

To overcome this problem we use Tsukamoto's monotonic membership functions, where the defuzzification is reduced to an application of the inverse function [1]. 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

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

The crisp value x belonging to the membership value y can be easily calculated by

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

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

In Fig. 1 a simple fuzzy neural network is shown that incorporates the two fuzzy-if-then rules

R_1: **IF** X_1 is PL **AND** X_2 is PL **THEN** C is PL,
R_2: **IF** X_1 is PL **AND** X_2 is PM **THEN** C is PM,

where PL and PM represent the usual linguistic expressions *positive large* and *positive medium*. The circles represent the nodes, and the squares represent the fuzzy weights.

The network consists of three layers. The first layer, the input layer, contains one node for each input variable. The states of these nodes reflect the crisp values of the variables. The intermediate layer is called the *rule layer.* It contains one node for each fuzzy-if-then rule. The state of each *rule node* represents the membership value obtained by the conjunction of the rule antecedents. The crisp input values are sent over the respective connections to the rule nodes. Each connection has a fuzzy weight attached to it that is combined with the input value. In our case these weights are membership functions μ_{ik_i} representing the linguistic values of the input variables. The combination of the weight μ_{ik_i} and the crisp input value x_i simply results in the membership value $\mu_{ik_i}(x_i)$. If an input variable takes part with the same linguistic value in more than one rule antecedent, it is connected to the respective rule nodes by connections that share a common weight (see Table I and Fig. 1, weight μ_{11}). The sharing of the fuzzy weights is an important aspect of the architecture. We want to interpret the behavior of the network, and so we need identical membership functions for identical linguistic values.

The rule nodes R_j collect all incoming membership values and use a T-norm (e.g. min-operation) to calculate the conjunction of their antecedents. This is done by all rule nodes in parallel, and then they pass their states on to the output layer that contains one or more nodes describing the control action for the next time step. The weights attached to the connections between the intermediate and the output layer are membership functions ν_k representing the linguistic values of the rule consequents. The calculations that are carried out on the connections represent a fuzzy implication (e.g. Mamdani's minimum operation

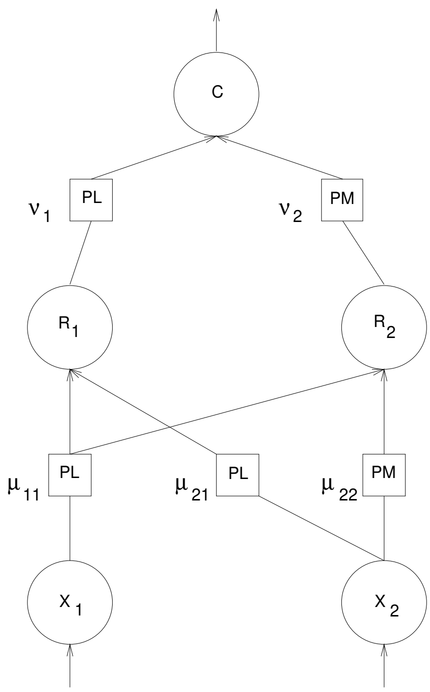
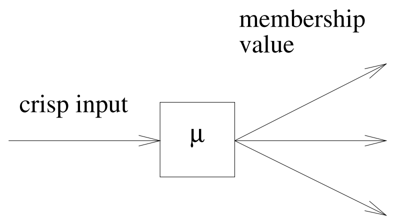
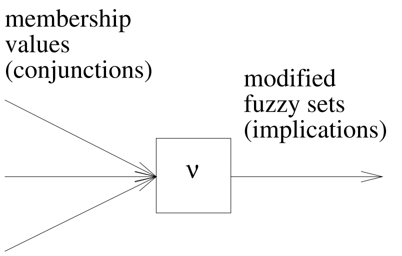


Fig. 1: A fuzzy neural network with two fuzzy rules

TABLE I:
Types of Connections in a Fuzzy Neural Network

<table>
<tr>
<td>
</td>
<td>A connection from an input variable to three different rule nodes, which use the same linguistic value of this input variable in their antecedents.</td>
</tr>
<tr>
<td>
</td>
<td>A connection from three rule nodes with the same linguistic value for their consequences to an output node. The output node receives three different signals from this connection.</td>
</tr>
</table>

as fuzzy implication) resulting in fuzzy sets $\nu^*_{R_j}$ describing the output of each rule R_j. Nodes representing rules with an identical linguistic output value share a common weight on their connection to the respective output node. In this case more than one value travels on the connection to the output node (see Table I).

Each output node collects the incoming fuzzy sets, aggregates them, and determines a crisp output value with the help of an appropriate defuzzification procedure D.

The learning algorithm for the fuzzy neural network needs to know the part each rule contributes to the final output value, and therefore the use of monotonic membership functions in the consequence parts of the fuzzy rules is required. This results in Tsukamoto's method for fuzzy reasoning [4] where the aggregation and the defuzzification are carried out in one step. The crisp output value c (activation of the output node) is calculated by

$$c = \frac{\sum_{j=1}^{m} r_j \nu^{-1}_{R_j}(r_j)}{\sum_{j=1}^{m} r_i},$$

where m is the number of rules, and r_j is the result inferred from rule R_j. This is a simplified reasoning method based on Mamdani's minimum operation rule that is usually used in fuzzy controllers [4].

If the fuzzy rules are known, the learning algorithm has just to tune the membership functions. This situation is described in [6]. In the case that the rules are unknown, the learning algorithm has to be extended, so they can be learned, too.

To build a fuzzy neural network we have to identify the input and output variables. Then for each variable a number of initial non-optimal fuzzy sets have to be chosen describing such linguistic terms as *positive large, negative small,* etc. If we assume that the fuzzy if-then-rules that are necessary to control the dynamical system are not known, we have to create a network, that represents every possible rule that can be created from the number of fuzzy sets attributed to the variables. If we have a MISO (multiple input single output) system with two input variables and five linguistic variables for each variable we have to create a fuzzy neural network with 2 input nodes, 1 output node, and $5^3 = 125$ intermediate rule nodes. An example of our fuzzy neural network where three fuzzy sets are attributed to each variable is depicted in Fig. 2. During the learning procedure the network has to remove those rule nodes that are not needed or that are counterproductive.

III. Fuzzy Error Backpropagation and Rule Learning

Our goal is to tune the membership functions of the fuzzy neural network by a learning algorithm. Because it is usually not possible to calculate the optimal control action by other means parallel to the network so we can derive the error directly, we have to obtain a measure that adequately describes the state of the plant under consideration.

The optimal state of the plant can be described by a vector of state variable values. That means, the plant has reached the desired state if all of its state variables have reached their value defined by this vector. 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 that characterizes the performance of the fuzzy neural network.

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

$$G_1 = \min\left\{\mu_{X_1}^{optimal}(x_1), \ldots, \mu_{X_n}^{optimal}(x_n)\right\},$$

where the membership functions $\mu_{X_i}^{optimal}$ 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

$$\begin{aligned} G_2 \quad = \quad \min \quad & \{\mu^{compensate_1}(x_1, \ldots, x_n), \ldots, \\ & \mu^{compensate_k}(x_1, \ldots, x_n)\} \end{aligned}$$

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

The overall fuzzy-goodness is defined as

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

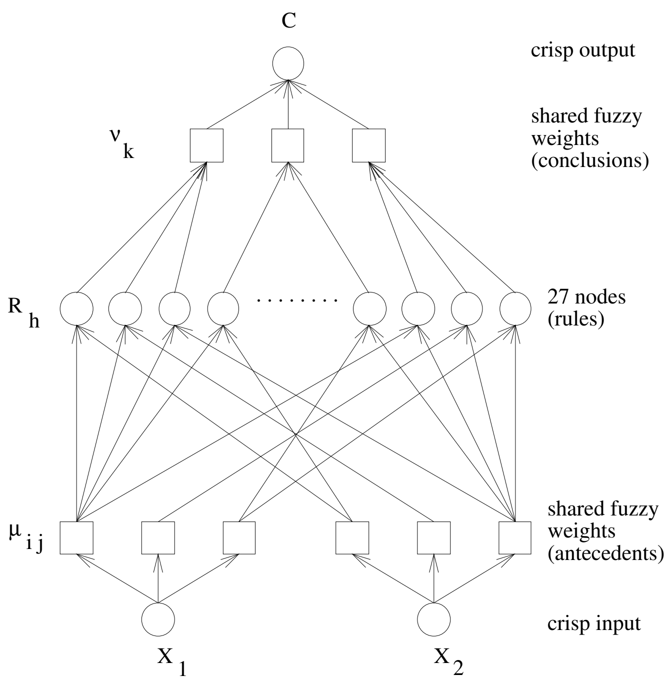


Fig. 2: The initial state of a fuzzy neural network

where the operation g has to be specified according to the actual application. In some cases a min-operation may be appropriate, and in other cases it may be more adequate to choose just one of the two goodness measures, perhaps depending on the sign of the current values of the state variables, e.g. we may want to use G_1 if all variables are positive or negative and G_2 if they are both positive and negative.

The fuzzy-error of the fuzzy neural network is defined as

$$E = 1 - G,$$

and it is needed to tune the membership functions.

In addition to this error measure we need an additional measure that helps to determine the rule nodes that have to be deleted from the network. We have to define when we conceive a transition between two states as desirable. By this we can derive a *fuzzy transition error* E_t to punish those rule nodes that contribute to overshooting or those nodes that apply a force that is too small to drive the system to a desired state.

The fuzzy transition error is defined as

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

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 for the respective variable.

We are now able to define our learning algorithm. For each rule node R_j the value r_j of the conjunction of its antecedent and the value c_{R_j} of its consequence is known. Because we are using monotonic membership functions, c_{R_j} is already crisp. After the control action has been determined in the output node, applied to the plant, and its new state is known, we propagate the fuzzy-error E, the fuzzy transition error E_t and the current values of the state variables back through the network. The output node can determine for each rule node R_j that has contributed to the control output, i.e. $r_j \neq 0$, whether its conclusion would drive the system to a better or to a worse state. For the first case the rule node 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.

The learning procedure is divided in three phases:

Phase I: 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.

Phase II: 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 the fuzzy transition error. 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 to control the plant under consideration.

Phase III: During phase III the performance of the fuzzy neural network is enhanced by tuning the membership functions. 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 changing a such that their ranges determined by $|b-a|$ are made wider. To make a rule less sensitive the ranges have to be made smaller. In addition to the changes in its antecedents, for firing rule the membership function of its conclusion have to be changed. If a rule has produced a good control value, this value is made better by decreasing the difference $|b-a|$, and a bad control value is made less worse by increasing $|b-a|$.

The output node C has to know the direction of the optimal control value c_{opt}. The value itself is unknown, of course, but from the observation of the current state it can be determined if e.g. a positive or a negative force had to be applied, i.e. $\text{sgn}(c_{\text{opt}})$ is known. The output node calculates an individual rule error e_{R_j} for each rule node R_j according to

$$e_{R_j} = \begin{cases} -r_j \cdot E & \text{if } \text{sgn}(c_{R_j}) = \text{sgn}(c_{\text{opt}}), \\ r_j \cdot E & \text{if } \text{sgn}(c_{R_j}) \neq \text{sgn}(c_{\text{opt}}). \end{cases}$$

The changes in the membership functions ν_k of the connections from the intermediate layer to the output layer are determined as follows:

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

where σ is a learning factor and rule node R_j is connected through ν_k to C. If a weight ν_k is shared, it is changed by the output node as often as rule nodes are connected to it through this weight.

The rule errors are now propagated back to the intermediate layer, where the rule nodes change the membership functions of their antecedents:

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

where input node X_i is connected to R_j through μ_{ik_i}, $k_i \in \{1, \dots, s_i\}$, s_i is the number of linguistic values of X_i. If a weight μ_{ik_i} is shared, it is changed by as much rule nodes as X_i is connected to through this weight.

At the end of the learning process we can interpret the structure of the network. The remaining rule nodes identify the fuzzy if-then-rules that are necessary to control the dynamical system under consideration. The fuzzy weights represent the membership functions that suitably describe the linguistic values of the input and output variables.

IV. Simulation Results

The fuzzy neural network has been simulated and applied to the control of an inverted pendulum. Although it is too early to present final results of the performance, and more tests have to be run, first experiments gave promising results. For the inverted pendulum a simplified version was used that is 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 movement of the rod is simulated by a Runge-Kutta procedure with a timestepwidth of 0.1.

There are eight linguistic values attributed to each of the three variables. This are the common values PL, PM, PS, PZ, NZ, NS, NM, NL. Because we use monotonic membership functions that are not symmetric, we model the value Zero as Positive Zero and Negative Zero. This

TABLE II:

The Rule Base After the Learning Process

	NL	NM	NS	NZ	PZ	PS	PM	PL
NL	NZ	NZ	NZ	NZ				
NM	NZ	NZ	NZ	NZ				
NS	NZ	NZ	NZ	NZ				
NZ	NZ	NM	NM	NS				
PZ					PS	PM	PM	PZ
PS					PM	PZ	PZ	PZ
PM					PZ	PZ	PZ	PZ
PL					PZ	PZ	PZ	PZ

TABLE III:

The Membership Functions Before and After the Learning Process

	angle			*angle velocity*			*force*		
	b	d	d'	b	d	d'	b	d	d'
NL	-45	45	43.9	-4.0	4.0	3.6	-30	30	15.0
NM	-30	30	30.0	-2.5	2.5	2.2	-20	20	14.9
NS	-15	15	14.9	-1.0	1.0	1.0	-10	10	8.0
NZ	0	-45	-14.9	0.0	-4.0	-1.0	0	-30	-8.0
PZ	0	45	14.5	0.0	4.0	0.9	0	30	8.0
PS	15	-15	-14.9	1.0	-1.0	-0.9	10	-10	-8.4
PM	30	-30	-29.9	2.5	-2.5	-2.1	20	-20	-15.0
PL	45	-45	-43.9	4.0	-4.0	-3.6	30	-30	-15.0

leads to an initial state of the network with $8 \cdot 8 \cdot 8 = 512$ possible rules. Several tests have been run under these conditions. One result is shown in the Tables II and III, where the final fuzzy rule base (remaining hidden nodes) and the initial and final state of the membership functions can be found. In Table III d denotes the difference $b - a$. As we stated above a monotonic membership function is adapted by changing its parameter a to a' during the learning process.

The system was able to balance the pendulum with this configuration. In our experiments the learning process succeeded in about 90% of all cases resulting in different rule bases and fuzzy sets depending on internal parameters describing the error measures.

V. Conclusions

The presented fuzzy neural network is able to learn fuzzy-if-then rules by deleting nodes of its intermediate layer, and it learns membership functions that are used as its weights by fuzzy error backpropagation. The structure of the network can be easily interpreted in terms of a fuzzy controller.

The network has not to learn from scratch as it is presented here, but knowledge in the form of fuzzy if-then rules can be 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 are not blurred by any semantically suspicious factors or weights attached to the rules [6, 7]. If some parameters of the controlled plant change, the fuzzy neural network can be easily changed, e.g. by adding additional rule nodes, and by restarting the the learning process to change the membership functions, if necessary.

Berenji has presented a similar approach [1] combining two neural networks into a system that behaves like a fuzzy controller. A comparison between Berenji's and our approach is presented in [8].

References

[1] Hamid R. Berenji: A reinforcement learning-based architecture for fuzzy logic control. Int. J. Approx. Reas., vol. 6, no. 2, 267-292 (1992)

[2] P. Eklund, F. Klawonn, D. Nauck: Distributing errors in neural fuzzy control. Proc. IIZUKA'92, 1139-1142, (1992)

[3] B. Kosko: Neural Networks and Fuzzy Systems. Prentice–Hall, Englewood Cliffs (1992)

[4] C.C. Lee: Fuzzy logic control systems: Fuzzy logic controller. IEEE Trans. Syst. Man Cybern., vol. 20, no. 2, 404-418 (Part I), 419-435 (Part II), (1990)

[5] E.H. Mamdani: Applications of fuzzy algorithms for a simple dynamic plant. Proc. IEE vol. 121, 1585-1588 (1974)

[6] D. Nauck, R. Kruse: A neural fuzzy controller learning by fuzzy error propagation. Proc. NAFIPS'92, 388-397 (1992)

[7] D. Nauck, R. Kruse: Interpreting the changes in the fuzzy sets of a self-adaptive neural fuzzy controller. Proc. IFIS'92 (1992)

[8] D. Nauck, F. Klawonn, R. Kruse: Combining neural networks and fuzzy controllers. Submitted for publication (1993)

[9] H. Takagi, I. Hayashi: NN–driven fuzzy reasoning. Int. J. Approx. Reas. vol. 5 no. 3, 191-212 (1991)

[10] L.A. Zadeh: Outline of a new approach to the analysis of complex systems and decision processes. IEEE Trans. Syst. Man Cybern., vol. SMC–3, 28–44 (1973)