A Neural Fuzzy Controller Learning by Fuzzy Error Propagation

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

Keywords: fuzzy control, fuzzy error, fuzzy error propagation, membership function, learning algorithm, neural network

Abstract

In this paper we describe a procedure to integrate techniques for the adaptation of membership functions in a linguistic variable based fuzzy control environment by using neural network learning principles. This is an extension to our work in [2].

We solve this problem by definining a fuzzy error that is propagated back through the architecture of our fuzzy controller. According to this fuzzy error and the strength of its antecedent each fuzzy rule determines its amount of error. Depending on the current state of the controlled system and the control action derived from the conclusion, each rule tunes the membership functions of its antecedent and its conclusion. By this we get an unsupervised learning technique that enables a fuzzy controller to adapt to a control task by knowing just about the global state and the fuzzy error.

1 Introduction

One of the design problems of a fuzzy controller is the choice of appropriate membership functions or the tuning of a priori membership functions in order to improve the performance of the fuzzy controller.

We solve this problem by definining a fuzzy error that is propagated back through the neural-like architecture of our fuzzy controller. According to this fuzzy error, the strength of its antecedent, the current state of the controlled system, and the control action derived from the conclusion, each fuzzy rule determines its amount of error and tunes the membership functions of its antecedent and its conclusion. This paper is an extension to our work in [2], where we proposed a supervised learning algorithm depending on a non-fuzzy error.

We refrained from just integrating neural nets in certain parts of the architecture as black boxes as it is done in other approaches, or from adding an extra module to the architecture taking care of the correction of errors for example by weighting the rules according to the errors as it is described in [4, 12].

Our main concern is to keep the structure of the fuzzy controller that is determined by the fuzzy rules. We think of those rules as a piece of structural knowledge that gives us a roughly correct representation of the system to be controlled. If the actual output of the controller differs from the desired behaviour, we consider an unsuitable choice of membership functions that model the linguistic values of the system variables to be responsible [8].

We understand the adaptations of the membership functions as a reverse mechanism deduced from the forwarding inference machinery. We consider the computation of the control value from given measured input

In: Proc. NAFIPS '92, Puerto Vallarta,Mexico, Dec. 14–17 (1992), 388–397

values as a feedforward procedure like in layered neural nets [10], where the inputs are forwarded through the net resulting in some output values. If the actual output is not able to drive the controlled system to a desired state, an error has to be propagated back through the architecture changing parameters taking into account the feed forward propagation of inputs.

Because it is usually not possible to determine an optimal control action for a given state, we are not able to calculate the error of the produced output directly. This means we cannot use a supervised learning procedure like standard backpropagation. But by evaluating the state of the controlled system, we are able to determine a global error measure. This enables us to define a non-supervised learning algorithm. Training a fuzzy controller with such a learning procedure allows us to keep track of the changes and to interpret the modified rules.

The term "non-supervised" indicates in this context, that there is no "teacher" providing a desired output value to be compared to the actual output value. The controller is able to calculate the fuzzy error by just knowing about the state of the plant. From another point of view one could say, that the system is watched by a supervisor who uses "good" and "bad" signals to guide the learning procedure. But this kind of reinforcement learning is not considered to be a plain supervised procedure, and so we prefer to call our learning algorithm non-supervised, although it is derived from the BP-algorithm for neural networks [10].

Considering the ideas on which fuzzy controllers are based, we think it is a natural approach to use a fuzzy error for our system, which, according to its structure, we may call neural fuzzy controller.

In the following sections we first present the structure of our controller. Then we describe the fuzzy error propagation algorithm that we use as our learning procedure. Next we consider some simulation results concerning the control of an inverted pendulum and in the last section we discuss our results.

2 The Neural Fuzzy Controller

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. For each of the mentioned variables 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.

The control action that drives the system S to a desired state is described by the well-known concept of fuzzy if-then rules [13], where a conjunction of input variables associated with their respective linguistic values determine 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. The conjunction of the inputs is usually done by the min-operation, and for the aggregation of the outputs of the rules the max-operation is usually chosen, as it is done by the well-known Zadeh-Mamdani procedure [7, 13].

For the evaluation of fuzzy rules the defuzzyfication-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 [5, 6]. 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, 6]. 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 defuzzification is carried out by

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

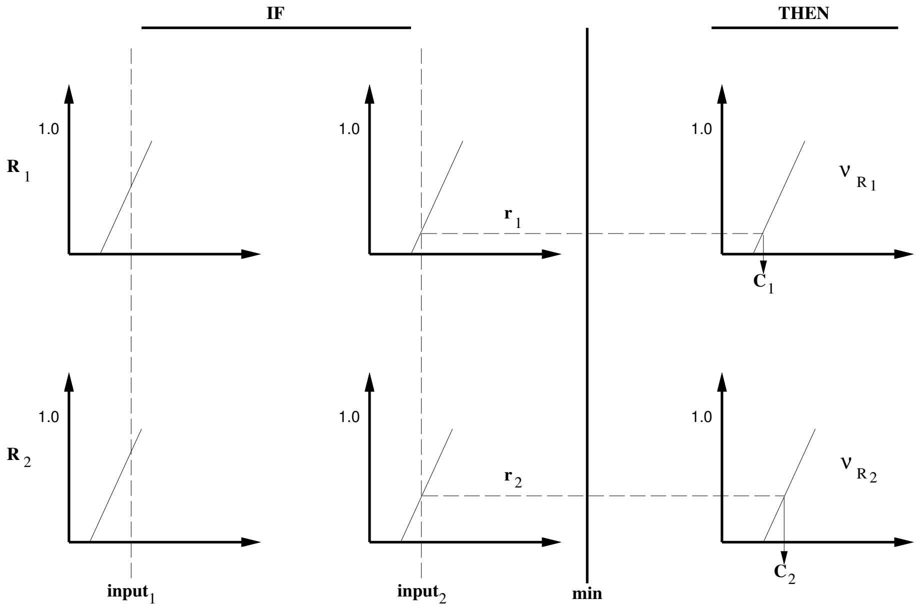

Figure 1: Defuzzification using Tsukamoto's monotonic membership functions

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

Consider the following two rules

R_1: **IF** θ is PM **AND** $\dot{\theta}$ is PS **THEN** F is PS,
R_2: **IF** θ is PS **AND** $\dot{\theta}$ is PZ **THEN** F is PZ,

where PM, PS and PZ represent the usual linguistic expressions *positive medium, positive small* and *positive zero.* The evaluation of those rules is presented in figure 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, even if we do not make use of this possibility in our controller for reasons of simplicity.

An example for the structure of our neural fuzzy controller is depicted in figure 2. The modules X_1 and X_2 represent the input variables that describe the state of the system to be controlled (plant, for short). These modules deliver their crisp values to their μ-modules which contain the membership functions interpreted as linguistic values assigned to the respective input variables. The μ-modules are connected to the following R-modules which represent the fuzzy if-then rules, the knowledge base of the controller. Each μ-module gives to its connected R-modules, the membership value $\mu_{ij}(x_i)$ of its input variable X_i. It is possible for each μ-module to be connected to several R-modules. 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 linguistic values of 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, which is usually a fuzzy set in a more general case. The conclusions are then passed on to the C-module where they are aggregated (e.g. by max-operation), and a crisp control value is determined by a defuzzification procedure.

In our case, however, monotonic membership functions are used, and so the ν-modules pass pairs $(r_i, \nu_k^{-1}(r_i))$ to the C-module, where the final output value is calculated by

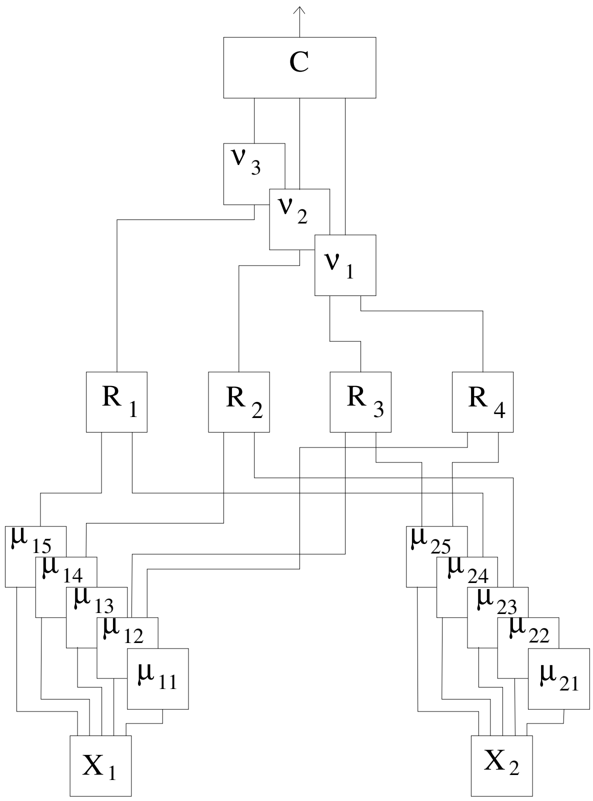

Figure 2: The structure of the neural fuzzy controller

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

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

From a more general point of view one can interpret the messages from the ν-modules to the C-module as fuzzy sets clipped by the min-implication at height r_i. The C-module aggregates the conclusions by a max-operation, and uses a non-standard defuzzification procedure as it is mentioned above.

As one can easily see, the system in figure 2 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 X- and C-layer are identified as input layer, and output layer, respectively, and the R-layer serves as the intermediate or hidden layer that constitutes the internal representation 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 [9]. 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, 3] and it is possible that one linguistic value is represented by different membership functions.

3 The Fuzzy Error Propagation Algorithm

Our goal is to tune the membership functions of the controller by a learning algorithm. Because it is usually not possible to calculate the optimal control action for a given state of the plant, so we can derive the error directly by comparing the optimal to the actual value, we are trying 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 our neural fuzzy controller.

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

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

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

$$G_2 = \min(\mu^{compensate_1}(X_1, \ldots, X_n), \ldots, \mu^{compensate_k}(X_1, \ldots, X_n))$$

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

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

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 that is made by our neural fuzzy controller is defined as

$$E = 1 - G.$$

We are now able to define our learning algorithm that works for each fuzzy rule in parallel. Each rule R_i knows the value r_i of the conjunction of its antecedents and the value c_i of its conclusion. Because we are using monotonic membership funcions, c_i is already crisp. 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. The actual control value cannot be determined, but its direction, i.e. $\mathrm{sgn}(c_{\mathrm{opt}})$, is known. For the case $\mathrm{sgn}(c_i) = \mathrm{sgn}(c_{\mathrm{opt}})$ 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 follwing 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.

Compared to learning algorithms used in neural networks one can see, that the error is not just passed back through the system, but that it is propagated to the intermediate layer constituted by the R-modules, where a rather sophisticated evaluation of this error is carried out, which is not typical for connectionistic systems. There the error signal is treated equally by each component of the network. In our system the R-modules propagate the error back and forward to the μ- and ν-modules, respectively, where less complicated calculations lead to a change of the membership functions, the "fuzzy weights" from a connectionistic point of view.

The neural fuzzy controller has not to learn from scratch, 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.

4 Controlling an Inverted Pendulum

In this section we present the results of a simulation of the neural fuzzy controller applied to the inverted pendulum (figure 3). An inverted pendulum is a well-known nonlinear dynamical system, often used to test fuzzy controllers. The sytem is described by two state variables that are the input variables to the controller, the angle θ measured against the upright position and the angle velocity $\dot{\theta}$, which also describe the error and the change of error. The pendulum is controlled by one control variable that is the control output, the force F applied to the base of the rod that is only allowed to fall to the left or to the right. We use a simplified version of the inverted pendulum in our simulation. The system 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 membershipfunctions that are not symmetric, we

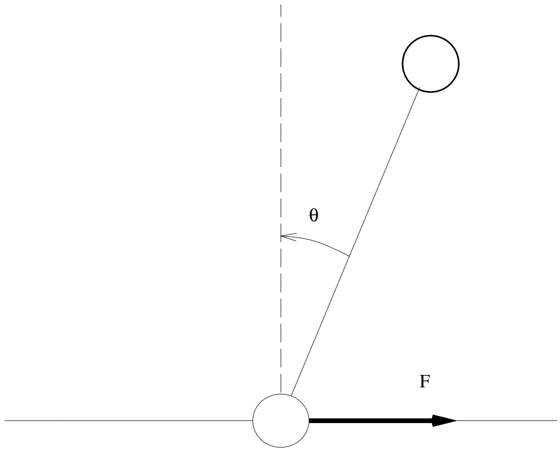

Figure 3: The inverted pendulum

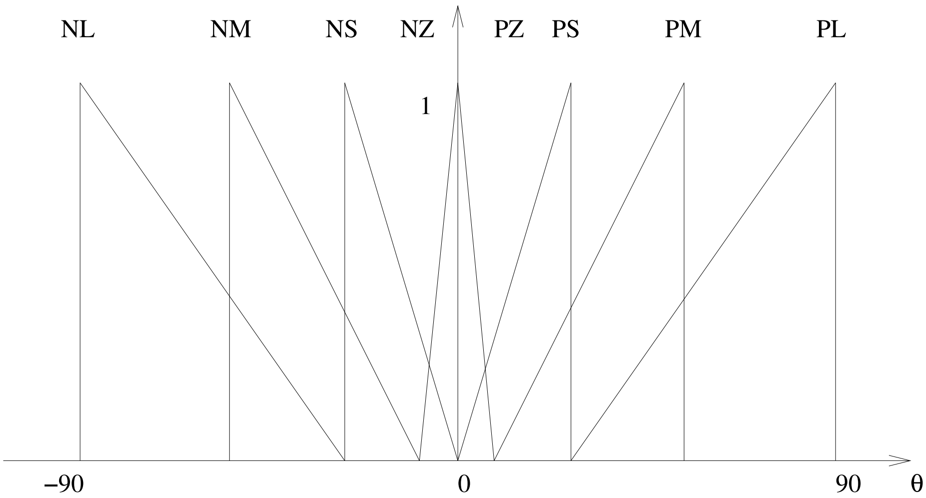

Figure 4: Monotonic membership functions modelling the linguistic values of θ

modell the value Zero as PositiveZero and NegativeZero. A typical definition for the membership functions used in our simulation is depicted in figure 4. The exact initial values of the above mentioned characterizing point b with $\mu(b) = 1$ and the range d defined as $b - a$ with $\mu(a) = 0$ for each membership function can be found in the following tables.

The rule base used to construct the controller is presented in figure 5. It was found that the controller was already able to balance the pendulum with this rule base and the initial membership functions. However, after the learning procedure was activated, the performance of the system became much better. It was also found that the controller was not able to cope with extreme initial positions of the pendulum, e.g. $\theta = 20$ and $\dot{\theta} = 2$, when only the initial membership functions were used. But when the learning algorithm was applied the controller was able to balance the rod after only three trials, i.e. the pendulum fell down ($|\theta| > 90$) just twice.

The results of the simulation can be found in table 1. We have only documented the changes in the membership functions of θ. For the other two variables similar changes have been found.

					$\dot\theta$				
		PL	PM	PS	PZ	NZ	NS	NM	NL
	PL	PL							
	PM		PM	PS	PS				
	PS		PS	PS	PS				
θ	PZ			PS	PZ				
	NZ					NZ	NS		
	NS					NS	NS	NS	
	NM					NS	NS	NM	
	NL								NL

Figure 5: The rule base of the neural fuzzy controller

The fuzzy error E has been defined by

$$E = \begin{cases} 1 - \min(1 - \frac{|\theta|}{3}, 1 - \frac{|\dot\theta|}{0.3}) & \text{if } \operatorname{sgn}(\theta) = \operatorname{sgn}(\dot\theta),\ -3 \le \theta \le 3,\ -0.3 \le \dot\theta \le 0.3 \\ \frac{|\theta + 10\,\dot\theta|}{3} & \text{if } \operatorname{sgn}(\theta) \ne \operatorname{sgn}(\dot\theta),\ -3 \le \theta + 10\,\dot\theta \le 3 \\ 1 & \text{otherwise.} \end{cases}$$

That means the fuzzy error is defined by one two-dimensional and two one-dimensional membership functions. The learning rate σ has been set to 0.01.

			Run 1	Run 2	Run 3	Run 4	Run 5
	b	initial d	$\theta = 0$ $\dot\theta = 0.1$	$\theta = 10$ $\dot\theta = 1$	$\theta = 20$ $\dot\theta = 1$	$\theta = 20$ $\dot\theta = 2$	$\theta = 30$ $\dot\theta = 2$
NL	-90	60.0000	60.0000	60.0000	60.0000	60.0000	60.0000
NM	-70	60.0000	60.0000	60.0000	60.0000	60.0000	171.9157
NS	-40	40.0000	48.9108	47.5219	49.0139	64.3483	74.5797
NZ	0	-13.0000	-8.0298	-9.5576	-10.0961	-9.6819	-9.2074
PZ	0	13.0000	31.4427	17.6432	13.5961	13.6028	15.1134
PS	40	-40.0000	-49.8010	-48.2032	-50.0540	-47.7372	-60.5255
PM	70	-60.0000	-60.0000	-61.0686	-62.0523	-82.9168	-180.7220
PL	90	-60.0000	-60.0000	-60.0000	-60.0000	-60.0000	-60.0000

Table 1: The changes in the ranges of the membership functions of θ

The next table shows the performance of the controller for each of the five runs. Each run consisted of 3000 loops and has been performed with and without the learning procedure. The performance is measure by the average over all absoulte values of θ measured during a run with n loops:

$$\bar\theta = \sum_{i=1}^{n} \frac{|\theta|}{n}.$$

The controller was able to keep an angle near zero with activated learning procedure, and was also able to balance the pendulum beginning from the extreme positions of runs 4 and 5 in 3 and 11 trials, respectively, whereas the controller was not able to balance (n.a.t.b.) the rod in these cases without learning.

	run 1	run 2	run 3	run 4	run 5
$\bar{\theta}$ without learning	4.05	4.17	4.29	n.a.t.b.	n.a.t.b.
$\bar{\theta}$ with learning	0.56	0.65	0.74	2.13	5.36
no. of trials	1	1	1	3	11

Table 2: The performance of the controller

5 Discussion

We have presented a learning algorithm for a neural fuzzy controller based on a fuzzy error measure. The structure of the controller resembles a neural network and the fuzzy error propagation can be compared to non-supervised learning procedures as they exist for certain kinds of connectionistic systems. Simulations of the controller have shown that the learning procedure improves the behaviour of the fuzzy controller and is able to handle situations where the non-learning controller fails.

The introduced fuzzy error measure is suitable for describing the performance of the controller and allows each rule to determine changes for the membership functions of its preconditions and its conclusion. The learning algorithm starts from a predefined rule base that can be obtained by clustering methods e.g. [11], 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 easily interpreted [8]. It is not possible that two rules use different fuzzy sets describing the same linguistic value.

Other neural fuzzy control environments, which are based more on neural network architectures [1, 3], often use factors to weight the rules or allow the rules to have different representations for the same input value. From our point of view in this case there are semantics involved, that are different to our approach. This has to be considered when an adaptive fuzzy control environmemt is used.

An extension to the presented learning algorithm that is used in combination with a fuzzy neural network that is capable to learn fuzzy rules and membership functions is presented in [9].

References

[1] Hamid R. Berenji: A Reinforcement Learning-Based Architecture for Fuzzy Logic Control. Int. J. Approximate Reasoning, vol. 6, no. 2, 267-292 (1992)

[2] P. Eklund, F. Klawonn, D. Nauck: Distributing Errors in Neural Fuzzy Control. Proc. 2nd Int. Conf. on Fuzzy Logic and Neural Networks (IIZUKA-92), 1139-1142, (1992)

[3] Jyh-Shing R. Jang: Fuzzy Modelling Using Generalized Neural Networks and Kalman Filter Algorithm. Proc. AAAI-91, vol. 2, 762-767, (1991)

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

[5] C.C. Lee: Fuzzy Logic Control Systems: Fuzzy Logic Controller – Part I. IEEE Trans. Syst. Man Cybern., vol. 20, no. 2, 404-418, (1990)

[6] C.C. Lee: Fuzzy Logic Control Systems: Fuzzy Logic Controller – Part II. IEEE Trans. Syst. Man Cybern., vol. 20, no. 2, 419-435, (1990)

[7] E.H. Mamdani: Applications of Fuzzy Algorithms for Simple Dynamic Plant. Proc. IEE vol. 121, 1585-1588 (1974)

[8] D. Nauck, R. Kruse: Interpreting Changes in the Membership Functions of a Self–Adaptive Neural Fuzzy Controller. Proc. 2nd Int. Workshop on Industrial Applications of Fuzzy Control and Intelligent Systems, Dec. 2-4, 1992, College Station, Texas, (1992)

[9] D. Nauck, R. Kruse: A Fuzzy Neural Network Learning Fuzzy Control Rules and Membership Functions by Fuzzy Error Backpropagation. Submitted for publication, (1992)

[10] D.E. Rumelhart, J.L. McClelland: Parallel Distributed Processing, Vol. 1. MIT Press, Cambridge, Massachusetts (1986)

[11] H. Takagi, I. Hayashi: NN–driven Fuzzy Reasoning. Int. J. Approximate Reasoning vol. 5 no. 3, 191-212 (1991)

[12] S. Shao: Fuzzy Self-Organizing Controller and its Application for Dynamic Processes. Fuzzy Sets and Systems 26, 151-164 (1988)

[13] 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)