Distributing Errors in Neural Fuzzy Control

P. Eklund
Department of Computer Science
Åbo Akademi University
SF-20520 Åbo, Finland

F. Klawonn D. Nauck
Department of Computer Science
Technical University of Braunschweig
W-3300 Braunschweig, Germany

Key words: neural network, fuzzy control, membership function, backpropagation

1 Introduction

The purpose of this paper is to describe a procedure to integrate techniques for the adaptation of membership functions in a linguistic variable based fuzzy control environment.

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 propose to solve this problem by adapting neural net learning techniques to an architecture of a fuzzy controller, but instead of integrating neural nets in certain parts of the architecture as black boxes, we show how an error in the resulting control value can be distributed among the control rules as well as between the antecedents and the conclusions of the corresponding rules.

A standard approach is to add an extra module to the architecture taking care of the correction of errors for example by weighting the rules according to the errors like it is described in [4]. Our main effort here is to understand adaptations as a reverse mechanism deduced from the forwarding inference machinery. We consider the computation of the control value from given measured input values as a feedforward procedure like in layered neural nets [6], where the inputs are forwarded through the net resulting in some output values. If the actual output differs from the desired output value the error has to be propagated back through the architecture changing parameters taking into account the feed forward propagation of inputs.

Training a fuzzy controller with such a learning procedure allows us to keep track of the changes and to interpret the modified rules.

In: Proc. 2nd Int. Conf. on Fuzzy Logic and Neural Networks IIZUKA'92, Iizuka, Jul. 17-22 (1992), 1139–1142

2 The inference scheme

Consider measurements in a subinterval $H = [h_1, h_2]$ of the real line. We will identify these measurements with corresponding images through the linear transformation I_H of H into $[0, 1]$. Thus, imprecision is modelled by mappings $\mu : [0, 1] \to [0, 1]$, in the sense of membership functions (msf's, for short), with the obvious interpretation as representations of linguistic values. For predicates we will have variables associated with msf's.

For the minMAX inference scheme we consider a variable together with a linguistic value to be considered as one predicate, i.e. we may hide the linguistic variable from the syntax. This is, of course, not to be taken as an argument against the use of linguistic variables.

As msf's we consider the range from triangular or trapezoidal forms to piecewise linear mappings, with corresponding parametric representations. We also distinguish between having the support of a msf either fixed or floating. For reasons of simplicity we restrict ourselves to the above mentioned msf's. However the described concepts can be extended to more general functions.

In the following we will set forward an architecture for the modelling of our adaptation process. For simplicity, we will consider single layer rule basis. Further, we may assume single outputs, since multiple outputs are represented by corresponding networks. Thus, a typical rule can be written as

$$Q \leftarrow P_1(x_1), \ldots, P_1(x_n).$$

To each variable x_i we assign an msf μ_i.

We now have the following molecules of the system architecture: an msf-module for the fuzzification, a T-module for the aggregation of inputs, an S-module for the aggregation of outputs, and the DEFUZ-module for the defuzzification. The usual choice of T and S are the min- and MAX-operations, respectively. Additionally, we may impose ad hoc learning strategies, i.e. for managing weightings of rules [4].

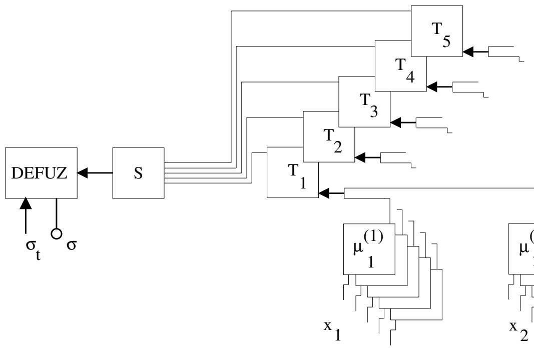

Figure 1: The system architecture

In fig. 1 we have two inputs, x_1 and x_2, for five rules, where msf's are given as $\mu_i^{(1)}, \ldots, \mu_i^{(5)}$, $i = 1, 2$.

The well-known Zadeh-Mamdani procedure [9, 5] can now be identified within the architecture. Measurements are handled by the msf-modules, and delivered to the T-modules, and further on through the S-module to the defuzzification. The output signal at σ is compared with the expected output at σ_t, and an error δ_σ is obtained.

The problem now is to generate adjustments on the input and output msf's, through some distribution of the error in the architecture. This is described in the following section. Other inference schemes as related to neural adaptations are developed in [1, 2, 8]. Consequences for the backpropagation algorithm are described in [3].

3 The learning procedure

Msf's can be adjusted either by laterally moving the domain or by bending segments of the function. The error $\delta_\sigma^{(i)}$ is considered to be a combination of errors resulting from lateral placement of domains, $\delta_\sigma^{(S,i)}$, and specification of function shapes, $\delta_\sigma^{(T,i)}$. These respective partial errors are now distributed over the backpropagating procedures in the S- and T-modules, respectively. We may write

$$\delta_\sigma^{(i)} = (\delta_\sigma^{(S,i)}, \delta_\sigma^{(T,i)})$$

with $\delta_\sigma^{(S,i)}$ and $\delta_\sigma^{(T,i)}$ being delivered to the S- and T-modules, respectively.

The subdivision of errors is related to the position of the output and target values. In a straightforward approach we might emphasize errors of function shapes if the output and target values are closer to the boundaries of the unit interval. In this case targets and actuals being more in the middle of the interval, would emphasize the lateral positions of the domains.

The subdivision of errors over input and output nodes is done accordingly. Details about these distribution strategies are described in an extended version of this paper.

We consider *center of gravity* (cog) and *mean of maxima* (mom) as possible defuzzification procedures. After we have obtained the crisp output value $\sigma = \mathrm{DEFUZ}(S)$ we are able to determine the error signal δ_σ to the difference $\sigma - \sigma_t$ between the output value σ and the target value σ_t.

We will now first consider the mom-procedure. There are four possibilities for our output value σ to be wrong.

1. The target value is located under the top of S but shifted to the left or to the right (see fig. 2(a)).
2. The target value is a member of the support of the fuzzy set T_i, that has produced the top of S, but it is not located under the top of S (see fig. 2(b)).
3. The target value is not a member of the support of the fuzzy set T_i, that has produced the top of S, but it is still inside the support of S (see fig. 3(a)).
4. The target value is not a member of the support of S (see fig. 3(b)).

In the first two cases the top of S has been produced by the correct output fuzzy set T but it is displaced. We assume that only T_i is responsible for the incorrect σ and we try to overcome this error by changing the form of T_i in such a way, that the mean of the top of Y_i is σ_t (see the dotted lines in fig. 2) by trying to keep the width of the top.

In the third case the top of S has been produced by the wrong output fuzzy set. This error is due to the input fuzzy sets and so we first have to change them in order to assure that the top of S is produced by the correct T_i. Assuming that the top of S is produced by T_j and $\sigma_t \in \mathrm{support}(T_i)$. So we have to raise the minimum produced by the inputs $\mu_k^{(j)}$ and lower the minimum of the inputs $\mu_k^{(i)}$. Then we have to change T_i in the same way as in the first two cases.

In the fourth case, when $\sigma_t \notin \mathrm{support}(S)$, we assume an error in our rulebase. There is either no rule that covers the area of the current σ_t or the respective rule has not fired. In either case we have to introduce a new rule that copes with the situation. This cannot be done automatically by our learning procedure, but has to be done by the user.

The Error Propagation Algorithm (mom)

1. After the unit DEFUZ has produced σ it backpropagates the correct value σ_t and its output value σ to the S-unit.

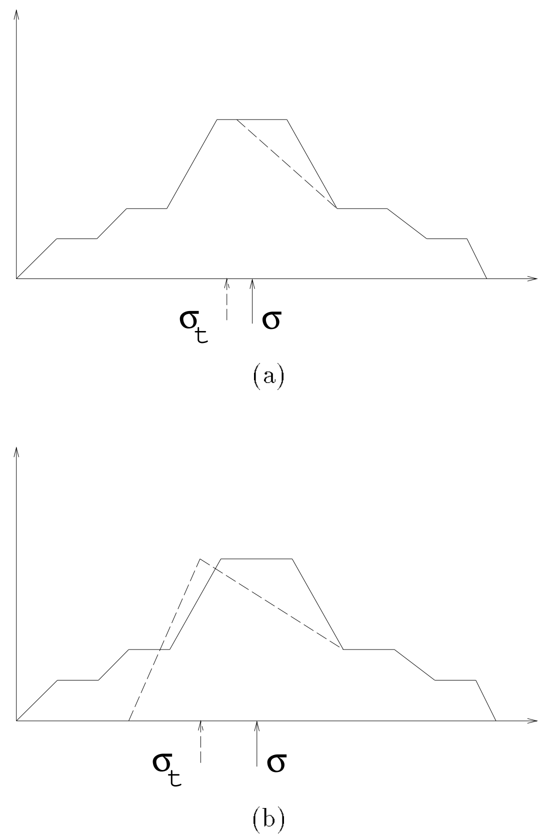

Figure 2: Adjusting σ by changing the form of the T fuzzy sets (mom)

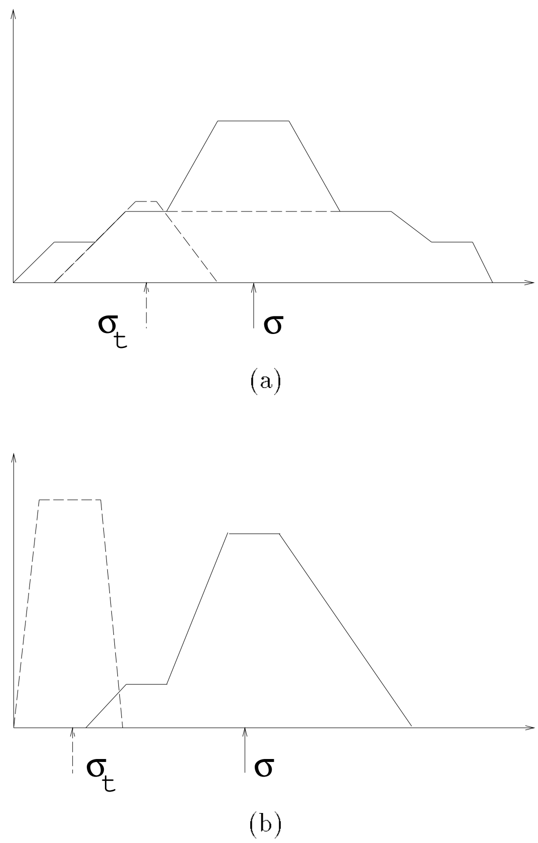

Figure 3: Adjusting σ by changing the form of the μ and T fuzzy sets (a) or by adding a new rule to the rule base (b) (mom)

2. Check if $\sigma_t \in \text{support}(S)$. If that is not true the user has to add a new rule into the rulebase and the current input has to be reevaluated before the learning process can proceed. Else go on with the next step.

3. The S-unit backpropagates σ, σ_t, the height of S and the maximum height $h_{\max}$ each output fuzzy set not producing the top of S is allowed to have to the T-units.

4. Each T-unit checks

   (a) **If** height(T_i) = height(S) **and** $\sigma_t \notin \text{support}(T_i)$
   Then send h_{max}, height(T_i) and a lower signal to all connected μ-units.

   (b) **If** (height(T_i) = height(S)) **and** $\sigma_t \in \text{support}(T_i)$ **and** $\sigma \neq \sigma_t$
   Then change the form of T_i such that σ_t instead of σ will be obtained as the mean of maximum.

   (c) **If** (height$(T_i) \neq$ height(S)) **and** $\sigma_t \in \text{support}(T_i)$
   Then send height(S), height(T_i) and a raise signal to all connected μ-units, assume height(S) will be received as the minimum of the μ-units and change the form of T_i such that σ_t instead of σ will be obtained as the mean of maximum.

   (d) **If** (height$(T_i) \neq$ height(S)) **and** $\sigma_t \notin \text{support}(T_i)$
   Then terminate the error propagation for this unit.

5. Each μ-unit checks

   (a) **If** there is an incoming lower signal **and** $\mu_i^{(j)}(x_i)$ = height(T_j)
   Then change the form of $\mu_i^{(j)}$ such that $\mu_i^{(j)}(x_i) = h_{\max}$

   (b) **If** there is an incoming raise signal **and** $\mu_i^{(j)}(x_i) <$ height(S)

Then change the form of $\mu_i(j)$ such that $\mu_i^{(j)}(x_i) = \text{height}(S)$

A similar algorithm can be derived for the cog-procedure and will be presented in an extended version of this paper. The main idea of this algorithm is to change the center of gravity either by shifting the T fuzzy sets or by lowering and raising parts of the S fuzzy set by changing the input fuzzy sets (see fig. 4).

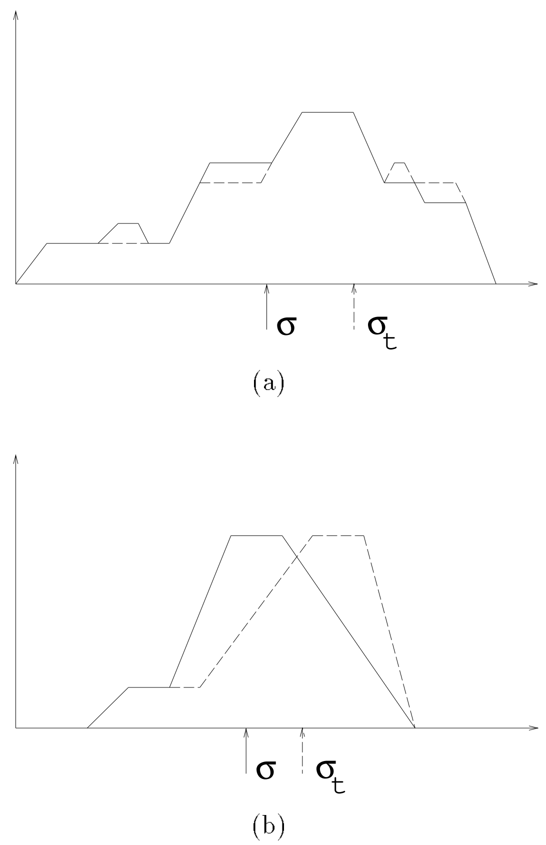

Figure 4: Adjusting σ by raising/lowering all or changing the form of one of the one T fuzzy sets (cog).

4 Discussion

We have presented an algorithm for distributing errors in a fuzzy control enviroment by adapting neural net learning techniques. We have considered a fuzzy controller as a neural-like system where the characteristics of the model are not determined by weighted connections but by fuzzy sets stored in the nodes of the system. The learning algorithm identifies the units responsible for the error in the output signal and backpropagates information through the network that enables the units to modify their fuzzy sets in a simple way. There is no need for attaching weights to the rules, each rule is equally important. By changing the fuzzy sets, the semantics of each rule intended by the user keeps unchanged, but the errors caused by an inaccurate modelling will be removed and the refined rules can be easily interpreted. We assume that the rules are given in at least a crude manner. If one has no ideas about the rules at all, there are clustering algorithms based on neural networks for the generation of rules [7].

References

[1] P. Eklund, F. Klawonn: A Formal Framework for Fuzzy Logic Based Diagnosis. In: R. Lowen, M. Roubens (eds.): Proc. 4th International Fuzzy Systems Association (IFSA) Congress: Mathematics, Brussels (Belgium), 58–61 (1991)

[2] P. Eklund, F. Klawonn: Neural Logic Programming. Manuscript, Dept. of Computer Science, Åbo Akademi University, Åbo, Finland (1991)

[3] P. Eklund, T. Riissanen, H. Virtanen: On the Fuzzy Logic Nature of Neural Nets. Proc. NeuroNimes'91, 293–300 (1991)

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

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

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

[7] H. Takagi, I. Hayashi: NN–driven Fuzzy Reasoning. Int. J. Approximate Reasoning 5 no 3 (1991)

[8] H. Virtanen: Combining and Incrementing Fuzzy Evidence – Heuristic and Formal Approaches to Fuzzy Logic Programming. In: R. Lowen, M. Roubens: Proc. 4th International Fuzzy Systems Association (IFSA) Congress: Artificial Intelligence, Brussels (Belgium), 200–203 (1991)

[9] L.A. Zadeh: Outline of a New Approach to the Ananlysis of Complex Systems and Decision Processes. IEEE Trans. Syst. Man Cybern., vol. SMC–3, 28–44 (1973)