A Fuzzy Perceptron as a Generic Model for Neuro–Fuzzy Approaches

Detlef Nauck
Department of Computer Science
Technical University of Braunschweig
Bueltenweg 74 / 75, D-38106 Braunschweig, Germany
Tel.: +49.531.391.3155, Fax: +49.531.391.5936, Email: nauck@ibr.cs.tu-bs.de

Abstract

This paper presents a fuzzy perceptron as a generic model of multilayer fuzzy neural networks, or neural fuzzy systems, respectively. This model is suggested to ease the comparision of different neuro–fuzzy approaches that are known from the literature. A fuzzy perceptron is not a fuzzification of a common neural network architecture, and it is not our intention to enhance neural learning algorithms by fuzzy methods.

The idea of the fuzzy perceptron is to provide an architecture that can be initialized with prior knowledge, and that can be trained using neural learning methods. The training is carried out in such a way that the learning result is interpretable in the form of linguistic fuzzy if–then rules. Next to the advantage of having a generic model to compare neuro–fuzzy models, the fuzzy perceptron can be specialized e.g. for data analysis and control tasks.

1 Introduction

Combinations of neural networks and fuzzy systems have become very popular during the last two years [Berenji, 1992; Berenji and Khedkar, 1992; Jang, 1991; Nauck and Kruse, 1993; Nauck et al., 1993; Nauck and Kruse, 1994; Nauck, 1994b; Takagi and Hayashi, 1991]. Most of these approaches are not easy to compare because they use very different architectures, activation functions, propagation and learning algorithms, etc. In this paper a fuzzy perceptron as a generic model of multilayer fuzzy neural networks is presented, and this model is proposed as a common base for neural fuzzy architectures in order to ease the comparision of different approaches. We are not interested in fuzzifying a multilayer perceptron in the whole, however we want to present an opportunity to put prior knowledge into the network to shorten the learning process, and to supply means to interpret the learning result in the form of linguistic rules.

By applying additional constraints to the definition of the fuzzy perceptron one can obtain a structure that can be interpreted as a usual fuzzy controller. By this a neural fuzzy controller like NEFCON can be easily created. Applicational aspects that also show how a NEFCON

In: Proc. Fuzzy–Systeme'94, 2nd GI–Workshop, Munich, Siemens Corporation, October 1994

model can be derived from the generic fuzzy perceptron can be found in [Nauck and Kruse, 1993; Nauck and Kruse, 1994; Nauck, 1994a; Nauck et al., 1994].

The model of the fuzzy perceptron is defined in section 2, and in section 3 a generic learning algorithm based on a fuzzy error is presented that can be used both for reinforcement learning and supervised learning. In section 4 we show how to refine this learning algorithm for a special fuzzy perceptron using supervised learning. We conclude with a section on showing how to derive a NEFCON architecture from a fuzzy perceptron for application to control tasks.

2 The Fuzzy Perceptron

A fuzzy perceptron has the architecture of an usual multilayer perceptron, but the weights are modelled as fuzzy sets and the activation, output, and propagation functions are changed accordingly. The intention of this model is to be interpretable in form of linguistic rules and to be able to use prior rule based knowledge, so the learning has not to start from scratch. This are also the intentions of the several approaches to neural fuzzy control [Berenji and Khedkar, 1992; Jang, 1991; Nauck and Kruse, 1993; Takagi and Hayashi, 1991]. These kind of models are usually created for control purposes only and are motivated by a need for an adaptive tuning process for fuzzy controllers. In this paper we suggest a generic model for neural fuzzy systems based on a 3-layer fuzzy perceptron. By using it to derive neural fuzzy systems for special domains, it would be possible to evaluate these different neuro-fuzzy approaches by means of the same underlying model.

Definition 1 *A 3–layer fuzzy perceptron is a 3–layer feedforward neural network* $(U, W, \mathrm{NET}, A, O, \mathrm{ex})$ *with the following specifications:*

(i) $U = \bigcup U_i$ $(i \in \{1,2,3\})$ *is a non–empty set of units. For all* $i, j \in \{1,2,3\}, U_i \neq \emptyset$ *and* $U_i \cap U_j = \emptyset$ *with* $i \neq j$ *holds.* U_1 *is called input layer,* U_2 *rule layer (hidden layer), and* U_3 *output layer.*

(ii) The structure of the network (connections) is defined as $W : U \times U \to \mathcal{F}(\mathbb{R})$, *such that there are only connections* $W(u,v)$ *with* $u \in U_i,\ v \in U_{i+1} (i \in \{1,2\})$ $(\mathcal{F}(\mathbb{R})$ *is the set of all fuzzy subsets of* $\mathbb{R})$.

(iii) A *defines an activation function* A_u *for each* $u \in U$ *to calculate the activation* a_u

(a) for input and rule units $u \in U_1 \cup U_2$: $\quad A_u : \mathbb{R} \to \mathbb{R}, \quad a_u = A_u(\mathrm{net}_u) = \mathrm{net}_u,$

(b) for output units $u \in U_3$: $\quad A_u : \mathcal{F}(\mathbb{R}) \to \mathcal{F}(\mathbb{R}), \quad a_u = A_u(\mathrm{net}_u) = \mathrm{net}_u.$

(iv) O *defines for each* $u \in U$ *an output function* O_u *to calculate the output* o_u

(a) for input and rule units $u \in U_1 \cup U_2 : O_u : \mathbb{R} \to \mathbb{R}, \quad o_u = O_u(a_u) = a_u,$

(b) for output units $u \in U_3 : O_u : \mathcal{F}(\mathbb{R}) \to \mathbb{R}, \quad o_u = O_u(\mathrm{net}_u) = \mathrm{DEFUZZ}_u(\mathrm{net}_u),$ *where* DEFUZZ_u *is a suitable defuzzification function.*

(v) NET *defines for each unit* $u \in U$ *a propagation function* NET_u *to calculate the net input* net_u

(a) for input units $u \in U_1 : \mathrm{NET}_u : \mathbb{R} \to \mathbb{R}, \quad \mathrm{net}_u = ex_u,$

(b) for rule units $u \in U_2 : \mathrm{NET}_u : (\mathbb{R} \times \mathcal{F}(\mathbb{R}))^{U_1} \to [0,1],$
$\mathrm{net}_u = \underset{u' \in U_1}{\top} (W(u',u)(o_{u'}))$*, where* $\top$ *is a* t*-norm,*

(c) for output units $u \in U_3 : \mathrm{NET}_u : ([0,1] \times \mathcal{F}(\mathbb{R}))^{U_2} \to \mathcal{F}(\mathbb{R}), \quad \mathrm{net}_u : \mathbb{R} \to [0,1],$
$\mathrm{net}_u(x) = \underset{u' \in U_2}{\perp} (\top(o_{u'}, W(u',u)(x)))$*, where* $\perp$ *is a* t*-conorm.*
If the fuzzy sets $W(u',u)$*,* $u' \in U_2$*,* $u \in U_3$*, are monotonic on their support, and* $W^{-1}(u',u)(\tau) = x \in \mathbb{R}$ *such that* $W(u',u)(x) = \tau$ *holds, then the propagation function* net_u *of an output unit* $u \in U_3$ *can alternatively be defined as*

$$\mathrm{net}_u(x) = \begin{cases} 1 & \text{if } x = \dfrac{\sum_{u' \in U_2} o_{u'} \cdot m(o_{u'})}{\sum_{u' \in U_2} o_{u'}} \\ 0 & \text{otherwise} \end{cases}$$

with $m(o_{u'}) = W^{-1}(u',u)(o_{u'})$*. To calculate the output* o_u *in this case* $o_u = x$*, with* $net_u(x) = 1$ *is used instead of (iv.b).*

(vi) $\mathrm{ex} : U_1 \to \mathbb{R}$*, defines for each input unit* $u \in U_1$ *its external input* $\mathrm{ex}(u) = \mathrm{ex}_u$*. For all other units* ex *is not defined.*

A fuzzy perceptron can be viewed as a 3–layer perceptron that is "fuzzified" to a certain extent. Only the weights, the net inputs and the activations of the output units are modelled as fuzzy sets. It is neither intented to fully fuzzify a perceptron by applying the extension principle nor to model a inference system based on fuzzy logic in the narrow sense. A fuzzy perceptron is like a usual perceptron used for function approximation. The advantage is to interpret its structure in the form of linguistic rules, because the fuzzy weights can be associated with linguistic terms. The network can also be created partly or in the whole out of fuzzy if–then rules.

For this reason the fuzzy perceptron has been defined as a 3–layer feedforward network. One can also think of models where the output layer inputs to an additional hidden (rule) layer, after which a new output layer is placed. This way fuzzy perceptrons can be build with an odd number of hidden layers that alternately have different functionalities. Such a model could be interpreted as a system of cascaded linguistic rules.

3 The Learning Algorithm

The definition of a learning algorithm for the fuzzy perceptron is not as straightforward as for its neural counterpart because of the usually non differentiable t–norms and t–conorms used as activation functions for the units. So a gradient descent method cannot be used here.

In some of the recent neural fuzzy approaches the product, or some kind of "soft" minimum function instead of a t–norm are used, and a quotient of summations is computed instead of a t–conorm to obtain differentiable activations functions (see e.g. [Berenji and Khedkar, 1992; Jang, 1991]). By this a gradient–descent–like learning algorithm can be defined that updates parameters defining the fuzzy sets used in the respective approaches. These methods usually only use the sign of the gradient, and they have problems with membership functions that have non differentiable points like e.g. the commonly used triangular functions. By using fuzzy sets as weights it is possible to derive the following simple heuristic as a learning procedure for the fuzzy perceptron.

Definition 2 *Consider a 3–layer fuzzy perceptron with n input and m output units. Let $\mathcal{L}$ be a learning task consisting of sample patterns $p = (i^{(p)}, t^{(p)})$ where $i^{(p)} \in \mathbb{R}^n$ and $t^{(p)} \in \mathbb{R}^m$ denote the input and output vectors of pattern $p \in \mathcal{L}$. Let $u \in U_3$, and let $t_u^{(p)}$ denote the desired output of the output unit u given the input vector $i^{(p)}$, and let $o_u^{(p)}$ be the actual output value. Also let $range_u$ be the difference between the maximal and minimal output values for unit u. The fuzzy error $E_u^{(p)}$ of u given p is defined as*

$$E_u^{(p)} = 1 - exp\left(\beta\left(\frac{t_u^{(p)} - o_u^{(p)}}{range_u}\right)^2\right),$$

where $\beta \in \mathbb{R}$ is a scaling factor.

The scaling factor β is used to adjust the sensitivity of the fuzzy error tolerating more or less large differences between the target and the output values. This is useful if e.g. a fuzzy classification is sufficient, and there is no need to obtain exact values to classify a given input pattern.

Definition 3 *Consider a 3–layer fuzzy perceptron and a learning task $\mathcal{L}$. The generic fuzzy error backpropagation algorithm is defined as follows.*

(i) Choose any $p \in \mathcal{L}$ and propagate the input vector $i^{(p)}$

(ii) Determine

$$\delta_u^{(p)} = \begin{cases} \operatorname{sgn}(t_u^{(p)} - o_u^{(p)}) \cdot E_u^{(p)} & \text{for } u \in U_3 \\ \sum_{v \in U_3} a_u^{(p)} \cdot \delta_v^{(p)} & \text{for } u \in U_2 \end{cases}$$

(iii) Determine $\Delta_p W(u,v) = f(\delta_v^{(p)}, a_u^{(p)}, \mathrm{net}_v^{(p)})$, $(u \in U_i,\ v \in U_j,\ i,j \in M, j = i+1)$

Repeat these steps for all $p \in \mathcal{L}$ until the overall error

$$E = \sum_{p \in \mathcal{L}} \sum_{u \in U_3} E_u^{(p)}$$

is suffiently small after an epoch is complete.

The change in the fuzzy weight $W(u,v)$ defined in step (iii) of the above defined algorithm depends on the δ-signal of the unit v, and it may also be depend on the activation and net input. This dependency has to be specified according to the actual fuzzy perceptron that is in use. Usually the fuzzy sets will be defined by parametrized membership functions, so that the changes have to be specified in terms of these parameters. The example in the following section explains the learning strategy for a special fuzzy perceptron.

4 A Min-Max Fuzzy Perceptron

Consider a 3–layer fuzzy perceptron using min and max as t–norm and t–conorm. The defuzzification function is the center of area method (COA). We denote the antecedent weights by $\mu_{i,j} = W(u_i, u_j)$ with $u_i \in U_1$ and $u_j \in U_2$, and the conclusion weights by $\nu_{j,k} = W(u_j, u_k)$ with $u_j \in U_2$ and $u_k \in U_3$. Each fuzzy weight $\mu_{i,j}$ is defined by a triangular function determined by three parameters $l_{i,j}, c_{i,j}, r_{i,j}$ such that $[l_{i,j}, r_{i,j}]$ is the support of the fuzzy set, and $\mu_{i,j}(c_{i,j}) = 1$, $\mu_{i,j}(l_{i,j}) = \mu_{i,j}(r_{i,j}) = 0$, and $l_{i,j} \le c_{i,j} \le r_{i,j}$ holds for all membership functions:

$$\mu_{i,j}(x) = \begin{cases} \dfrac{x - l_{i,j}}{c_{i,j} - l_{i,j}} & \text{for } x \in [l_{i,j}, c_{i,j}), \\[2ex] \dfrac{r_{i,j} - x}{r_{i,j} - c_{i,j}} & \text{for } x \in [c_{i,j}, r_{i,j}], \\[2ex] 0 & \text{otherwise.} \end{cases}$$

The same holds for the fuzzy sets $\nu_{j,k}$. We use a learning task $\mathcal{L}$ with patterns $p = (i^{(p)}, t^{(p)})$, such that $i^{(p)} \in [0,1]^n$, and $t^{(p)} \in [0,1]^m$.

If the output value of an output unit has to be incremented to match the target value, the support of the conclusions has to be shifted to larger values, meaning the parameters l, c, and r have to be increased. In addition the membership degrees of the input values to the antecedents have to be increased. This can be achieved, by making the support of the antecedents wider, i.e. increasing r and decreasing l. In addition the parameter c is changed such that it approaches the respective input value. In fig. 1 an example of the effects of

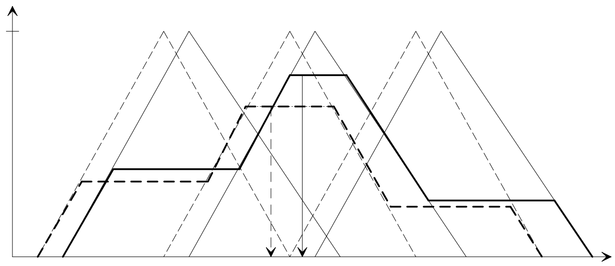

Figure 1: Adapting the membership functions (conclusions, - - - = before, —— = afterwards)

this strategy are displayed for the conclusions. If the output value has to be increased, and we start with the situation represented by the broken lines, we obtain the depicted new fuzzy sets, provided we also have an increase in the matching degree of the antecedents. The calculations used for the propagation of patterns in this example are the same as in a Mamdani fuzzy controller. The necessary calculations for the fuzzy error backpropagation are given in the following definition.

Definition 4 *Consider a 3–layer fuzzy perceptron using min and max as t–norm and t–conorm, and using COA defuzzification. The fuzzy error backpropagation algorithm for this kind of fuzzy perceptron is defined as follows:*
Set $E = 0$ and for each $p \in \mathcal{L}$ repeat the following steps:

(i) Choose any $p \in \mathcal{L}$ that has not already been choosen during the current epoch, and propagate the input vector $i^{(p)}$

(ii) Calculate δ_u for all $u \in U_2 \cup U_3$ (see Def. 3), and update the overall error E.

(iii) Determine the parameter changes for the fuzzy weights. For all $\nu_{j,k}$ with learning rates $\sigma_l, \sigma_c, \sigma_r \in \mathbb{R}$:

$$\begin{aligned}
\Delta_p l_{j,k} &= \sigma_l \cdot \delta_{u_k} \cdot (c_{j,k} - l_{j,k}),\\
\Delta_p c_{j,k} &= \sigma_c \cdot \delta_{u_k} \cdot (r_{j,k} - l_{j,k}),\\
\Delta_p r_{j,k} &= \sigma_r \cdot \delta_{u_k} \cdot (r_{j,k} - c_{j,k}),
\end{aligned}$$

and for all $\mu_{i,j}$ with learning rates $\eta_l, \eta_c, \eta_r \in \mathbb{R}$,:

$$\begin{aligned}
\Delta_p l_{i,j} &= -\eta_l \cdot \delta_{u_j} \cdot (c_{j,k} - l_{i,j}),\\
\Delta_p c_{i,j} &= \eta_c \cdot \delta_{u_j} \cdot (a_{u_i} - c_{i,j}),\\
\Delta_p r_{i,j} &= \eta_r \cdot \delta_{u_j} \cdot (r_{i,j} - c_{j,k}).
\end{aligned}$$

The algorithm stops, when E is suffiently small after an epoch has been completed, otherwise it has to be repeated for another epoch.

The different learning rates allow for changes of different magnitude in the parameters. This can be used e.g. to let the fuzzy sets shift only to a small extent from their initial position and to try to learn mainly by making the supports wider or smaller. This can be beneficial, if prior knowledge has been used to initialize the fuzzy weights. In addition to the above defined steps it has to be checked, that the changes in the parameters do not violate against the condition $l \leq c \leq r$, and that their values are still within the domain of the respective input or output unit's output function. One might also want to put some further constraints on the parameters to assure that some special semantical conditions are not violated.

This kind of architecture can be used for function approximation given a set of sample data. The trained system is then usable for classification purposes, control tasks, etc. In addition one receives a set of fuzzy if–then rules to interpret the structure of the sample data, if it is possible to assign linguistic labels to the fuzzy sets that developed during the training process. The last section shows how a NEFCON (NEural Fuzzy CONtrol) model can be derived from the fuzzy perceptron.

5 A Special Fuzzy Perceptron for Control Tasks

The NEFCON model is derived from the generic fuzzy perceptron, and it is used for control purposes. The learning algorithm is a special version of fuzzy error backpropagation and uses a reinforcement procedure that allows on–line learning.

A NEFCON system is a special 3–layer fuzzy perceptron with the following specifications:

(i) The input units are denoted as $\xi_1, \ldots, \xi_n$, the hidden rule units are denoted as $R_1, \ldots, R_k$, and the single output unit is denoted as η.

(ii) Each connection between units ξ_i and R_r is labelled with a linguistic term $A_{j_r}^{(i)}$ ($j_r \in \{1, \ldots, p_i\}$).

(iii) Each connection between units R_r and the output unit η is labelled with a linguistic term B_{j_r} ($j_r \in \{1, \ldots, q\}$).

(iv) Connections coming from the same input unit ξ_i and having identical labels, bear the same fuzzy weight at all times. These connections are called *linked conncetions.* An analogous condition holds for the connections leading to the output unit η.

(v) Let $L_{\xi,R}$ denote the label of the connection between the input unit ξ and the rule unit R. For all rule units R, R'

$$(\forall \xi \; L_{\xi,R} = L_{\xi,R'}) \Longrightarrow R = R'$$

holds.

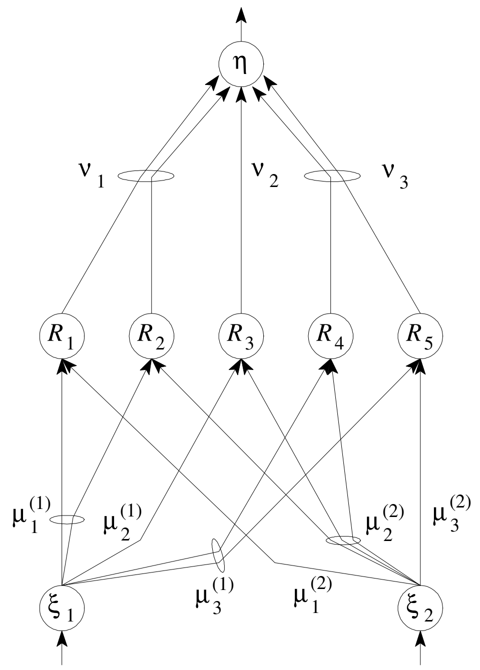

Figure 2: A NEFCON system with two input variables

This definition makes it possible to interpret a NEFCON system in terms of a fuzzy controller; each hidden unit represents a fuzzy if–then rule. Condition (iv) specifies that there have to be *shared weights.* If this feature is missing, it would be possible for fuzzy weights representing identical linguistic terms to evolve differently during the learning process. If this is allowed to happen, the architecture of the NEFCON system cannot be understood as a fuzzy rule base. Condition (v) determines that there are no rules with identical antecedents. A network that does not adhere to this condition is called *overcommitted NEFCON system,* and it is used for learning a linguistic rule base by deleting rule units [Nauck and Kruse, 1993; Nauck and Kruse, 1994].

NEFCON uses a reinforcement type version of the fuzzy error backpropagation algorithm to adapt its membership function on–line. This algorithm was also extended to enable the model to learn fuzzy rules. The training procedures are described at length in [Nauck and Kruse, 1994; Nauck et al., 1994]

NEFCON has been implemented in form of NEFCON–I, an X–Window based graphical simulation environment for UNIX machines that can be freely obtained by anonymous FTP. Establish an FTP connection to `ftp.ibr.cs.tu-bs.de` (= 134.169.34.15), and change to the directory `pub/local/nefcon/`, where you will find the source files, documentation, and further directories of binaries for SUN 4 and LINUX machines.

6 Conclusions

The presented 3–layer fuzzy perceptron is a useful combination of fuzzy systems and neural networks, and can be used as a generic model to explain and compare different neuro–fuzzy approaches. The model can be initialized by prior knowledge using fuzzy if–then rules and it can also be interpreted this way, i.e. the fuzzy perceptron is no black box. That the model is capable of learning has already been successfully shown by the NEFCON system that uses reinforcement learning and can be easily derived from the generic fuzzy perceptron [Nauck and Kruse, 1993; Nauck, 1994a; Nauck et al., 1994]. Future applications are currently considered in the area of data analysis.

References

H. R. Berenji (1992). *A Reinforcement Learning-Based Architecture for Fuzzy Logic Control.* Int. J. Approximate Reasoning, 6:267 –292.

H. R. Berenji and P. Khedkar (1992). *Learning and Tuning Fuzzy Logic Controllers Through Reinforcements.* IEEE Trans. Neural Networks, 3:724–740.

J.-S. R. Jang (1991). *Fuzzy Modeling Using Generalized Neural Networks and Kalman Filter Algorithm.* In *Proc. of the Ninth National Conf. on Artificial Intelligence (AAAI-91)*, pages 762–767.

R. Kruse, J. Gebhardt and R. Palm, eds. (1994). *Fuzzy Systems in Computer Science.* Vieweg, Braunschweig.

D. Nauck (1994a). *Building Neural Fuzzy Controllers with NEFCON–I.* In [Kruse et al., 1994], pages 141–151. Vieweg, Braunschweig.

D. Nauck (1994b). *Fuzzy Neuro Systems: An Overview.* In [Kruse et al., 1994], pages 91–107. Vieweg, Braunschweig.

D. Nauck, F. Klawonn and R. Kruse (1993). *Combining Neural Networks and Fuzzy Controllers.* In E. P. Klement and W. Slany, eds.: *Fuzzy Logic in Artificial Intelligence (FLAI93)*, pages 35–46, Berlin. Springer–Verlag.

D. Nauck, F. Klawonn and R. Kruse (1994). *Neuronale Netze und Fuzzy–Systeme.* Vieweg, Wiesbaden.

D. Nauck and R. Kruse (1993). *A Fuzzy Neural Network Learning Fuzzy Control Rules and Membership Functions by Fuzzy Error Backpropagation.* In *Proc. IEEE Int. Conf. on Neural Networks 1993*, pages 1022–1027, San Francisco.

D. Nauck and R. Kruse (1994). *NEFCON–I: An X–Window based Simulator for Neural Fuzzy Controllers*. In *Proc. IEEE Int. Conf. Neural Networks 1994 at IEEE WCCI'94*, pages 1638–1643, Orlando.

H. Takagi and I. Hayashi (1991). *NN–Driven Fuzzy Reasoning*. Int. J. Approximate Reasoning, 5:191–212.