Reducing the Number of Parameters of a Fuzzy System Using Scaling Functions

Frank Klawonn
Department of Computer Science
University of Applied Sciences Braunschweig/Wolfenbuettel
Salzdahlumer Str. 46/48
D-38302 Wolfenbuettel, Germany
e-mail: f.klawonn@fh-wolfenbuettel.de

Abstract

Learning techniques are tailored for fuzzy systems in order to tune them or even for deriving fuzzy rules from data. However, a compromise between accuracy and interpretability has to be found. Flexible fuzzy systems with a large number of parameters and high degrees of freedom tend to function as black boxes. In this paper, we introduce an interpretation of fuzzy systems that enables us to work with a small number of parameters without loosing flexibility or interpretability. In this way, we can provide a learning algorithm that is efficient and yields accuracy as well as interpretability. Our fuzzy system is based on extremely simple fuzzy sets and transformations using an interpretable scaling functions of the input variables.

Keywords: fuzzy function approximation, alternating optimization, gradient descend

1 Introduction

When L.A. Zadeh [9] introduced the notion of a fuzzy set in 1965, his original intention aimed at modelling human expertise. A fuzzy system was considered as a framework to formalize knowledge incorporating vagueness. However, later on, it turned out that for most of the applications it is not sufficient to simply specify the expert knowledge in terms of ad-hoc fuzzy rules. In order to design a system that performs well, it is necessary to choose and

tune a number of parameters like the shape and the location of the fuzzy sets, the number of rules or suitable aggregation operations. Therefore, a vast number of tuning methods, most of them based on neuro-fuzzy techniques (for an overview see for instance [7]) or evolutionary algorithms (for an overview see for instance [8]), have been designed for fuzzy systems.

In recent years, the emphasis in research has even been shifted further from tuning to machine learning and rule or knowledge extraction from data using fuzzy systems. One important aspect in this field of application of fuzzy systems is to find a balance or compromise between accuracy and interpretability [1]. A fuzzy system with a large number of parameters and a high degree of freedom might be capable of fitting very well to a given data set to which it should adapt. However, the price for the high accuracy might result in a fuzzy system that is barely understandable any more. In this case, where the fuzzy system simply functions as a black box, there are usually better and faster techniques for learning from data. On the other hand, putting a strong emphasis on the interpretability of the resulting fuzzy system, limiting the number of parameters and the degrees of freedom, might lead to poor performance in terms of accuracy that is not acceptable for real world applications.

The paper is organized as follows. In section 2 we briefly review the basic terminology of fuzzy controllers or fuzzy rules for function approximation that is needed for this paper. Section 3 takes a closer look at a certain common way of adapting fuzzy sets and how this can be interpreted in terms of scaling or transformations. Based on these ideas we develop an efficient learning scheme for this kind of fuzzy system in section 4. A detailed example is presented in section 5. Finally, in the conclusions we briefly outline, how our ideas might be extended to fuzzy classifiers.

2 A simple fuzzy system

In this paper, we consider fuzzy systems suited for fuzzy control as well as for function approximation or regression problems. We assume that we have k continuous input variables $X_1, \ldots, X_k$ and one continuous output variable Y. We assume that the fuzzy system uses rules of the following type:

$$R: \text{ If } X_1 \text{ is } \mu_1^{(R)} \text{ and } \ldots \text{ and } X_k \text{ is } \mu_k^{(R)} \text{ then } Y \text{ is } y^{(R)}. \tag{1}$$

$\mu_1^{(R)}, \ldots, \mu_k^{(R)}$ are fuzzy sets on the domains of the input variables $X_1, \ldots, X_k$, respectively. $y^{(R)} \in \mathbb{R}$ is the crisp output value assigned to the rule R. Technically speaking, we consider a simple Takagi-Sugeno controller with constant output functions.

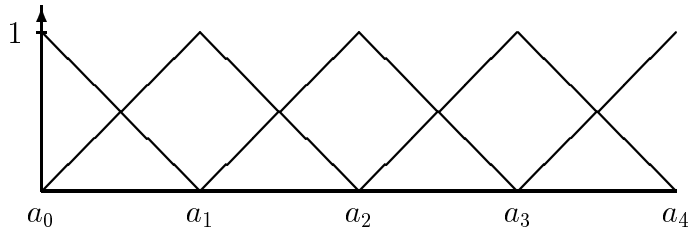

Figure 1: A uniform fuzzy partition.

Of course, in applications the fuzzy sets occurring in a rule as described by (1) would be replaced by linguistic terms like *approximately zero*, *small*, *big* etc.

Let $\mathcal{R}$ be a finite set of rules of the form (1). Then, for a given input tuple $(x_1, \ldots, x_k)$, the output of this rule base is given by

$$F_{\mathcal{R}}(x_1, \ldots, x_k) \;=\; \frac{\sum_{R \in \mathcal{R}} \mu_R(x_1, \ldots, x_k) \cdot y^{(R)}}{\sum_{R \in \mathcal{R}} \mu_R(x_1, \ldots, x_k)} \tag{2}$$

where

$$\mu_R(x_1, \ldots, x_k) \;=\; \mu_1^{(R)}(x_1) \odot \ldots \odot \mu_k^{(R)}(x_k) \tag{3}$$

is the firing degree of rule R and $\odot$ is an operation that corresponds to a suitable t-norm like the minimum, the product or the Łukasiewicz t-norm. Later on we will require that $\odot$ is differentiable almost everywhere – a condition that is satisfied by the three above mentioned t-norms as well as by most of the t-norms used in practical applications.

3 The idea of scaling and its interpretation

Since we restrict our considerations to fuzzy rules with a constant value in the conclusion, we do not have to bother about the defuzzification problem. The parameters that have to be specified are

- the fuzzy sets $\mu_i^{(R)}$,
- the output values $y^{(R)}$ and
- the aggregation operation (t-norm) $\odot$.

It is very common to start with a uniform "fuzzy partition" for each input variable X_i as it is shown in figure 1 and to adapt or tune the fuzzy sets – their shape and location – later on.

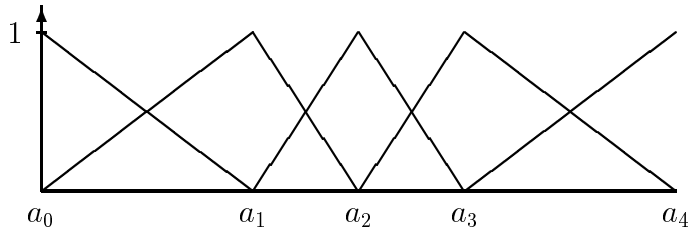

Figure 2: A modified fuzzy partition.

Especially in control applications, the resulting fuzzy partitions very often resemble the one shown in figure 2, at least in the case where the error and the change of the error are used as input variables. The fuzzy sets around zero (at the value a_2 in figure 2) are narrower than those further away from zero. The reason for this phenomenon is that the controller's actions must be very precise around the working or set point where the error is already small, whereas rough actions are sufficient, when the error is large. As long as the error is large – the process is still far away from the desired state – it is often sufficient to apply rough and strong control actions that drive the process to the vicinity of the desired state. Once this vicinity is reached, more careful actions have to be taken.

In order to better understand this phenomenon we take a look at a very simple fuzzy system with just one input variable X. Figure 3 shows the output function induced by the fuzzy system with the rules

$$R_i: \text{ If } X \text{ is } \mu^{(i)} \text{ then } Y \text{ is } y^{(i)}. \quad (i = 0, \ldots, 5) \tag{4}$$

The resulting function is – at least in this case with just one input variable – piecewise linear. In the region where the fuzzy sets are narrower, we can model greater variations of the output function than in the outer regions where the fuzzy sets are wider.

The change from the uniform fuzzy partition in figure 1 to the non-uniform fuzzy partition in figure 2 can be viewed in terms of modifying the fuzzy sets. But it can also be seen as a piecewise linear transformation. We obtain the same output function as in figure 3, when we use the uniform partition from figure 1, the same set of rules (4), but apply the transformation $t(x)$ shown in figure 4 as a first step, before we hand the input value over to the fuzzy system. We obviously have

$$\mu_i(x) \;=\; \tilde{\mu}_i(\tilde{x}) \;=\; \tilde{\mu}_i(t(x)) \tag{5}$$

where the μ_i are the fuzzy sets from the non-uniform and the $\tilde{\mu}_i$ from the uniform fuzzy partition.

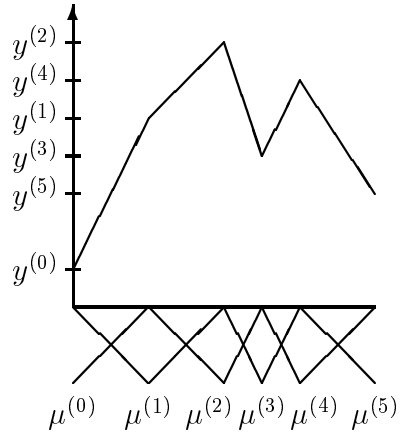

Figure 3: The piecewise linear function induced by a simple fuzzy system.

The transformation t has also a very appealing interpretation. In those regions where the graph of t is very steep, i.e. t where t has a high value for the derivative, the fuzzy sets μ are narrower. This will occur in those regions where the desired output has either strong variations or where a high accuracy of the output function is required.

It should be noted that the fuzzy sets μ_i of the non-uniform fuzzy partition induced by the transformation t via (5) are not necessarily of triangular shape, even if we assume that we have a piecewise linear transformation. Figure 5 shows this situation. Of course, a piecewise linear transformation will always lead to piecewise linear fuzzy sets. We can even drop the assumption that the transformation has to be piecewise linear. All we need is that it is continuous and non-decreasing. Even in this general case the fuzzy sets μ_i induced by (5) will be continuous and unimodal.

It should be noted that the transformation t must be non-decreasing. But it is not required that the interval ranges of the original variable and the transformed one are identical.

As mentioned before, the derivative of the function t, if it exists, plays an important role. The slope of t is inherited to the fuzzy sets μ that have the same slope in terms of the absolute value. And the slope can be interpreted as the accuracy needed in the region for approximating the considered function or achieving the necessary precision for control actions in the corresponding range. In this sense, the transformation t carries out a scaling of the domain of the input variable X. Ranges, where the function to be approximated varies heavily or where a high accuracy is required, are stretched or enlarged in order to specify the output more precisely. Regions where the function to be approximated does not vary much or where a higher error can be tolerated,

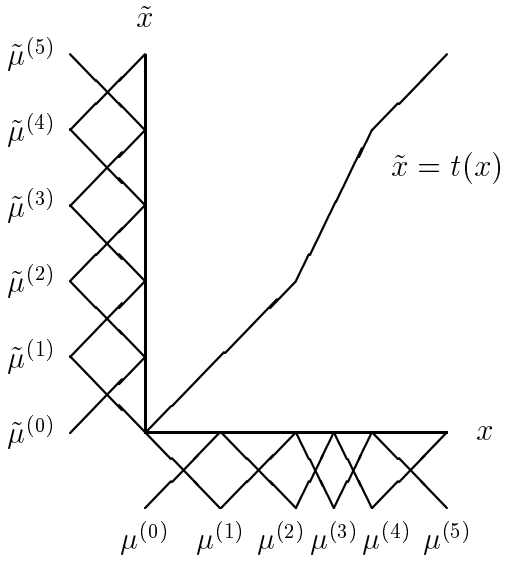

Figure 4: The piecewise linear transformation $\tilde{x} = t(x)$ between a uniform and a non-uniform fuzzy partition.

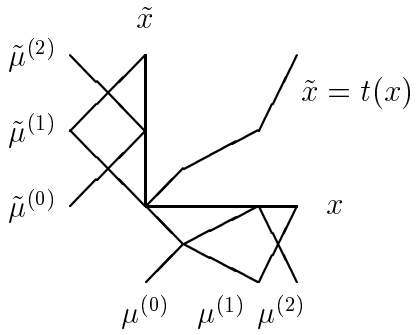

Figure 5: A piecewise linear transformation $\tilde{x} = t(x)$ inducing non-triangular membership functions.

are shrunk and covered by a smaller number of fuzzy sets and rules. With this idea of scaling in mind, we can also interpret the fuzzy rules (1) in the sense of

$$R: \quad \text{If } X_1 \text{ is approximatley } x_1^{(R)} \text{ and } \ldots \text{ and } X_k \text{ is approximately } x_k^{(R)} \text{ then } Y \text{ is } y^{(R)}. \tag{6}$$

The values $x_i^{(R)}$ represent those points where the corresponding fuzzy sets in figure 1 have membership degree 1. The similarity between the actual value x_i of the variable X_i and the specified value $x_i^{(R)}$ is then simply one minus the distance between x_i and $x_i^{(R)}$ in the scaled or transformed domain, i.e. the membership degree of x_i to the fuzzy set $\mu_i^{(R)}$ corresponding to the linguistic value *approximately* $x_i^{(R)}$ is given by

$$\mu_i^{(R)}(x_i) \;=\; \max\{1 - |t_i(x_i) - t_i(x_i^{(R)})|, 0\}$$

where t_i is the corresponding transformation associated with ith domain.

A similar idea of scaling functions and an interpretation of fuzzy controllers based on equality relations can be found [4, 5]. However, for our purposes it is sufficient to understand the simple fuzzy systems we consider here in the following way.

- For each input domain, we use a uniform fuzzy partition with triangular membership functions. Only the number of fuzzy sets has to be specified.
- In addition to these uniform fuzzy partitions, we need a suitable transformation for each input domain. The derivative of each transformation can be interpreted of how much accuracy is needed in the corresponding point or how strong the function to be described varies in the vicinity of this point.
- Furthermore, we need the fuzzy rules with the corresponding output values.

The choice of the type of transformation or scaling function depends on the application. As already mentioned, for many control applications high accuracy in the vicinity of the set point of the system is needed, whereas precision is not extremely important far away from the set point. This means that we should have high values for the derivative around the set point and lower values further away. Therefore, sigmoidal functions seem to be a good

choice in this case. However, in other applications like general fuzzy regression techniques for data analysis, other transformations might be desired in the form of polynomials or spline functions. Of course, in these cases we have to make sure that the transformations are always non-decreasing.

In the following section we develop for this kind of fuzzy system a simple and efficient scheme for learning from data.

4 The learning scheme

Let us assume we have a regression problem defined by a data set with n measured points of an unknown function with k input variables:

$$D \;=\; \{(x_{1,1}, \ldots, x_{1,k}, y_1), \ldots, (x_{n,1}, \ldots, x_{n,k}, y_n)\}.$$

For each input variable we choose the number of fuzzy sets. We also have to select which combinations of fuzzy sets will be considered as antecedents of the fuzzy rules. In principle, we can of course take all possible combinations into account. However, this can lead to a very high number of rules, when we have more than just a few input variables.

We also have to choose k parametric transformation functions

$$\tilde{x}_i \;=\; t_i(x; \alpha_1^{(i)}, \ldots, \alpha_{p_i}^{(i)}),$$

one for each input variable.

The aim is to choose the parameters $\alpha_j^{(i)}$ and the output values $y^{(R)}$ in such a way that the given data set D is approximated as well as possible. As an error measure we choose the most common approach, the sum of squared errors. This means, we have to determine the values $\alpha_j^{(i)}$ and $y^{(R)}$ in such a way that

$$E \;=\; \sum_{j=1}^{n} \left(F_{\mathcal{R}} \left(t_1(x_{j1}; \alpha_1^{(1)}, \ldots, \alpha_{p_1}^{(1)}), \ldots, t_k(x_{jk}; \alpha_1^{(k)}, \ldots, \alpha_{p_k}^{(k)}) \right) - y_j \right)^2 \quad (7)$$

is minimized, where $F_{\mathcal{R}}$ is computed according to (2).

Unfortunately, minimizing (7) is a nonlinear optimization problem. We propose to apply an alternating optimization scheme similar to the one that is used in fuzzy clustering (see for instance [2]). We alternatingly consider one set of parameters as fixed, while we optimize the other set of parameters. Here, this means that we first fix the parameters $\alpha_j^{(i)}$ and optimize the output values $y^{(R)}$, then we fix these optimized output values and optimize the $\alpha_j^{(i)}$ values and continue alternatingly, until the process converges. It is well

known (see for instance [3] or in a more general framework [6]) and also obvious from (2) that the regression function $F_{\mathcal{R}}$ is linear in the output values $y^{(R)}$, so that they can be computed directly by a standard least squares approach.

However, this simple solution does not work for the parameters $\alpha_j^{(i)}$ of the scaling functions, since $F_{\mathcal{R}}$ is non-linear in these parameters. Therefore, we apply a gradient descend technique to optimize these parameters. Thus we need the partial derivatives of (7):

$$\begin{aligned}\frac{\partial E}{\partial \alpha_j^{(i)}} &= 2\sum_{j=1}^{n}\left(F_{\mathcal{R}}\left(t_1(x_{j1};\alpha_1^{(1)},\ldots,\alpha_{p_1}^{(1)}),\ldots,t_k(x_{jk};\alpha_1^{(k)},\ldots,\alpha_{p_k}^{(k)})\right)-y_i\right)\\ &\quad\cdot\frac{\partial F_{\mathcal{R}}}{\partial \alpha_j^{(i)}}(x_{j1},\ldots,x_{jk}) \qquad (8)\end{aligned}$$

where

$$\begin{aligned}\frac{\partial F_{\mathcal{R}}}{\partial \alpha_j^{(i)}}(x_1,\ldots,x_k) &= \frac{\partial F_{\mathcal{R}}}{\partial \alpha_j^{(i)}}\left(t_1(x;\alpha_1^{(1)},\ldots,\alpha_{p_1}^{(1)}),\ldots,t_k(x;\alpha_1^{(k)},\ldots,\alpha_{p_k}^{(k)})\right)\\ &\quad\frac{\partial F_{\mathcal{R}}}{\partial \tilde{x}_i}(\tilde{x}_1,\ldots,\tilde{x}_k)\cdot\frac{\partial t_i(x_i;\alpha_1^{(i)},\ldots,\alpha_{p_i}^{(i)})}{\partial \alpha_j^{(i)}}(x_i) \qquad (9)\end{aligned}$$

and

$$\tilde{x}_i = t_i(x_i;\alpha_1^{(i)},\ldots,\alpha_{p_i}^{(i)}).$$

This is the general form of the gradient. $\frac{\partial F_{\mathcal{R}}}{\partial \tilde{x}_i}$ depends on the choice of the rules and the t-norm $\odot$ and $\frac{\partial t_i(x_i;\alpha_1^{(i)},\ldots,\alpha_{p_i}^{(i)})}{\partial \alpha_j^{(i)}}$ depends on the kind of the chosen transformation function. In order to optimize the parameters $\alpha_j^{(i)}$, we apply a gradient descend technique and compute

$$\alpha_{j,\text{new}}^{(i)} = \alpha_{j,\text{old}}^{(i)} - \eta\cdot\frac{\partial E}{\partial \alpha_j^{(i)}}$$

where η is the learning rate or the step size of the gradient method. We apply this formula iteratively to all $\alpha_j^{(i)}$ for a fixed number of steps or until convergence is roughly reached. Note that it is not necessary to fully iterate the gradient descend scheme until final convergence is reached, since the output values $y^{(R)}$ are re-adjusted in the next step anyway. Therefore, it is sufficient to get only close to the corresponding minimum by the gradient descent technique.

In the following section we provide a concrete example where we use the product t-norm and sigmoidal transformations.

5 An example

In this section we consider a more specific case. We choose the product t-norm for the operation $\odot$. This t-norm has two advantages. It does not cause any problems with the partial derivatives and in the case, when we use all possible combinations of fuzzy sets in the rule base, the denominator in (2) is always one and can therefore be neglected.

Since we use uniform fuzzy partitions (after the transformations have been applied), for any value of an input variable there are exactly two fuzzy sets yielding non-zero membership degrees – except for the case that when we go further to the left or to the right than indicated for the example in figure 3 or when we meet exactly one of the top points of the triangular membership functions. In figure three we assume that the fuzzy sets $\mu^{(0)}$ and $\mu^{(1)}$ are continued with membership degree one to the left and right, respectively. Of course, also in the case that one of the fuzzy sets yields membership degree one, all others yield membership degree zero. Let us neglect these extreme cases for the moment. Let $\mu_{\text{left}(x,i)}$ and $\mu_{\text{right}(x,i)}$ denote the left and right fuzzy sets, respectively, yielding a non-zero membership degree for the value x in the fuzzy partition for the variable X_i. Furthermore, let $y(\mu_1, \ldots, \mu_k)$ denote the output value $y^{(R)}$ specified in the conclusion of the rule with the fuzzy sets $\mu_1, \ldots, \mu_k$ in the antecedent.

Then we obtain

$$\begin{aligned}
\frac{\partial F_{\mathcal{R}}}{\partial \tilde{x}_i}(\tilde{x}_1, \ldots, \tilde{x}_k) \;=\; & \frac{d\,\mu_{\text{left}(\tilde{x}_i,i)}(\tilde{x}_i)}{d\,\tilde{x}_i} \\
& \cdot \sum_{c \in \{\text{left, right}\}} y\left(\mu_{c(\tilde{x}_1,1)}, \ldots, \mu_{\text{left}(\tilde{x}_i,i)}, \ldots, \mu_{c(\tilde{x}_k,k)}\right) \\
& \qquad \cdot \prod_{s=1, s\neq i}^{k} \mu_{c(\tilde{x}_s,s)}(\tilde{x}_s) \\
& + \frac{d\,\mu_{\text{right}(\tilde{x}_i,i)}(\tilde{x}_i)}{d\,\tilde{x}_i} \\
& \cdot \sum_{c \in \{\text{left, right}\}} y\left(\mu_{c(\tilde{x}_1,1)}, \ldots, \mu_{\text{right}(\tilde{x}_i,i)}, \ldots, \mu_{c(\tilde{x}_k,k)}\right) \\
& \qquad \cdot \prod_{s=1, s\neq i}^{k} \mu_{c(\tilde{x}_s,s)}(\tilde{x}_s). \qquad (10)
\end{aligned}$$

Since the values $\tilde{x_i}$ are in any case transformed values, we can even assume without loss of generality that neighbouring triangular fuzzy sets in our uniform fuzzy partitions have a distance of one (between the points where

they reach the membership degree one). In this case the derivatives

$$\frac{d\,\mu_{\mathrm{left}\tilde{x}_i,i)}(\tilde{x}_i)}{d\,\tilde{x}_i} \quad \text{and} \quad \frac{d\,\mu_{\mathrm{right}\tilde{x}_i,i)}(\tilde{x}_i)}{d\,\tilde{x}_i}$$

become -1 and 1, respectively, so that (10) simplifies to

$$\begin{aligned}\frac{\partial F_{\mathcal{R}}}{\partial \tilde{x}_i}(\tilde{x}_1,\ldots,\tilde{x}_k) \;=\; & -\sum_{c\in\{\text{left, right}\}} y\left(\mu_{c(\tilde{x}_1,1)},\ldots,\mu_{\text{left}(\tilde{x}_i,i)},\ldots,\mu_{c(\tilde{x}_k,k)}\right) \\ & \qquad \cdot \prod_{s=1,s\neq i}^{k} \mu_{c(\tilde{x}_s,s)}(\tilde{x}_s) \\ & + \sum_{c\in\{\text{left, right}\}} y\left(\mu_{c(\tilde{x}_1,1)},\ldots,\mu_{\text{right}(\tilde{x}_i,i)},\ldots,\mu_{c(\tilde{x}_k,k)}\right) \\ & \qquad \cdot \prod_{s=1,s\neq i}^{k} \mu_{c(\tilde{x}_s,s)}(\tilde{x}_s). \qquad (11)\end{aligned}$$

We still have to consider the above mentioned special cases where $\tilde{x}_i$ yields only for one fuzzy set a non-zero membership degree – in this case the membership degree to this fuzzy set must be 1. If $\tilde{x}_i$ is left or right of the left-most or right-most fuzzy set, then the derivative is zero. If $\tilde{x}_i$ hits exactly the unique point of a triangular membership function where it yields membership degree 1, then the derivative is not defined. In this case, we set the derivative to zero, too.

Let us choose sigmoidal functions of the form

$$g(x) \;=\; \frac{1}{1+e^{-b(x-a)}}$$

with parameters a and b for the transformations t_i. It is easy to verify that

$$\begin{aligned}\frac{\partial g}{\partial a}(x) &= -g(x)\cdot(1-g(x))\cdot b \\ \frac{\partial g}{\partial b}(x) &= g(x)\cdot(1-g(x))\cdot(x-a)\end{aligned}$$

holds.

Let us consider a concrete example of a function with two input variables

X_1 and X_2. In this case (11) simplifies to

$$\begin{aligned}
\frac{\partial F_{\mathcal{R}}}{\partial \tilde{x}_1}(\tilde{x}_1, \tilde{x}_2) &= -\mu_{\text{left}(\tilde{x}_2,2)}(\tilde{x}_2) \cdot y\left(\mu_{\text{left}(\tilde{x}_1,1)}, \mu_{\text{left}(\tilde{x}_2,2)}\right) \\
&\quad -\mu_{\text{right}(\tilde{x}_2,2)}(\tilde{x}_2) \cdot y\left(\mu_{\text{left}(\tilde{x}_1,1)}, \mu_{\text{right}(\tilde{x}_2,2)}\right) \\
&\quad +\mu_{\text{left}(\tilde{x}_2,2)}(\tilde{x}_2) \cdot y\left(\mu_{\text{right}(\tilde{x}_1,1)}, \mu_{\text{left}(\tilde{x}_2,2)}\right) \\
&\quad +\mu_{\text{right}(\tilde{x}_2,2)}(\tilde{x}_2) \cdot y\left(\mu_{\text{right}(\tilde{x}_1,1)}, \mu_{\text{right}(\tilde{x}_2,2)}\right) \\
\frac{\partial F_{\mathcal{R}}}{\partial \tilde{x}_2}(\tilde{x}_1, \tilde{x}_2) &= -\mu_{\text{left}(\tilde{x}_1,1)}(\tilde{x}_1) \cdot y\left(\mu_{\text{left}(\tilde{x}_1,1)}, \mu_{\text{left}(\tilde{x}_2,2)}\right) \\
&\quad +\mu_{\text{left}(\tilde{x}_1,1)}(\tilde{x}_1) \cdot y\left(\mu_{\text{left}(\tilde{x}_1,1)}, \mu_{\text{right}(\tilde{x}_2,2)}\right) \\
&\quad -\mu_{\text{right}(\tilde{x}_1,1)}(\tilde{x}_1) \cdot y\left(\mu_{\text{right}(\tilde{x}_1,1)}, \mu_{\text{left}(\tilde{x}_2,2)}\right) \\
&\quad +\mu_{\text{right}(\tilde{x}_1,1)}(\tilde{x}_1) \cdot y\left(\mu_{\text{right}(\tilde{x}_1,1)}, \mu_{\text{right}(\tilde{x}_2,2)}\right).
\end{aligned}$$

We consider the function

$$y \;=\; f(x_1, x_2) \;=\; x_2^3 \cdot \ln(x_1 + 2). \tag{12}$$

We sample this function on a regular grid of 17×17 points in the rectangle $[0, 4] \times [0, 4]$ and use these 289 points as the training set. We use five fuzzy sets for each of the two input domains so that we have 25 rules all together. Table 1 shows the mean square error after up to 500 iteration steps. The first column of this table indicates the number of iteration steps that were carried out, the second column contains the mean square error after the linear regression step for the output value has been completed and the last column contains the mean square error after the gradient descend method has been applied to improve the parameters of the sigmoidal transformations. We have chosen a very small learning rate of $\eta = 0.001$ in order to avoid overshoots.

The lower columns of table 1 show almost no improvements for the gradient descend method. Although the improvements in latter steps tend to be smaller for the gradient descend step than for the linear regression step, the gradient technique still contributes significantly to the improvements. The difference in the numbers is caused by the fact the we do not show the results after each iteration step, but after a greater number of iteration steps. So the first column always indicates the error after the corresponding number of iteration steps, whereas the second number shows the error after applying the gradient technique directly after the corresponding regression step. Looking at the first number in table 1 showing of the fuzzy system after applying linear regression only, we can see the significant improvement of the system after 500 iteration steps. Note that the linear regression will not lead to further improvements, when the fuzzy sets (or the transformations determining the fuzzy sets) are not changed.

	mean square error	
step	regression	gradient descent
1	34.885	28.227
2	25.549	23.241
3	22.239	21.045
4	20.500	19.656
5	19.291	18.544
6	18.272	15.655
7	13.876	12.096
8	11.173	10.174
9	9.664	9.095
10	8.789	8.432
20	6.257	5.778
30	2.806	2.761
40	2.318	2.299
50	2.096	2.084
60	1.959	1.951
70	1.867	1.861
80	1.802	1.798
90	1.755	1.751
100	1.702	1.698
200	1.485	1.485
300	1.442	1.442
400	1.431	1.431
500	1.425	1.425

Table 1: Reduction of the mean square error.

	a	b
X_1	1.773	1.106
X_2	2.287	1.189

Table 2: The final parameters of the transformations.

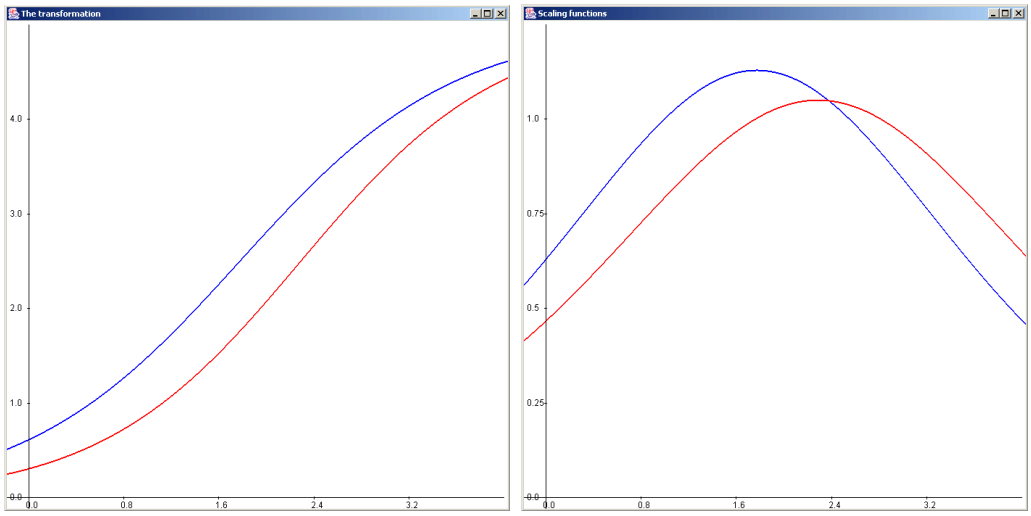

Figure 6: The transformations (left) and the scaling functions (right).

The optimized parameters of the transformations are shown in table 2. Initially the parameter a was set to $a = 2$ – the midpoint of the range of the sampled function – and the parameter b as well to $b = 2$ for both input variables. When we have no further information where the function to be approximated has its strongest changes, placing the best approximation quality somewhere in the middle of the ranges of the variables seems to be a reasonable starting point – better than concentrating somewhere on the boundary of the domain. Our rule of thumb is also that we increase the parameter b with the extension of the interval from which the data come. Another reasonable heuristic strategy would be to choose the mean value or median of the data of the corresponding variable for a and the deviation for b.

The transformations and the corresponding scaling functions (i.e. the derivatives of the scaling functions) are plotted in figure 6.

The final result for the parameters of the transformation functions after the alternating optimization has been carried out, fits perfectly to the interpretation of the transformation functions in the sense of scaling that we have explained in section 3. The variable X_1 occurs in the function (12) within the logarithm. For larger values the logarithm is very flat, almost linear or can even be considered as nearly constant on a larger interval for greater values. Therefore, in order to increase the accuracy of our approximation, it is important to concentrate on the highly non-linear part of the logarithm. Therefore, the value a for the sigmoidal transformation of X_1 is biased to

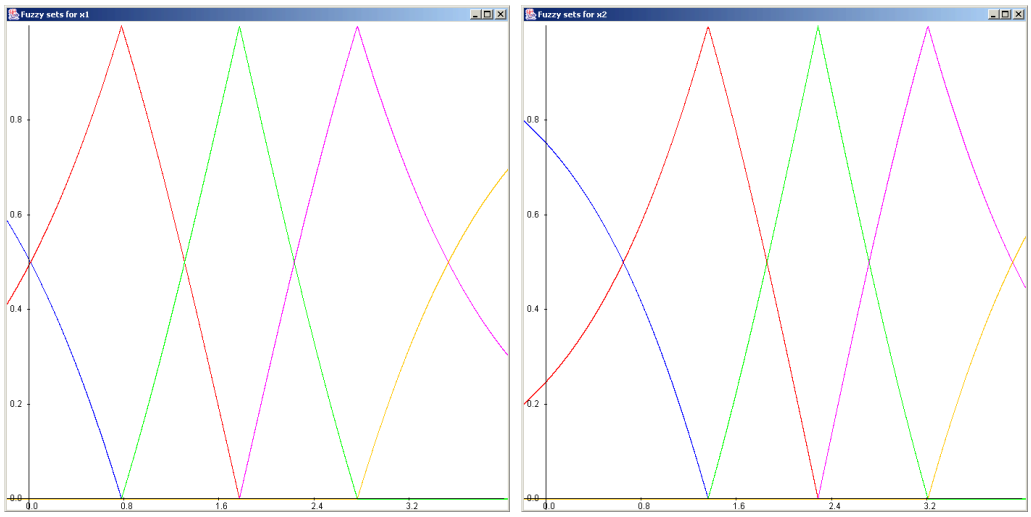

Figure 7: The induced fuzzy sets for the variables x_1 (left) and x_2 (right).

output table	fuzzy set no. for x_2				
fuzzy set no. for x_1	1	2	3	4	5
1	-0.397	0.787	4.238	11.701	34.146
2	-1.084	2.151	11.589	31.993	93.366
3	-1.385	2.747	14.807	40.878	119.296
4	-1.612	3.199	17.230	47.567	138.815
5	-1.955	3.880	20.899	57.696	168.374

Table 3: The output values for the rules.

the left side of the range. The second variable has a cubic influence on our example function. The cubic function is very flat and almost linear for small values and deviates stronger from a linear behaviour for larger values. Therefore, the bias of the transformation of the second variable tends more to the right side of the range. This can also been seen in figure 7, when we look at the resulting fuzzy sets induced by the transformations. It is quite remarkable that the seemingly small change of the fuzzy sets enables the system to reduce the mean square error from almost 35 to less than 1.5.

The largest (absolute) error of 3.865 occurs at the very extreme data point $(4, 4)$ where the correct output is 114.673. The output values for the fuzzy rules with the fuzzy sets indicated in figure 7 numbered from left to right are show in table 3.

6 Conclusions

We have proposed a very simple fuzzy system with a small number of parameters that can be trained efficiently and can also be interpreted easily. The algorithm we have described is suited for regression problems in general, where an understanding of the regression function in terms of fuzzy rules is desirable. The regression problem can either arise from a control problem where sample data of the control function are available or in general terms where a continuous variable has to be predicted on the basis of other continuous variables. Depending on the regression problem suitable parametric transformations have to be chosen. As mentioned before, for many control problems, sigmoidal transformations seem to be suitable. The unimodal derivative of a sigmoidal transformation will (hopefully) be adjusted in such a way that its maximum will be placed in that region where highest precision is needed. However, for other regression problems, transformations with unimodal derivatives might not be sufficient. In this case one might think of a sum of a few transformations of the form

$$t(x) \;=\; a(x + \sin(bx)) \qquad (x > 0).$$

The combination of linear regression and a gradient descent method that we propose has the advantage that the optimization process is strictly directed. Of course, other techniques like evolutionary algorithms could also be applied. But the usually lead to much higher computational costs. Especially, when we can keep the number of parameters of our transformations small, our approach is quite efficient, even if we consider a fuzzy system with a higher number of fuzzy sets. Increasing the number of fuzzy sets and adjusting them directly would lead to an increase of parameters to be optimized. In our case, the only additional parameters we have to consider, when we increase the number of fuzzy sets, are the rule outputs. But these are computed very efficiently by linear regression.

Further research will also concentrate on fuzzy classification systems. In this case the output of the rules are not real values, but discrete classes. Regression and gradient descent techniques are not well suited for classification problems, since a typical error function will be the misclassification rate. But we could still apply the alternating optimization scheme in the following way. In order to determine the output values (classes instead of real values) for the rules, we simply determine the majority class for which the corresponding rule fires with the highest membership degree. The parameters of transformations cannot be learned by a regression technique, but we could apply entropy minimization techniques similar as in decision trees with continuous attributes in order to adapt the transformations.

Acknowledgements: The author is highly indebted to the anonymous reviewers for their valuable comments.

References

[1] J. Casillas, O. Cordón, F. Herrera, L. Magdalena (eds.): Interpretability Issues in Fuzzy Modelling. Springer, Berlin (2003).

[2] F. Höppner, F. Klawonn, R. Kruse, T. Runkler: Fuzzy Cluster Analysis. Wiley, Chichester (1999).

[3] V. Kecman, B.-M. Pfeiffer: Exploiting the Structural Equivalence of Learning Fuzzy Systems and Radial Basis Function Neural Networks. Proc. Second European Congress on Intelligent Techniques and Soft Computing (EUFIT'94), Aachen (1994), 58-66.

[4] F. Klawonn: Fuzzy Sets and Vague Environments. Fuzzy Sets and Systems 66 (1994), 207-221.

[5] F. Klawonn, J. Gebhardt, R. Kruse: Fuzzy Control on the Basis of Equality Relations – with an Example from Idle Speed Control. IEEE Transactions on Fuzzy Systems 3 (1995), 336-350.

[6] F. Klawonn, R. Kruse: Techniques and Applications of Control Systems Based on Knowledge-Based Interpolation. In: C.T. Leondes (ed.): Fuzzy Theory Systems: Techniques and Applications. Academic Press, San Diego (1999), 431-460.

[7] D. Nauck, F. Klawonn, R. Kruse: Foundations of Neuro-Fuzzy Systems. Wiley, Chichester (1999).

[8] W. Pedrycz (ed.): Fuzzy Evolutionary Computation. Kluwer, Boston (1997).

[9] L.A. Zadeh, Fuzzy Sets: Information and Control 8 (1965), 338-353.