PERGAMON

Annual Reviews in Control 27 (2003) 73–85

Annual Reviews in Control

www.elsevier.com/locate/arcontrol

Review

Neuro-fuzzy methods for nonlinear system identification☆

Robert Babuška*, Henk Verbruggen

Delft Center for Systems and Control, Delft University of Technology, Mekelweg 2, 2628 CD Delft, The Netherlands

Abstract

Most processes in industry are characterized by nonlinear and time-varying behavior. Nonlinear system identification is becoming an important tool which can be used to improve control performance and achieve robust fault-tolerant behavior. Among the different nonlinear identification techniques, methods based on neuro-fuzzy models are gradually becoming established not only in the academia but also in industrial applications. Neuro-fuzzy modeling can be regarded as a gray-box technique on the boundary between neural networks and qualitative fuzzy models. The tools for building neuro-fuzzy models are based on combinations of algorithms from the fields of neural networks, pattern recognition and regression analysis. In this paper, an overview of neuro-fuzzy modeling methods for nonlinear system identification is given, with an emphasis on the tradeoff between accuracy and interpretability.
© 2003 Published by Elsevier Science Ltd.

Keywords: Neuro-fuzzy systems; Nonlinear identification; ANFIS network; Takagi–Sugeno fuzzy system; Clustering

1. Introduction

The design of control systems is currently driven by a large number of requirements posed by increasing competition, environmental requirements, energy and material costs and the demand for robust, fault-tolerant systems. These considerations introduce extra needs for effective process modeling techniques. Many systems are not amenable to conventional modeling approaches due to the lack of precise, formal knowledge about the system, due to strongly nonlinear behavior, high degree of uncertainty, or time-varying characteristics.

Neuro-fuzzy modeling has been recognized as a powerful tool which can facilitate the effective development of models by combining information from different sources, such as empirical models, heuristics and data. Neuro-fuzzy models describe systems by means of fuzzy *if–then* rules represented in a network structure, to which learning algorithms known from the area of artificial neural networks can be applied. Thanks to this structure, neuro-fuzzy models are to a certain degree transparent to interpretation and analysis, i.e. they can be better used to explain solutions to users than completely black-box models such as neural networks.

The paper is organized as follows: an overview of fuzzy systems and neural networks is first given in Section 2. Neuro-fuzzy systems are then addressed in Section 3. Various methods for parameter estimation are presented in Section 4 and two numerical examples can be found in Section 5: approximation of a static function and dynamic modeling of the pH process. Section 6 concludes the paper.

2. Fuzzy systems and neural networks

Both neural networks and fuzzy systems are motivated by imitating human reasoning processes. In fuzzy systems, relationships are represented explicitly in the form of if–then rules. In neural networks, the relations are not explicitly given, but are 'coded' in the network and its parameters. In contrast to knowledge-based techniques, no explicit knowledge is needed for the application of neural networks.

Neuro-fuzzy systems combine the semantic transparency of rule-based fuzzy systems with the learning capability of neural networks. This section gives the background on nonlinear input–output modeling, fuzzy systems and neural networks, which is essential for understanding the rest of this article.

☆ Partly based on: Babuška, R. (2000). Neuro-fuzzy methods for modeling and identification. In A. Abraham, L. C. Jain, & J. Kacprzyk (Eds.), *Recent advances in intelligent paradigms and applications* (pp. 161–186). Heidelberg: Springer-Verlag. Reprinted with permission.

* Corresponding author. Tel.: +31-15-27-85117; fax: +31-15-27-86679.
E-mail address: r.babuska@dcsc.tudelft.nl (R. Babuška).

1367-5788/$ – see front matter © 2003 Published by Elsevier Science Ltd.
doi:10.1016/S1367-5788(03)00009-9

2.1. Nonlinear system identification

A wide class of nonlinear dynamic systems with an input u and an output y can be described in discrete time by the NARX (nonlinear autoregressive with exogenous input) input–output model:

$$y(k+1) = f(\boldsymbol{x}(k)), \tag{1}$$

where $y(k+1)$ denotes the output predicted at the future time instant $k+1$ and $\boldsymbol{x}(k)$ is the *regressor* vector, consisting of a finite number of past inputs and outputs:

$$\boldsymbol{x}(k) = \begin{bmatrix} y(k) \\ \vdots \\ y(k-n_y+1) \\ u(k) \\ \vdots \\ u(k-n_u+1) \end{bmatrix}. \tag{2}$$

The dynamic order of the system is represented by the number of lags n_u and n_y. Although for simplicity stated with a scalar input and output, the NARX model can also be used for multivariable systems. In that case, however, the number of regressors usually becomes large and one may prefer the nonlinear state-space description:

$$\boldsymbol{\xi}(k+1) = g(\boldsymbol{\xi}(k), \boldsymbol{u}(k)), \qquad \boldsymbol{y}(k) = h(\boldsymbol{\xi}(k)). \tag{3}$$

The problem of nonlinear system identification is to infer the unknown function f in (1) or the functions g and h in (3) from some sampled data sequences $\{(\boldsymbol{u}(k), \boldsymbol{y}(k)) | k = 1, 2, \ldots, N\}$.

In black-box modeling, these functions are approximated by some general function approximators such as neural networks, neuro-fuzzy systems, splines, interpolated look-up tables, etc. If the aim of modeling is only to obtain an accurate predictor for y, there is not much difference between these models, as they all can approximate smooth nonlinear systems arbitrarily well. Often, however, besides accurate predictions, one wants to have a model that can be used to learn something about the underlying system and analyze its properties. From this point of view, fuzzy and neuro-fuzzy systems are more transparent than most other black-box techniques.

2.2. Fuzzy models

A mathematical model which in some way uses fuzzy sets is called a *fuzzy model*. In system identification, rule-based fuzzy models are usually applied. In these models, the relationships between variables are represented by means of if–then rules with imprecise (ambiguous) predicates, such as:

If *heating is high* then *temperature increase is fast*.

This rule defines in a rather qualitative way the relationship between the heating and the temperature in a room, for instance. To make such a model operational, the meaning of the terms ‘high’ and ‘fast’ must be defined more precisely. This is done by using fuzzy sets, i.e. sets where the membership is changing gradually rather than in an abrupt way. Fuzzy sets are defined through their membership functions (denoted by μ) which map the elements of the considered universe to the unit interval $[0, 1]$. The extreme values 0 and 1 denote complete membership and non-membership, respectively, while a degree between 0 and 1 means partial membership in the fuzzy set. Depending on the structure of the if–then rules, two main types of fuzzy models can be distinguished: the Mamdani (or linguistic) model and the Takagi–Sugeno model.

2.2.1. Mamdani model

In this model, the antecedent (if-part of the rule) and the consequent (then-part of the rule) are fuzzy propositions:

$$\mathcal{R}_i: \text{ If } x \text{ is } A_i \text{ then } y \text{ is } B_i, \quad i = 1, 2, \ldots, K. \tag{4}$$

Here A_i and B_i are the antecedent and consequent linguistic terms (such as ‘small,’ ‘large,’ etc.), represented by fuzzy sets, and K is the number of rules in the model. The linguistic fuzzy model is useful for representing qualitative knowledge, illustrated in the following example.

Example 1. Consider a qualitative description of the relationship between the oxygen supply to a gas burner (x) and its heating power (y):

$\mathcal{R}_1$: If O_2 flow rate is *Low* then power is *Low*.
$\mathcal{R}_2$: If O_2 flow rate is *OK* then power is *High*.
$\mathcal{R}_3$: If O_2 flow rate is *High* then power is *Low*.

The meaning of the linguistic terms $\{Low, OK, High\}$ and $\{Low, High\}$ is defined by membership functions such as the ones depicted in Fig. 1. Membership functions can be defined by the model developer based on prior knowledge or by using data (in this example, the membership functions and their domains are selected quite arbitrarily).

The meaning of the linguistic terms is, of course, not universally given. In this example, the definition of the fuzzy set OK, for instance, may depend on the flow rate of the fuel gas, the type of burner, etc. When input–output data of the system under study are available, the membership functions can be constructed or adjusted automatically, as discussed later on. Note, however, that the qualitative relationship given by the rules is usually expected to be valid for a range of conditions.

2.2.2. Takagi–Sugeno model

The Mamdani model is typically used in knowledge-based (expert) systems. In data-driven identification, the model due to Takagi and Sugeno has become popular. In this model, the antecedent is defined in the same way as above, while the consequent is an affine linear function of the input variables:

$$\mathcal{R}_i: \text{ If } \boldsymbol{x} \text{ is } A_i \text{ then } y_i = \boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{x} + b_i, \tag{5}$$

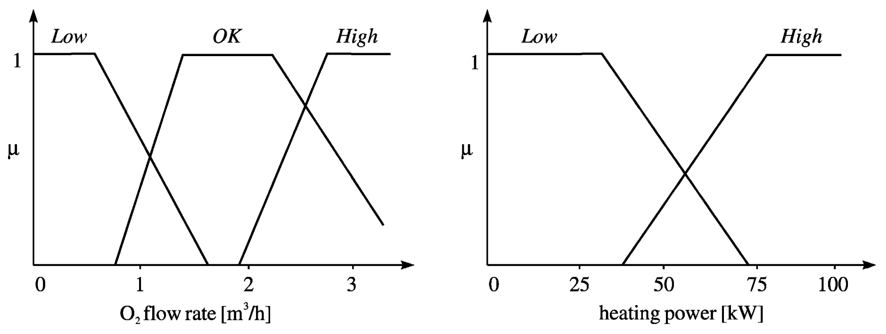


Fig. 1. Membership functions for the Mamdani model of Example 1.

where $\boldsymbol{a}_i$ is the consequent parameter vector, b_i is a scalar offset and $i = 1, \ldots, K$. This model combines a linguistic description with standard functional regression: the antecedents describe fuzzy regions in the input space in which the consequent functions are valid. The output y is computed by taking the weighted average of the individual rules' contributions:

$$y = \frac{\sum_{i=1}^{K} \beta_i(\boldsymbol{x}) y_i}{\sum_{i=1}^{K} \beta_i(\boldsymbol{x})} = \frac{\sum_{i=1}^{K} \beta_i(\boldsymbol{x})(\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{x} + b_i)}{\sum_{i=1}^{K} \beta_i(\boldsymbol{x})}, \tag{6}$$

where $\beta_i(\boldsymbol{x})$ is the *degree of fulfillment* of the ith rule. For the rule (5), $\beta_i(\boldsymbol{x}) = \mu_{A_i}(\boldsymbol{x})$, but it can also be a more complicated expression, as shown later on. The antecedent fuzzy sets are usually defined to describe distinct, partly overlapping regions in the input space. The parameters $\boldsymbol{a}_i$ are then (approximate) local linear models of the considered nonlinear system. The TS model can thus be regarded as a smooth piece-wise linear approximation of a nonlinear function or a parameter-scheduling model. Note that the antecedent and consequent variables may be different.

Example 2. Consider a static characteristic of an actuator with a dead-zone and a non-symmetrical response for positive and negative inputs. Such a system can conveniently be represented by a TS model with three rules each covering a subset of the operating domain that can be approximated by a local linear model, see Fig. 2. The corresponding rules

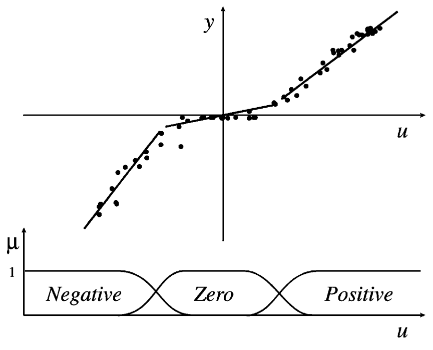


Fig. 2. A Takagi–Sugeno fuzzy model as a piece-wise linear approximation of a nonlinear system.

and the input–output equation are:

$$\begin{aligned} \mathcal{R}_1 &: \text{ If } u \text{ is } \mathit{Negative} \text{ then } y_1 = a_1 u + b_1 \\ \mathcal{R}_2 &: \text{ If } u \text{ is } \mathit{Zero} \text{ then } y_2 = a_2 u + b_2 \\ \mathcal{R}_3 &: \text{ If } u \text{ is } \mathit{Positive} \text{ then } y_3 = a_3 u + b_3 \end{aligned}$$

and

$$y = \frac{\mu_{Neg}(u) y_1 + \mu_{Zero}(u) y_2 + \mu_{Pos}(u) y_3}{\mu_{Neg}(u) + \mu_{Zero}(u) + \mu_{Pos}(u)}.$$

As the consequent parameters are first-order polynomials in the input variables, model (5) is in the literature also called the *first-order TS model*. This is in order to distinguish it from the zero-order TS model whose consequents are simply constants (zero-order polynomials):

$$\mathcal{R}_i : \text{ If } \boldsymbol{x} \text{ is } A_i \text{ then } y_i = b_i, \quad i = 1, 2, \ldots, K. \tag{7}$$

For this model, the input–output Eq. (6) reduces to:

$$y = \frac{\sum_{i=1}^{K} \beta_i(\boldsymbol{x}) b_i}{\sum_{i=1}^{K} \beta_i(\boldsymbol{x})}. \tag{8}$$

The above model can also be obtained as a special case of the Mamdani system (4) in which the consequent fuzzy sets degenerate to singletons (real numbers):

$$\mu_{B_i}(y) = \begin{cases} 1, & \text{if } y = b_i, \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

2.2.3. *Fuzzy logic operators*

In fuzzy systems with multiple inputs, the antecedent proposition is usually represented as a combination of terms with univariate membership functions, by using logic operators 'and' (conjunction), 'or' (disjunction) and 'not' (complement). In fuzzy set theory, several families of operators have been introduced for these logical connectives. Table 1 shows the two most common ones.

Table 1
Common fuzzy logic operators

	A and B	A or B	not A
Zadeh	$\min(\mu_A, \mu_B)$	$\max(\mu_A, \mu_B)$	$1 - \mu_A$
Probabilistic	$\mu_A \cdot \mu_B$	$\mu_A + \mu_B - \mu_A \cdot \mu_B$	$1 - \mu_A$

As an example, consider the *conjunctive form* of the antecedent, which is given by:

$$\mathcal{R}_i: \text{ If } x_1 \text{ is } A_{i1} \text{ and} \ldots \text{and } x_p \text{ is } A_{ip} \text{ then } y_i = \boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{x} + b_i \tag{10}$$

with the degree of fulfillment:

$$\beta_i(\boldsymbol{x}) = \min(\mu_{A_{i1}}(x_1), \ldots, \mu_{A_{ip}}(x_p)) \quad \text{or}$$
$$\beta_i(\boldsymbol{x}) = \mu_{A_{i1}}(x_1) \cdot \mu_{A_{i2}}(x_2) \cdots \mu A_{ip}(x_p) \tag{11}$$

for the minimum and product conjunction operators, respectively. The complete set of rules (10) divide the input domain into a lattice of overlapping axis-parallel hyperboxes. Each of these hyperboxes is a Cartesian product intersection of the corresponding univariate fuzzy sets.

2.2.4. Dynamic fuzzy models

In the modeling of dynamic systems, fuzzy models are used to parameterize of the nonlinear functions *f* in (1) or *g* and *h* in (3). Consider, for instance, the TS NARX model:

$$\mathcal{R}_i: \text{ If } \boldsymbol{x}(k) \text{ is } A_i \text{ then } y_i(k+1) = \sum_{j=1}^{n_y} a_{ij} y(k-j+1) + \sum_{j=1}^{n_u} b_{ij} u(k-j+1) + c_i, \tag{12}$$

where the antecedent regressor $\boldsymbol{x}(k)$ is generally given by (2), but it may of course contain only some of the past inputs and outputs or even other variables than *u* and *y*. Similarly, state-space models can be represented in the TS framework by:

$$\mathcal{R}_i: \text{ If } \boldsymbol{\xi}(k) \text{ is } A_i \text{ and } \boldsymbol{u}(k) \text{ is } B_i \text{ then } \begin{cases} \boldsymbol{\xi}_i(k+1) = \boldsymbol{\Phi}_i\boldsymbol{\xi}(k) + \boldsymbol{\Gamma}_i\boldsymbol{u}(k) + \boldsymbol{a}_i \\ \boldsymbol{y}_i(k) = \boldsymbol{C}_i\boldsymbol{\xi}(k) + \boldsymbol{c}_i. \end{cases} \tag{13}$$

An advantage of the state-space modeling approach is that the structure of the model can easily be related to the physical structure of the real system, and, consequently, the model parameters are physically relevant. This is not necessarily the case with input–output models. In addition, the dimension of the regression problem in state-space modeling is often smaller than with input–output models.

2.3. Artificial neural networks

Artificial neural networks (ANNs), originally inspired by the functionality of biological neural networks can learn complex functional relations by generalizing from a limited amount of training data. Neural networks can thus serve as black-box models of nonlinear, multivariable static and dynamic systems and can be trained by using input–output data observed on the system. The most common ANNs consist of several layers of simple processing elements called neurons, interconnections among them and weights assigned to these interconnections. The information relevant to the input–output mapping of the net is stored in the weights.

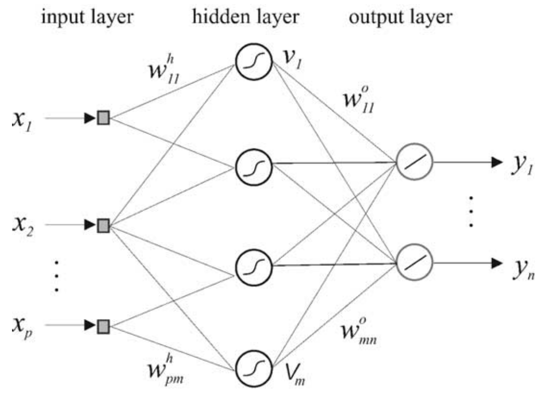


Fig. 3. A feedforward with one hidden layer.

2.3.1. Multi-layer neural network

A feedforward multi-layer neural network (MNN) has one input layer, one output layer and an number of hidden layers between them. For illustration purposes, consider a MNN with one hidden layer (Fig. 3).

The input-layer neurons do not perform any computations, they merely distribute the inputs x_i to the weights w_{ij}^{h} of the hidden layer. In the neurons of the hidden layer, first the weighted sum of the inputs is computed:

$$z_j = \sum_{i=1}^{p} w_{ij}^{\mathrm{h}} x_i = (\boldsymbol{w}_j^{\mathrm{h}})^{\mathrm{T}}\boldsymbol{x}, \quad j = 1, \ldots, m. \tag{14}$$

It is then passed through a nonlinear *activation* function, such as the tangent hyperbolic:

$$v_j = \frac{1 - \exp(-2z_j)}{1 + \exp(-2z_j)}, \quad j = 1, 2, \ldots, m. \tag{15}$$

Other typical activation functions are the threshold function (hard limiter) and the sigmoidal function. The neurons in the output layer are linear, i.e. only compute the weighted sum of their inputs:

$$y_l = \sum_{j=1}^{h} w_{jl}^{\mathrm{o}} v_j = (\boldsymbol{w}_j^{\mathrm{o}})^{\mathrm{T}}\boldsymbol{x}, \quad l = 1, \ldots, n. \tag{16}$$

Training is the adaptation of weights in a multi-layer network such that the error between the desired output and the network output is minimized. Two steps are distinguished in this procedure:

(1) *Feedforward computation*. From the network inputs x_i, the outputs of the first hidden layer are first computed. Then using these values as inputs to the second hidden layer, the outputs of this layer are computed, etc. Finally, the output of the network is obtained.
(2) *Weight adaptation*. The output of the network is compared to the desired output. The difference of these two values, the error, is then used to adjust the weights first in the output layer, then in the layer before, etc., in order to decrease the error (gradient-descent optimization). This

backward computation is called error backpropagation (Rumelhart, Hinton, & Williams, 1986).

A network with one hidden layer is sufficient for most approximation tasks. More layers can give a better fit, but the training takes longer. Choosing the right number of neurons in the hidden layer is essential for a good result. Too few neurons give a poor fit, while too many neurons result in over-training of the net (poor generalization to unseen data). A compromise is usually sought by trial and error methods.

2.3.2. Dynamic neural networks

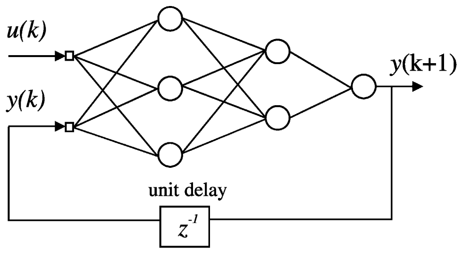
A dynamic network can be realized by using a static feedforward network combined with an external feedback connection. The output of the network is fed back to its input through delay operators z^{-1}. This is in fact a realization of the NARX model (1). Fig. 4a shows an example of a first-order system $y(k+1) = f_{\mathrm{nn}}(y(k), u(k))$.

Another possibility is to use recurrent networks in which neurons are arranged in one or more layers and feedback is introduced either internally in the neurons, to other neurons in the same layer, or to neurons in preceding layers. Examples of these networks are the Elman network (Fig. 4b) or the Hopfield network.

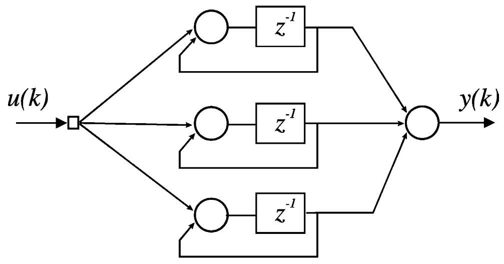
2.3.3. Error backpropagation

Consider for simplicity a MNN with one output. A set of N input–output data pairs $\{(\boldsymbol{x}_k, y_k^*) | k = 1, 2, \ldots, N\}$ is available. We represent this set as a matrix $\boldsymbol{X} \in \mathbb{R}^{N\times p}$, having the input vectors $\boldsymbol{x}_k$ in its rows, and a column vector $\boldsymbol{y}^* \in \mathbb{R}^N$, containing the desired outputs y_k^*:

$$\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]^{\mathrm{T}}, \qquad \boldsymbol{y}^* = [y_1^*, \ldots, y_N^*]^{\mathrm{T}}. \tag{17}$$



(a) neural network NARX model



(b) Elman network

Fig. 4. Neural network models of dynamic systems.

The difference between the desired output y^* and the output of the network y is called the error. This error is used to adjust the weights in the net via the minimization of the following cost function:

$$J = \frac{1}{2}\sum_{k=1}^{N} e_k^2 \quad \text{with} \quad e_k = y_k^* - y_k.$$

Note that the network's output y is nonlinear in the weights $\boldsymbol{w}$ (for notational convenience, all the weights are lumped in a single vector $\boldsymbol{w}$). The training of a MNN is thus a *nonlinear optimization problem* to which various methods can be applied:

- Error backpropagation (first-order gradient).
- Newton, Levenberg–Marquardt methods (second-order gradient).
- Genetic algorithms and many others techniques.

First-order gradient methods are based on the following general update rule for the weights:

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \alpha(n)\nabla J(\boldsymbol{w}(n)), \tag{18}$$

where $\boldsymbol{w}(n)$ is the weight vector at iteration n, $\alpha(n)$ is a (variable) learning rate (a parameter) and $\nabla J(\boldsymbol{w})$ is the Jacobian of the network:

$$\nabla J(\boldsymbol{w}) = \left[\frac{\partial J(\boldsymbol{w})}{\partial w_1}, \frac{\partial J(\boldsymbol{w})}{\partial w_2}, \ldots, \frac{\partial J(\boldsymbol{w})}{\partial w_M}\right]^{\mathrm{T}}. \tag{19}$$

The nonlinear optimization problem is thus solved by using the first term of its Taylor series expansion (the gradient). Second-order gradient methods make use of the second term as well:

$$J(\boldsymbol{w}) \approx J(\boldsymbol{w}_0) + \nabla J(\boldsymbol{w}_0)^{\mathrm{T}}(\boldsymbol{w} - \boldsymbol{w}_0) + \tfrac{1}{2}(\boldsymbol{w} - \boldsymbol{w}_0)^{\mathrm{T}}\boldsymbol{H}(\boldsymbol{w}_0)(\boldsymbol{w} - \boldsymbol{w}_0),$$

where $\boldsymbol{H}(\boldsymbol{w}_0)$ is the Hessian. Second-order methods are usually more effective than first-order ones.

2.3.4. Radial basis function network

The radial basis function (RBF) network is a two-layer network with an architecture depicted in Fig. 5. This network is represented by the following function:

$$y = f(\boldsymbol{x}) = \sum_{i=1}^{m} w_i \phi_i(\boldsymbol{x}), \tag{20}$$

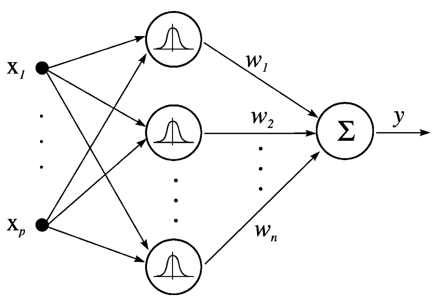


Fig. 5. Radial basis function network.

where the usual choice for the basis functions $\phi_i(\boldsymbol{x})$ is the Gaussian function:

$$\phi_i(\boldsymbol{x}) = \exp\left(-\frac{||\boldsymbol{x}-\boldsymbol{c}_i||^2}{2\sigma_i^2}\right).$$

Note that adjustable weights are only present in the output layer. The connections from the input layer to the hidden layer are fixed to unit weights. The free parameters of RBF networks are the output weights w_i and the parameters of the basis functions (centers $\boldsymbol{c}_i$ and radii σ_i). Since the network's output (20) is linear in the weights w_i, these weights can be estimated by least-squares methods. For each data point $\boldsymbol{x}_k$, first the outputs of the neurons are computed:

$$v_{ki} = \phi_i(\boldsymbol{x}_k)$$

and put in the matrix $\boldsymbol{V} = [v_{ki}]$. Introducing the weight vector $\boldsymbol{w} = [w_1, w_2, \ldots, w_m]$, we can write the following matrix equation for the whole data set:

$$\boldsymbol{y}^* = V\boldsymbol{w}.$$

The least-square estimate of the weights $\boldsymbol{w}$ that minimize the network error $\boldsymbol{e} = \boldsymbol{y}^* - \boldsymbol{y}$ is:

$$\boldsymbol{w} = [\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V}]^{-1}\boldsymbol{V}^{\mathrm{T}}\boldsymbol{y}^*. \tag{21}$$

The adaptation of the RBF parameters $\boldsymbol{c}_i$ and σ_i is a nonlinear optimization problem that can be solved by the gradient-descent method (18).

3. Neuro-fuzzy modeling

At the computational level, a fuzzy system can be seen as a layered structure (network), similar to artificial neural networks of the RBF-type (Jang & Sun, 1993). In order to optimize parameters in a fuzzy system, gradient-descent training algorithms known from the area of neural networks can be employed. Hence, this approach is usually referred to as neuro-fuzzy modeling (Brown & Harris, 1994; Jang, 1993; Jang, Sun, & Mizutani, 1997).

Consider first a simple example of a zero-order TS fuzzy model with the following two rules:

If x_1 is A_{11} and x_2 is A_{21} then $y = b_1$
If x_1 is A_{12} and x_2 is A_{22} then $y = b_2$.

Fig. 6 shows a network representation of these two rules. The nodes in the first layer compute the membership degree of the inputs in the antecedent fuzzy sets. The product nodes Π in the second layer represent the antecedent connective (here the 'and' operator). The normalization node N and the summation node Σ realize the fuzzy-mean operator (6). This system is called adaptive neuro-fuzzy inference system (ANFIS; Jang, 1993).

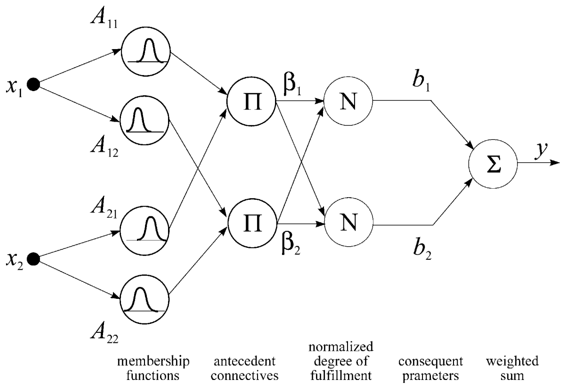


Fig. 6. An example of a zero-order TS fuzzy model with two rules represented as a neuro-fuzzy network.

Typically, smooth antecedent membership functions are used, such as the Gaussian functions:

$$\mu_{A_{ij}}(x_j; c_{ij}, \sigma_{ij}) = \exp\left(-\frac{(x_j - c_{ij})^2}{2\sigma_{ij}^2}\right). \tag{22}$$

The input–output equation of a general zero-order TS model with the conjunctive form antecedent is:

$$y = \sum_{i=1}^{K}\gamma_i(\boldsymbol{x})b_i \tag{23}$$

with

$$\gamma_i(\boldsymbol{x}) = \frac{\prod_{j=1}^{p}\exp(-(x_j - c_{ij})^2/2\sigma_{ij}^2)}{\sum_{i=1}^{K}\prod_{j=1}^{p}\exp(-(x_j - c_{ij})^2/2\sigma_{ij}^2)}.$$

The first-order TS fuzzy model can be represented in a similar fashion. Consider again the example with two rules:

If x_1 is A_{11} and x_2 is A_{21} then $y_1 = a_{11}x_1 + a_{12}x_2 + b_1$
If x_1 is A_{12} and x_2 is A_{22} then $y_2 = a_{21}x_1 + a_{22}x_2 + b_2$

for which the corresponding network is given in Fig. 7.

The input–output equation of this first-order TS model is:

$$y = \sum_{i=1}^{K}\gamma_i(\boldsymbol{x})(\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{x} + b_i) \tag{24}$$

with $\gamma_i(\boldsymbol{x})$ given by (23).

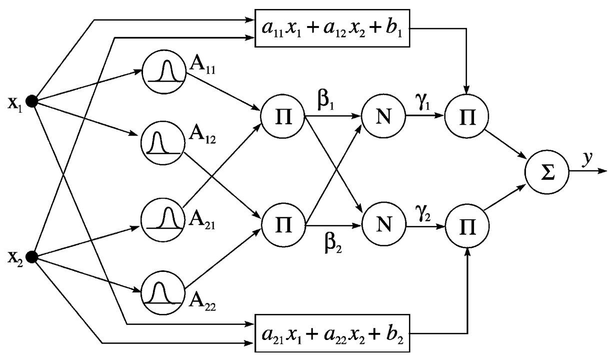


Fig. 7. An example of a first-order TS fuzzy model with two rules represented as a neuro-fuzzy network called ANFIS.

3.1. Constructing neuro-fuzzy networks

Both prior knowledge and process data can be used to construct neuro-fuzzy systems. Prior knowledge can be of a rather approximate nature (qualitative, heuristics). Two main approaches to the integration of knowledge and data can be distinguished:

(1) Expert knowledge is formulated as a collection of if–then rules. In this way, an initial model is created. The parameters of this model (the membership functions, consequent parameters) are then fine-tuned by using process data.
(2) Fuzzy rules (including the associated parameters) are constructed from scratch by using numerical data. In this case, the advantage of using a neuro-fuzzy model is the possibility to interpret the obtained result (which is not possible with truly black-box structures like neural networks). An expert can confront the information stored in the rule base with his own knowledge, can modify the rules, or supply additional ones to extend the validity of the model, etc.

The above techniques can, of course, be combined, depending on the problem at hand.

3.2. Structure and parameters

The two basic steps in system identification are *structure identification* and *parameter estimation*. The choice of the model's structure is very important, as it determines the flexibility of the model in the approximation of (unknown) systems. A model with a rich structure can approximate more complicated functions, but, at the same time, will have worse generalization properties. Good generalization means that a model fitted to one data set will also perform well on another data set from the same process. In neuro-fuzzy models, the structure selection process involves the following main choices:

- *Selection of input variables*. This involves not only the physical inputs u but also the dynamic regressors, defined by the input and output lags, n_y and n_u, respectively. Prior knowledge, insight in the process behavior and the purpose of the modeling exercise are the typical sources of information for the choice of an initial set of possible inputs. Automatic data-driven selection can then be used to compare different structures in terms of some specified performance criteria.
- *Number and type of membership functions, number of rules*. These two structural parameters are mutually related (for more membership functions more rules must be defined) and determine the level of detail, called the granularity, of the model. The purpose of modeling and the amount of available information (knowledge and data) will determine this choice. Automated, methods can be used to add or remove membership functions and rules.

3.3. Gradient-based learning

It is quite straightforward to derive the gradient-descent learning rule for the b_i c_{ij} and σ_{ij} parameters. The procedure is identical to the derivation of the backpropagation formulas for neural networks. Consider the zero-order ANFIS model, given by rules (7). For the consequent parameters b_i, we have the Jacobian (19):

$$\frac{\partial J}{\partial b_i} = \frac{\partial J}{\partial e}\frac{\partial e}{\partial b_i} = -\gamma_i e, \quad i = 1, \ldots, K \tag{25}$$

and the update law:

$$b_i(n+1) = b_i(n) + \alpha(n)\gamma_i e. \tag{26}$$

For the centers and spreads of the Gaussian membership functions (22), we apply the chain rule for differentiation and after some algebra, the following update formulas are obtained:

$$c_{ij}(n+1) = c_{ij}(n) + 2\alpha(n)\gamma_i e \frac{x_j - c_{ij}}{\sigma_{ij}^2}[b_i - y]$$

and

$$\sigma_{ij}(n+1) = \sigma_{ij}(n) + 2\alpha(n)\gamma_i e \frac{(x_j - c_{ij})^2}{\sigma_{ij}^3}[b_i - y].$$

The parameter-update equations for the first-order ANFIS model can be derived in a similar fashion.

4. Hybrid learning techniques

We have already noticed that the output-layer parameters in RBF networks can be estimated by linear least-squares (LS) techniques (21). As LS methods are more effective than the gradient-based update rule (26), hybrid methods are often applied that combine one-shot least-squares estimation of the consequent parameters with iterative gradient-based optimization of the membership functions (Mathworks, 2001).

In terms of error minimization, the choice of a particular least-squares estimation method is not crucial. If, however, the consequent parameters are to be interpreted as local models, for instance, great care must be taken in the choice of the estimation method. The problem is that the ANFIS models, especially the first-order one, tend to be over-parameterized for most approximation problems. This may lead to numerical problems, over-fitting and meaningless parameter estimates. The following example demonstrates this problem.

Example 3. Assume we wish to approximate a second-order polynomial $y = f_s(u) = 3u^2 - 5u + 6$ by a first-order ANFIS model. First we choose two points t_1 and t_2 and define initial triangular membership functions for $t_1 \leq u < t_2$:

$$\mu_{A_1}(u) = \frac{u - t_1}{t_2 - t_1}, \qquad \mu_{A_2} = 1 - \mu_{A_1}. \tag{27}$$

The model consists of two rules:

$$\mathcal{R}_i: \text{ If } u \text{ is } A_i \text{ then } y_i = a_i u + b_i, \quad i = 1, 2.$$

By substituting the membership functions (27) into (6), the output of the TS model is obtained (after some elementary algebra):

$$y = \frac{a_1 - a_2}{t_2 - t_1}u^2 + \frac{t_2a_2 - t_1a_1 + b_1 - b_2}{t_2 - t_1}u + \frac{t_2b_2 - t_1b_1}{t_2 - t_1}.$$

As this is a second-order polynomial in u, our model can perfectly represent the given nonlinear system. However, it has four free parameters (a_1, a_2, b_1 and b_2) while three are sufficient to fit the polynomial—it is thus over-parameterized. This is a very simple example, but the essence of the over-parameterization problem remains the same when approximating complex unknown systems.

4.1. Consequent parameter estimation

To circumvent over-parameterization, the basic least-squares criterion can be combined with additional criteria for local fit, or with constraints on the parameter values. In the following, several techniques are discussed.

4.1.1. Global least-squares estimation

The global least-squares estimation method yields parameters that minimize the following prediction error criterion:

$$\boldsymbol{\theta} = \arg\min \sum_{k=1}^{K} \left(y_k^* - \sum_{i=1}^{K} \gamma_i(\boldsymbol{x}_k)[\,\boldsymbol{x}_k^{\mathrm{T}} \quad \mathbf{1}\,]\boldsymbol{\theta}_i \right)^2,$$

where $\boldsymbol{\theta}^{\mathrm{T}} = [\boldsymbol{\theta}_1^{\mathrm{T}}, \ldots, \boldsymbol{\theta}_K^{\mathrm{T}}]$ is the concatenation of all the individual rules' parameter vectors. For the data matrices (17), this criterion can be rewritten in a matrix form:

$$\boldsymbol{\theta} = \arg\min(\boldsymbol{y}^* - \boldsymbol{\Lambda\theta})^{\mathrm{T}}(\boldsymbol{y}^* - \boldsymbol{\Lambda\theta}) \tag{28}$$

with $\boldsymbol{\Lambda} = [\Gamma_{1\varphi}, \ldots, \Gamma_{K\varphi}]$ where $\varphi = [\,\boldsymbol{X} \quad \mathbf{1}\,]$ and $\boldsymbol{\Gamma}_i = \mathrm{diag}(\gamma_i(\boldsymbol{x}_1), \ldots, \gamma_i(\boldsymbol{x}_N))$, i.e. a diagonal matrix having $\gamma_i(\boldsymbol{x}_k)$ as its kth diagonal element. The optimal solution of (28) is then directly obtained by using matrix pseudo-inverse:

$$\boldsymbol{\theta} = (\boldsymbol{\Lambda}^{\mathrm{T}}\boldsymbol{\Lambda})^{-1}\boldsymbol{\Lambda}^{\mathrm{T}}y^*. \tag{29}$$

4.1.2. Local least-squares estimation

While the global solution gives the minimal prediction error, it may bias the estimates of the consequents as parameters of local models. If locally relevant model parameters are required, a weighted least-squares approach applied per rule should be used. This is done by minimizing a set of K weighted local LS criteria:

$$\boldsymbol{\theta}_i = \arg\min(\boldsymbol{y}^* - \varphi\boldsymbol{\theta}_i)^{\mathrm{T}}\boldsymbol{\Gamma}_i(\boldsymbol{y}^* - \varphi\boldsymbol{\theta}_i), \quad i = 1, 2, \ldots, K \tag{30}$$

for which the solutions are:

$$\boldsymbol{\theta}_i = (\varphi^{\mathrm{T}}\boldsymbol{\Gamma}_i\varphi)^{-1}\varphi^{\mathrm{T}}\boldsymbol{\Gamma}_i\boldsymbol{y}^*, \quad i = 1, 2, \ldots, K.$$

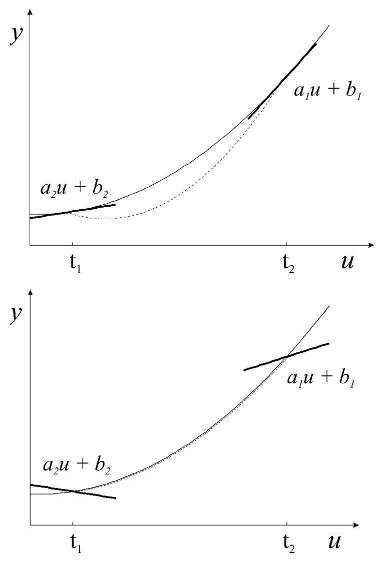


Fig. 8. The result of local (top) and global (bottom) estimation of the consequent parameters. The dashed line is the output of the model.

In this case, the consequent parameters of the individual rules are estimated independently of each other, and therefore the result is not influenced by the interactions of the rules. At the same time, however, a larger prediction error is obtained than with global least squares.

Example 4. The application of local and global estimation to the TS model from Example 3 results in the consequent models given in Fig. 8. Note that the consequents estimated by local least squares describe properly the local behavior of the function, but do not give a good fit. For global least squares, the opposite holds—a perfect fit is obtained, but the consequents are not relevant for the local behavior of the system.

When interpreting ANFIS models obtained from data, one has to be aware of the tradeoffs between local and global estimation. Constrained and multicriteria optimization can also be applied to restrict the freedom in the parameters.

4.1.3. Constrained estimation

Knowledge about the dynamic system such as its stability, minimal or maximal static gain, or its settling time can be translated into convex constraints on the consequent parameters (see Fig. 9). By using input–output data, optimal parameter values are then found by means of quadratic programming, instead of least squares. There are two types of constraints, global and local. Local constraints represent detail knowledge pertaining to each specific rule, while global constraints apply to the entire model and should thus refer to some global system properties such as the overall stability. To see this, realize that the affine TS model (5) can be regarded as one quasi-linear system:

$$y = \left(\sum_{i=1}^{K} \gamma_i(\boldsymbol{x})\boldsymbol{a}_i^{\mathrm{T}} \right) \boldsymbol{x} + \sum_{i=1}^{K} \gamma_i(\boldsymbol{x})b_i = \boldsymbol{a}^{\mathrm{T}}(\boldsymbol{x})\boldsymbol{x} + b(\boldsymbol{x}) \tag{31}$$

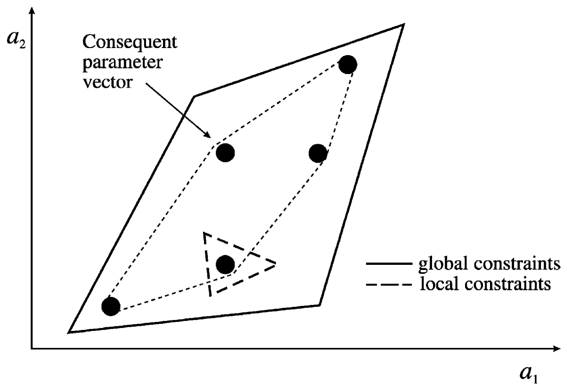


Fig. 9. Convex constraints on the consequent parameters.

with input-dependent 'parameters' $\boldsymbol{a}(\boldsymbol{x}), b(\boldsymbol{x})$ which are convex linear combinations of the individual consequent parameters $\boldsymbol{a}_i$ and b_i, i.e.:

$$\boldsymbol{a}(\boldsymbol{x}) = \sum_{i=1}^{K} \gamma_i(\boldsymbol{x})\boldsymbol{a}_i, \qquad b(\boldsymbol{x}) = \sum_{i=1}^{K} \gamma_i(\boldsymbol{x}) b_i. \tag{32}$$

This property allows us to define global convex constraints for the entire model. Besides, it also facilitates the analysis of TS models in the framework of poly-topic systems (linear differential inclusions). Methods have been developed to design controllers with desired closed loop characteristics and to analyze their stability (Tanaka, Ikeda, & Wang, 1998).

4.1.4. *Multi-objective optimization*

Another possibility is to regularize the estimation by penalizing undesired local behavior of the model. This can dramatically improve the robustness of the construction algorithm, eventually leading to more relevant (interpretable) parameter estimates. One way is to minimize the weighted sum of the global and local identification criteria (28) and (31):

$$\boldsymbol{\theta} = \arg\min \left\{ (\boldsymbol{y}^* - \boldsymbol{\Gamma\theta})^{\mathrm{T}}(\boldsymbol{y}^* - \boldsymbol{\Gamma\theta}) + \sum_{i=1}^{K} \delta_i (\boldsymbol{y}^* - \varphi\boldsymbol{\theta}_i)^{\mathrm{T}} \boldsymbol{\Gamma}_i (\boldsymbol{y}^* - \varphi\boldsymbol{\theta}_i) \right\}.$$

The weighting parameters $\delta_i \geq 0$ parameterize the set of Pareto-optimal solutions of the underlying multi-objective optimization problem and thus determine the tradeoff between the possibly conflicting objectives of global model accuracy and local interpretability of the parameters.

4.2. *Initialization of antecedent membership functions*

For a successful application of gradient-descent learning to the membership function parameters, good initialization is important. Several initialization methods are briefly reviewed in this section.

4.2.1. *Template-based membership functions*

With this method, the domains of the antecedent variables are a priori partitioned by a number of membership functions. These are usually evenly spaced and shaped. The rule base is then established to cover all the combinations of the antecedent terms. A severe drawback of this approach is that the number of rules in the model grows exponentially. Furthermore, if no knowledge is available as to which variables cause the nonlinearity of the system, all the antecedent variables are usually partitioned uniformly. However, the complexity of the system's behavior is typically not uniform. Some operating regions can be well approximated by a local linear model, while other regions require a rather fine partitioning. In order to obtain an efficient representation with as few rules as possible, the membership functions must be placed such that they capture the non-uniform behavior of the system.

4.2.2. *Discrete search methods*

Iterative tree-search algorithms can be applied to decompose the antecedent space into hyper-rectangles by axis-orthogonal splits. In each iteration, the region with the worst local error measure is divided into two halves (or other portions). Splits in all dimensions of the input are tested and the one with the highest performance improvement is chosen. This successive partitioning stops when a specified error goal is met or when the desired number of rules is reached. The first four steps of such an algorithm are illustrated in Fig. 10. An advantage of this approach is its effectiveness for high-dimensional data and the transparency of the obtained partition. A drawback is that the tree building procedure is sub-optimal (greedy) and hence the number of rules obtained can be quite large (Nelles, Fink, & Babuška, 2000).

4.2.3. *Fuzzy clustering*

Construction methods based on fuzzy clustering originate from data analysis and pattern recognition, where the concept of fuzzy membership is used to represent the degree to which a given data object is similar to some prototypical object. The degree of similarity can be calculated by using a suitable distance measure. Based on the similarity, data

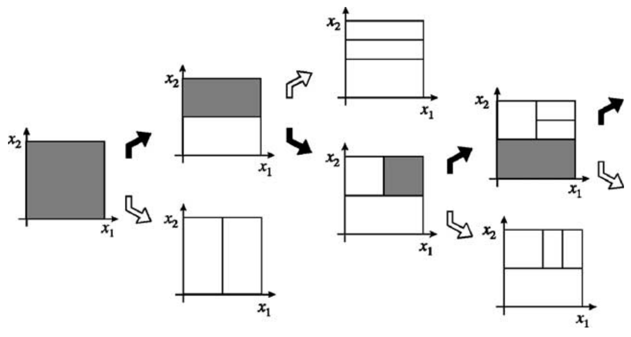


Fig. 10. Antecedent space decomposition by a heuristic search algorithm. The dark areas represent rules with the worst local fit in the given step.

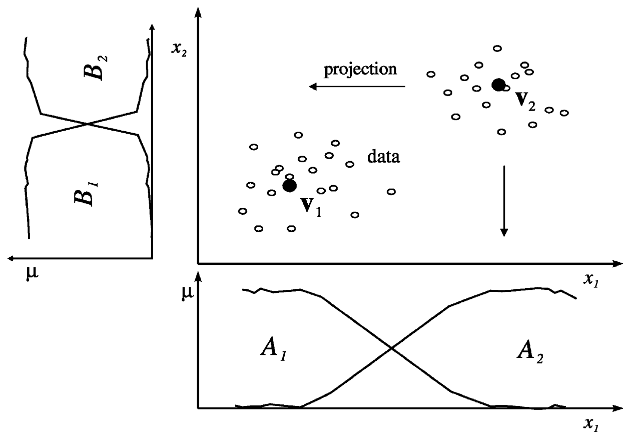


Fig. 11. Identification of membership functions through fuzzy clustering.

vectors are clustered such that the data within a cluster are as similar as possible, and data from different clusters are as dissimilar as possible.

Fig. 11 gives an example of two clusters in $\mathbb{R}^2$ with prototypes $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$. The partitioning of the data is expressed in the *fuzzy partition matrix* $\boldsymbol{U} = [\mu_{ij}]$ whose elements are the membership degrees of the data vectors $\boldsymbol{x}_k$ in the fuzzy clusters with prototypes $\boldsymbol{v}_j$. The antecedent membership functions are then extracted by projecting the clusters onto the individual variables. For the initialization of first-order ANFIS models, the prototypes can be defined as linear subspaces or the clusters are ellipsoids with adaptively determined shape. The number of clusters in the data can either be determined a priori or sought automatically by using cluster validity measures and merging techniques (Babuška, 1998).

5. Simulation examples

In this section, two simulation examples are given to illustrate several important issues related to the training of neuro-fuzzy systems. The first example is a simple fitting problem of a univariate static function. It demonstrates the typical construction procedure of a neuro-fuzzy model. Numerical results show that an improvement in performance is achieved at the expense of obtaining if–then rules that are not completely relevant as local descriptions of the system.

The second example, the modeling of a nonlinear dynamic system, illustrates that the performance of a neuro-fuzzy model does not necessarily improve after training. This is due to over-fitting which in the case of dynamic systems can easily occur when the data only sparsely cover the domains.

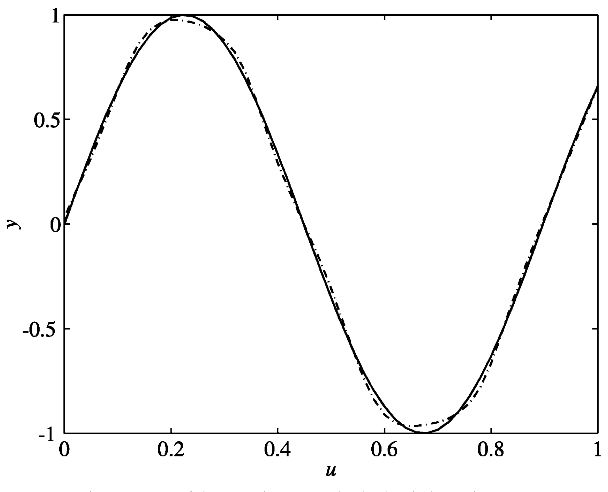
5.1. Static function

Let us approximate a univariate function $y = \sin(7u)$ by the ANFIS model with linear consequent functions. We choose the number of rules to be five and construct an initial model by clustering the data $U \times Y$, using a methodology based on the Gustafson–Kessel algorithm (Babuška, 1998). The following rules are obtained:

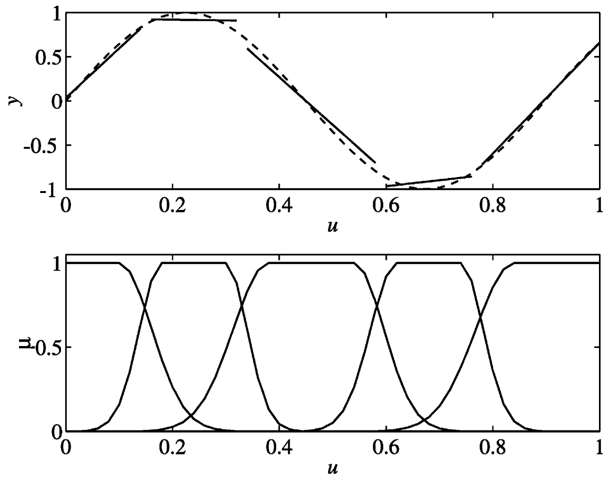
If u is A_1 then $y = 5.721u + 0.030$
If u is A_2 then $y = 0.035u + 0.904$
If u is A_3 then $y = -5.302u + 2.380$
If u is A_4 then $y = 0.734u - 1.413$
If u is A_5 then $y = 6.283u - 5.623$.

The fit of the function with this initial model is shown in Fig. 12a. The membership functions and the corresponding local models are given in Fig. 12b. The membership functions are denoted from left to right by A_1 through A_5.

Note that this initial model can easily be interpreted in terms of the local behavior (the rule consequents) and it is already reasonably accurate (the root-mean-squared error is RMS = 0.0258). However, by using the ANFIS method, the model parameters can be fine-tuned and the approximation accuracy can be significantly improved. A model



(a) The data (solid) and the model (dashed-dotted).



(b) Local models (top) and membership functions (bottom).

Fig. 12. Approximation by the initial TS fuzzy model. The root-mean-squared error is RMS = 0.0258.

obtained after 100 learning epochs of hybrid learning using the `anfis` function of the MATLAB Fuzzy Logic Toolbox (Mathworks, 2001) is described by the following fuzzy rules:

If u is A'_1 then $y = 5.275u + 0.065$
If u is A'_2 then $y = 0.442u + 0.899$
If u is A'_3 then $y = -3.206u + 1.405$
If u is A'_4 then $y = 0.977u - 1.693$
If u is A'_5 then $y = 5.062u - 4.388$.

The performance has improved to that degree that no approximation error is visible (not shown). The root-mean-squared error is now RMS $= 0.0011$, which is about 23 times better than the initial model. The membership functions have only been slightly modified, but the change in the local models is more apparent.

After learning, the local models are much further from the true local description of the function. To quantify this, we can compute the difference between the consequent parameters of the fuzzy models, denoted by $\boldsymbol{\theta}$, and the 'true' local estimates $\boldsymbol{\theta}_0$, computed by least squares for the data in core (A_i). For the initial fuzzy model, we have $||\boldsymbol{\theta} - \boldsymbol{\theta}_0|| = 1.81$, while the ANFIS model gives $||\boldsymbol{\theta} - \boldsymbol{\theta}_0|| = 5.30$. The rules of the fine-tuned neuro-fuzzy model are thus less accurate in describing the system locally. This contradiction between local and global approximation accuracy is inherent to TS fuzzy systems with linear consequents (Abonyi & Babuška, 2000) and thus also to the ANFIS network. Great care must be exercised when one attempts to interpret rules in trained neuro-fuzzy models.

5.2. pH neutralization process

A neutralization tank with three influent streams (acid, buffer and base) and one effluent stream is considered. The identification and validation data sets are obtained by simulating the model by Hall and Seborg (1989) for random changes of the influent base stream flow rate Q. The influent buffer stream and the influent acid stream are kept constant. The output is the pH in the tank. The identification data set, containing $N = 499$ samples with the sampling time of 15 s, is shown in Fig. 13. This data set was obtained from Johansen (1994).

The process is approximated as a first-order discrete time NARX model:

$$\mathrm{pH}(k+1) = f(\mathrm{pH}(k), Q(k)),$$

where k denotes the sampling instant, and f is an unknown relationship approximated by a neuro-fuzzy model. Based on prior knowledge about the process, it was decided to include only $Q(k)$ in the antecedent (it is known that the main source of nonlinearity is the titration curve, which is the steady-state characteristic relating Q to pH). The number of membership functions (and thus also rules) was set to three. The initial membership functions were evenly spread over the domain, as shown in the left panel of Fig. 14.

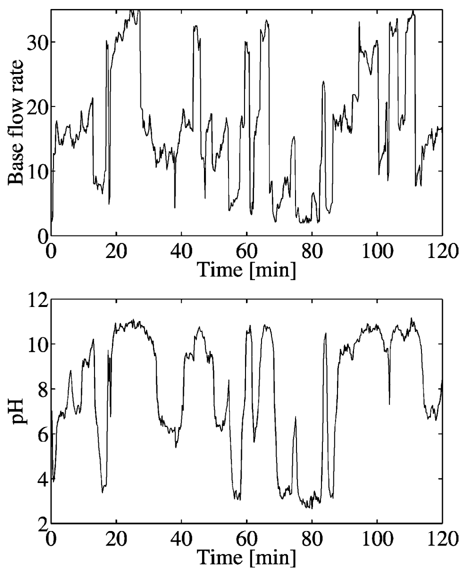


Fig. 13. Identification data.

The initial rule base, with the consequent estimated by weighted local least squares (31), is given by:

If $Q(k)$ is *Low* then
$\mathrm{pH}(k+1) = 0.83\mathrm{pH}(k) + 0.09Q(k) + 0.03$
If $Q(k)$ is *Medium* then
$\mathrm{pH}(k+1) = 0.83\mathrm{pH}(k) + 0.09Q(k) + 0.10$
If $Q(k)$ is *High* then
$\mathrm{pH}(k+1) = 0.46\mathrm{pH}(k) + 0.02Q(k) + 5.09$.

After 1000 epochs of hybrid learning using the ANFIS function of the MATLAB Fuzzy Logic Toolbox (Mathworks, 2001), the following rule base has been obtained:

If $Q(k)$ is *Low'* then
$\mathrm{pH}(k+1) = 0.37\mathrm{pH}(k) - 0.05Q(k) + 2.14$
If $Q(k)$ is *Medium'* then
$\mathrm{pH}(k+1) = 0.91\mathrm{pH}(k) + 0.06Q(k) - 0.23$
If $Q(k)$ is *High'* then
$\mathrm{pH}(k+1) = 0.40\mathrm{pH}(k) + 0.03Q(k) + 5.63$.

Note that the consequent model in the first rule has a negative coefficient for $Q(k)$. As this is a physically impossible value, not interpretation can be given to these parameters and this trained model has become a complete black-box. Also notice in Fig. 14 that the membership functions were adjusted in

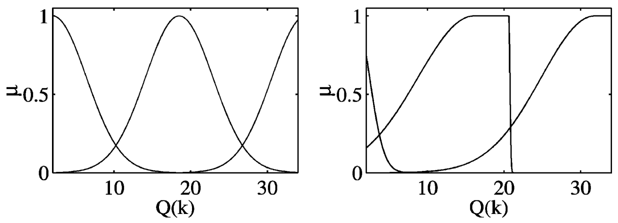


Fig. 14. Membership functions before (left) and after training (right). The membership functions are denoted from left to right by 'Low,' 'Medium' and 'High.'

Table 2
RMS before and after training

	Before training	After training
Training data set	0.90	0.82
Validation data set	0.81	0.89

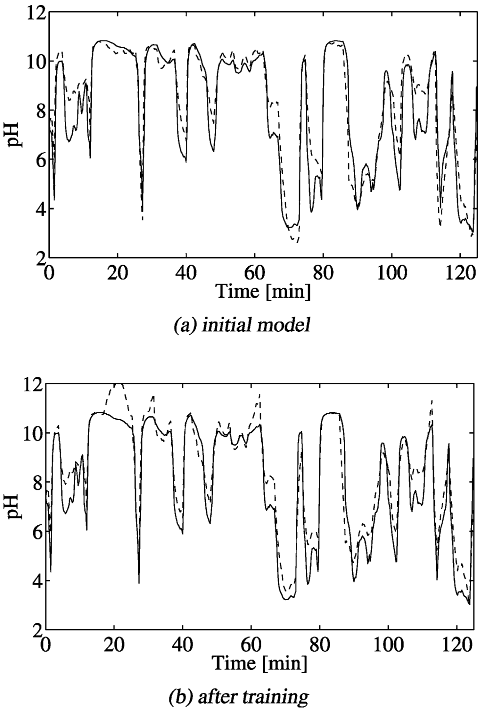


Fig. 15. Performance of the initial and trained model on the validation data set (solid line: data; dashed line: model).

a very peculiar way by the gradient-descent optimization method.

Table 2 shows that while the numerical performance in terms of the RMS error improved for the training data, it has become worse for the validation data. This is a typical example of over-training. This can also be observed in Fig. 15 where the predictions generated by the model are less accurate after training than before. Clearly, this kind of behavior is difficult to predict for a new problem at hand. The importance proper model validation can thus hardly be overemphasized.

6. Concluding remarks

Neuro-fuzzy modeling is a flexible framework in which different paradigms can be combined, providing, on the one hand, a transparent interface with the designer and, on the other hand, a tool for accurate nonlinear modeling and control. The rule-based character of neuro-fuzzy models allows for the analysis and interpretation of the result. Conventional methods for numerical validation can be complemented by human expertise, which often involves heuristic knowledge and intuition.

A drawback of neuro-fuzzy modeling is that the current techniques for constructing and tuning fuzzy models are rather complex, and their use requires specific skills and knowledge. In this sense, neuro-fuzzy modeling will probably never become a 'one-button,' fully automated identification technique. It should rather be seen as an interactive method, facilitating the active participation of the user in a computer-assisted modeling session. This holds, to a certain degree, also for other, more established methods. Modeling of complex systems will always remain an interactive approach.

6.1. Further reading

More details on the different methods and tools can be found in references Brown and Harris (1994), Jang et al. (1997), and Hellendoorn and Driankov (1997), among others. A large number of works are being regularly published in fuzzy systems oriented journals (*IEEE Transactions on Fuzzy Systems*, *Fuzzy Sets and Systems*) and also *IEEE Transactions on Systems, Man & Cybernetics*.

6.2. Software

Various tools have been developed for MATLAB. Examples are the Fuzzy Logic Toolbox (http://www.mathworks.com/products/fuzzylogic) and the Fuzzy Modeling and Identification Toolbox by R. Babuška (http://dcsc.tudelft.nl/~babuska). These tools were used to generate the solutions of the examples in this paper.

References

Abonyi, J., & Babuška, R. (2000). Local and global identification and interpretation of parameters in Takagi–Sugeno fuzzy models. In *Proceedings of the 9th IEEE International Conference on Fuzzy Systems* (pp. 835–840), San Antonio, USA.

Babuška, R. (1998). *Fuzzy modeling for control*. Boston: Kluwer Academic Publishers.

Brown, M., & Harris, C. (1994). *Neuro-fuzzy adaptive modelling and control*. New York: Prentice-Hall.

Hall, R. C., & Seborg, D. E. (1989). Modelling and self-tuning control of a multivariable pH neutralization process. Part I: Modelling and multi-loop control. In *Proceedings of American Control Conference* (Vol. 2, pp. 1822–1827), Pittsburgh, PA.

Hellendoorn, H., & Driankov, D. (Eds.). (1997). *Fuzzy model identification: Selected approaches*. Berlin, Germany: Springer-Verlag.

Jang, J.-S.R. (1993). ANFIS: Adaptive-network-based fuzzy inference systems. *IEEE Transactions on Systems, Man & Cybernetics, 23*(3), 665–685.

Jang, J.-S.R., Sun, C.-T. (1993). Functional equivalence between radial basis function networks and fuzzy inference systems. *IEEE Transactions on Neural Networks, 4*(1), 156–159.

Jang, J.-S.R., Sun, C.-T., & Mizutani, E. (1997). *Neuro-fuzzy and soft computing: A computational approach to learning and machine intelligence*. Upper Saddle River, NJ: Prentice-Hall.

Johansen, T. A. (1994). *Operating regime based process modelling and identification*. Ph.D. dissertation, The Norwegian Institute of Technology, University of Trondheim. Trondheim, Norway.

Mathworks, T. (2001). *Fuzzy logic toolbox for use with* MATLAB. *User's guide*, version 2. Natick, MA: The Mathworks, Inc.

Nelles, O., Fink, A., Babuška, R., Setnes, M. (2000). Comparison of two construction algorithms for Takagi–Sugeno fuzzy models. *International Journal of Applied Mathematics and Computer Science*, *10*(4), 835–855.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel distributed processing*. Cambridge, MA: MIT Press.

Tanaka, K., Ikeda, T., Wang, H.O. (1998). Fuzzy regulators and fuzzy observers: Relaxed stability conditions and LMI-based designs. *IEEE Transactions on Fuzzy Systems*, *6*(2), 250–265.