Neurofuzzy Modelling

Jan Jantzen <jj@iau.dtu.dk>[1]

Abstract

A neural network can approximate a function, but it is impossible to interpret the result in terms of natural language. The fusion of neural networks and fuzzy logic in neurofuzzy models provide learning as well as readability. Control engineers find this useful, because the models can be interpreted and supplemented by process operators.

Contents

[1] Technical University of Denmark, Department of Automation, Bldg 326, DK-2800 Lyngby, DENMARK. Tech. report no 98-H-874 (nfmod), 30 Oct 1998.

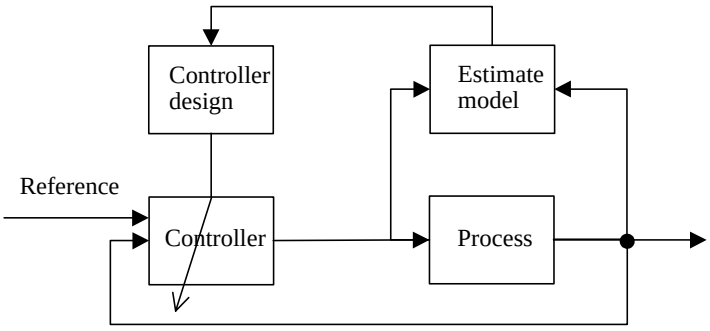

Figure 1: Indirect adaptive control: The controller parameters are updated indirectly via a process model.

1. Introduction

A neural network can model a dynamic plant by means of a nonlinear regression in the discrete time domain. The result is a network, with adjusted weights, which approximates the plant. It is a problem, though, that the knowledge is stored in an *opaque* fashion; the learning results in a (large) set of parameter values, almost impossible to interpret in words. Conversely, a fuzzy rule base consists of readable if-then statements that are almost natural language, but it cannot learn the rules itself. The two are combined in *neurofuzzy systems* in order to achieve readability and learning ability at the same time. The obtained rules may reveal insight into the data that generated the model, and for control purposes, they can be integrated with rules formulated by control experts (operators).

Assume the problem is to model a process such as in the *indirect adaptive controller* in Fig. 1. A mechanism is supposed to extract a model of the nonlinear process, depending on the current operating region. Given a model, a controller for that operating region is to be designed using, say, a pole placement design method. One approach is to build a two-layer perceptron network that models the plant, linearise it around the operating points, and adjust the model depending on the current state (Nørgaard, 1996). The problem seems well suited for the so-called *Takagi-Sugeno* type of neurofuzzy model, because it is based on piecewise linearisation.

Extracting rules from data is a form of modelling activity within *pattern recognition*, *data analysis* or *data mining,* also referred to as *the search for structure in data* (Bezdek and Pal, 1992). The goal is to reduce the complexity in a problem, or to reduce the amount of data associated with a problem. The field of data analysis comprises a great variety of methods; the objective of this note is to present a feasible way of combining fuzzy and neural networks. The neural network research started in the 1940s, and the fuzzy logic research in the 1960s, but the neurofuzzy research area is relatively new. The first book was probably by Kosko (1992). His ideas were implemented slightly earlier in the commercial tool TILGen (Hill, Horstkotte and Teichrow, 1990), and in 1995 came the Fuzzy Logic

Toolbox for Matlab (Jang and Gulley, 1995), which includes a neurofuzzy method. Many other commercial neurofuzzy tools are now available (see MIT, 1995).

Basically, we intend to try and describe a set of data collected from a process by means of rules. The description would be able to reproduce the training data, not necessarily with a zero error, but in an interpolative way. The approach here is to investigate a plain trial and error method, refine it with clustering methods, and refine it further with learning methods.

2. Trial and error

The classical way of modelling a plant, or a controller, is to acquire rules from experts based on their experience. The most common approach is to question experts or operators using a carefully organised questionnaire. The *input space*, that is, the coordinate system formed by the input variables (position, velocity, error, change in error) is partitioned into a number of regions. Each input variable is associated with a family of fuzzy term sets, say, 'negative', 'zero', and 'positive'. The membership functions must then be defined by the expert. For each valid combination of inputs, the expert is supposed to give typical values for the outputs.

For many systems, however, knowledge about the partitioning of the input space is unavailable. In that case, the input space can be partitioned by a number of equally spaced and shaped membership functions. The rule base can be established as all the possible combinations of the input terms. The task for the expert is then to estimate the outputs. The design procedure would be

1. Select relevant input and output variables,
2. determine the number of membership functions associated with each input and output, and
3. design a collection of fuzzy rules.

To accomplish this the expert relies on common sense, physical laws, or just trial and error. The result is a rule base which more or less describes the behaviour of the plant.

Example 1 (expert modelling) *Given a data set, plotted with a dashed line in Fig.* 2 *(top), we shall assume the expert's role and try to describe it with rules. Let us divide the input space into three partitions: 'small', 'medium', and 'large'. A set of rules could be*

If input is small then output is 0

If input is medium then output is 0.5

If input is large then output is 1

Clearly, with triangular membership functions equally spaced, the result would be a straight line. Therefore we use the shapes in Fig. 2 *(bottom) after some trial and error. The input-output relationship of the model is shown in the upper plot of the figure. Obviously the fit is somewhat poor, but for a first attempt it might be acceptable. The model uses the weighted average of the output singletons for defuzzification.*

□

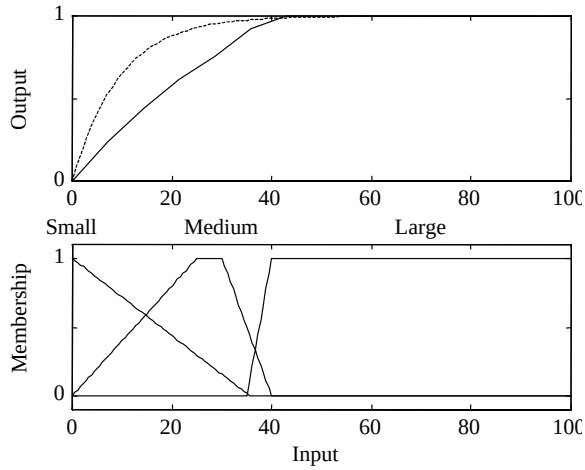

Figure 2: A fuzzy model approximation (solid line, top) of a data set (dashed line, top). The input space is divided into three fuzzy regions (bottom).

3. Clustering

The example above shows that it can be rather difficult to fit the fuzzy model to the target data using trial and error, although it is quite easy to express linguistically the relation between input and output. A better approach is to approximate the target function with a piece-wise linear function and interpolate, in some way, between the linear regions.

In the *Takagi-Sugeno* model (Takagi & Sugeno, 1985) the idea is that each rule in a rule base defines a region for a model, which can be linear. The left hand side of each rule defines a fuzzy validity region for the linear model on the right hand side. The inference mechanism interpolates smoothly between each local model to provide a global model. The general Takagi-Sugeno rule structure is

$$\text{If } f(e_1 \text{ is } \mathbf{A}_1, e_2 \text{ is } \mathbf{A}_2, \ldots, e_k \text{ is } \mathbf{A}_k) \text{ then } y = g(e_1, e_1 \ldots)$$

Here f is a logical function that connects the sentences forming the condition, y is the output, and g is a function of the inputs e_i. An example is

If error is positive and change in error is positive then
$u = K_p\,(error + T_d * change\ in\ error)$

where u is a controller's output, and the constants K_p and T_d are the familiar tuning constants for a proportional-derivative (PD) controller. Another rule could specify a PD controller with different tuning settings, for another operating region. The inference mechanism is then able to interpolate between the two controllers in regions of overlap.

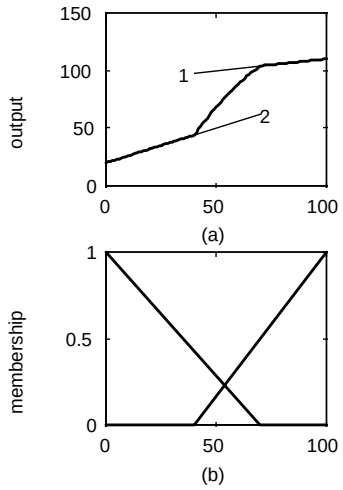

Figure 3: Interpolation between two lines (top) in the overlap of input sets (bottom).

Example 2 (Sugeno interpolation) *Suppose we have two rules*

1. *If error is Large then output is Line1*
2. *If error is Small then output is Line2*

Line 1 is defined as $0.2 * error + 90$ *and line 2 is defined as* $0.6 * error + 20$*. The rules interpolate between the two lines in the region where the membership functions overlap (Fig.* 3*). Outside of that region the output is a linear function of the* error.

□

In order set up a Takagi-Sugeno model of a process, it would be helpful to have an automatic method that finds *clusters* in the data that could be candidates for the linear regions. The objective of such a cluster analysis is to partition the data set into a number of natural and homogeneous subsets, where the elements of each subset are as similar to each other as possible, and at the same time as different from those of the other sets as possible. A *cluster* is such a grouping of objects, that is, it consists of all points close, in some sense, to the cluster centre.

3.1 Feature determination

In general, data analysis (Zimmermann, 1993) concerns *objects* which are described by *features*. Consider the example in Table 1 which contains data for some vehicles. Each row of such a table describes an object, and each column describes a feature. A feature can be regarded as a pool of values from which the actual values appearing in a given column are drawn. The features form axes of an abstract *feature space* in which each object is represented by a point. For instance, in the four-dimensional coordinate system spanned

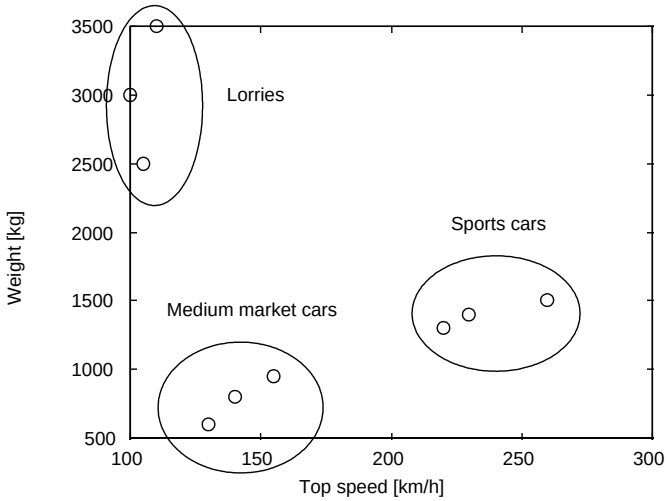

Figure 4: Clusters of vehicles in a feature space.

by the axes *top speed*, *air resistance*, *colour*, and *weight*, vehicle $V1$ is represented by the point $(x_1, x_2, x_3, x_4) = (220, red, 0.30, 1300)$. The colour axis is different from the other axes because its values are drawn from a discrete domain rather than the domain of real numbers. In order to maintain the idea of a feature space, the discrete domain must therefore be ordered.

Features can be selected directly, or they can be generated by a suitable combination of features. The latter option is necessary in case a large number of features needs to be reduced to a smaller number of features. Objects are fuzzy when one or more features are described in fuzzy terms. An example is a vehicle with a 'very fast' car engine, rather than top speed equal to some crisp number. Clusters are fuzzy when each object is associated with a degree of membership rather than a crisp membership. An example is the cluster of 'sports cars'; a particular car would be a member to a degree depending on its top speed, air resistance, and weight.

Example 3 (cars) *Figure* 4 *is a plot of top speed against air resistance of the data from Table* 1*. The plot can be regarded as a feature space, but as we are only plotting two of the four features, it is more precise to regard it as a projection of the total feature space onto the plane spanned by top speed and weight. It looks as if there are three clusters present in the data, say,* sports cars *(V1, V2, V3),* medium market cars *(V4, V5, V6), and* lorries *(V7, V8, V9). In this case we have selected two features out of four to make an assignment of vehicles to the three clusters.*

□

In practice data conditioning (preprocessing) is necessary. It can be an advantage if each

Vehicle	Top speed [km/h]	Colour	Air resistance	Weight [kg]
V1	220	red	0.30	1300
V2	230	black	0.32	1400
V3	260	red	0.29	1500
V4	140	grey	0.35	800
V5	155	blue	0.33	950
V6	130	white	0.40	600
V7	100	black	0.50	3000
V8	105	red	0.60	2500
V9	110	grey	0.55	3500

Table 1: Sample data (from MIT, 1997)

feature value u is *normalised* to its normalised value $\overline{u}$ on a standard universe, say $[0, 1]$, according to

$$\overline{u} = \frac{u - u_{\min}}{u_{\max} - u_{\min}} \tag{1}$$

Here u_{min} is the smallest value in a series of measurements and u_{max} is the largest. *Standardisation* transforms the mean of the set of feature values to zero, and the standard deviation to one. If the data are distributed according to the normal distribution with a mean value of m and standard deviation σ, the standardised values are found as follows:

$$u^* = \frac{u - m}{\sigma} \tag{2}$$

It may also be necessary to *scale* the values onto a particular range using an affine mapping. Scaling to a range $[u_1, u_2]$ is a linear transformation according to

$$u' = \frac{u - u_1}{u_2 - u_1}(u_2 - u_1) + u_1 \tag{3}$$

After preprocessing, relevant features can be selected by an expert. For a series of data $\mathbf{u} = (u_1, u_2, \ldots, u_K)$ some characteristic quantities, which can be used in forming the features, are

$$\begin{array}{ll} \text{mean value} & m = \frac{1}{K}\sum_{i=1}^{K} u_i \\ \text{variance} & v = \frac{1}{K-1}\sum_{i=1}^{K}(u_i - m)^2 \\ \text{standard deviation} & \sigma = \sqrt{v} \\ \text{range} & s = u_{\max} - u_{\min} \end{array} \tag{4}$$

Graphical representations such as plots of frequency spectra and histograms can support the selection.

Features may be correlated, that is, a change in one feature X is associated with a similar change in another feature Y. A *correlation coefficient* can be calculated as

$$r = \frac{\sum_{i=1}^{K}(x_i - m_1)(y_i - m_2)}{\sqrt{\sum_{i=1}^{K}(x_i - m_1)^2 \sum_{i=1}^{K}(y_i - m_2)^2}} \tag{5}$$

Here m_1 is the mean value of all the values x_i of feature X, and m_2 is the mean of all y_i values of the Y feature. The correlation coefficient assumes values in the interval $[-1, 1]$.

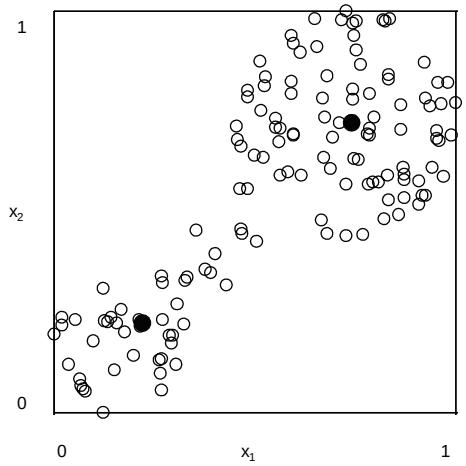

Figure 5: Example with two clusters (Jang & Gulley, 1995). Cluster centres are marked with solid circles.

When $r = -1$ there is a strong negative correlation between X and Y, when $r = 1$ there is a strong positive correlation, and when $r = 0$ there is no correlation at all. Strongly correlated features may indicate a linear dependency. If two features are linearly dependent one of them is *redundant* (unnecessary); it is sufficient to select just one of them as a feature.

3.2 Hard clusters (HCM algorithm).

The hard c-means (HCM) algorithm tries to locate clusters in the multi-dimensional feature space. The goal is to assign each point in the feature space to a particular cluster. The basic approach is as follows (see for example Lewis, 1990).

1. Manually seed the algorithm with c cluster centres, one for each cluster we are seeking. This requires prior information from the outside world of the number of different clusters into which the points are to be divided; thus the algorithm belongs to the class of *supervised* algorithms.
2. Each point is assigned to the cluster centre nearest to it.
3. A new cluster centre is computed for each class by taking the mean values of the coordinates of the points assigned to it.
4. If not finished according to some stopping criterion (see later), go to step 2.

Some additional rules can be added to remove the necessity of knowing precisely how many clusters there are. The rules allow nearby clusters to merge and clusters which have large standard deviations in coordinate to split.

Example 4 (two clusters) *Consider the data points plotted in Fig.* 5. *The cluster algo-*

rithm finds one centre in the lower left hand corner and another in the upper right hand corner. The initial centres are more or less in the middle of the plot, and during the iterations they move towards their final positions. Each point belongs to one or the other class, so the clusters are crisp. In general, the feature data have to be normalised in order for the distance measure to work properly.

□

Generally speaking the hard c-means algorithm is based on a *c-partition* of the data space U into a family of clusters $\{C_i\}$, $i = 1, 2, \ldots, c$, where the following set-theoretic equations apply,

$$\bigcup_{i=1}^{c} C_i = U \tag{6}$$

$$C_i \cap C_j = \emptyset, \quad \text{all } i \neq j \tag{7}$$

$$\emptyset \subset C_i \subset U, \quad \text{all } i \tag{8}$$

The set $U = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_K\}$ is a finite set of points in a space spanned by the feature axes, and c is the number of clusters. We note,

$$2 \leq c \leq K \tag{9}$$

because $c = K$ clusters just places each data sample into its own cluster, and $c = 1$ places all data samples into the same cluster. Equations (6)-(8) express, respectively, that the set of clusters exhausts the whole universe, that none of the clusters overlap, and that a cluster can neither be empty nor contain all data samples.

Formally, the c-means algorithm finds a centre in each cluster, minimising an objective function of a distance measure. The objective function depends on the distances between vectors $\mathbf{u}_k$ and cluster centres $\mathbf{c}_i$, and when the Euclidean distance is chosen as a distance function, the expression for the objective function is,

$$J = \sum_{i=1}^{c} J_i = \sum_{i=1}^{c} \left(\sum_{k, \mathbf{u}_k \in C_i} \|\mathbf{u}_k - \mathbf{c}_i\|^2 \right) \tag{10}$$

where J_i is the objective function within cluster i.

The partitioned clusters are typically defined by a $c \times K$ binary characteristic matrix $\mathbf{M}$, called the *membership matrix*, where each element m_{ik} is 1 if the kth data point $\mathbf{u}_k$ belongs to cluster i, and 0 otherwise. Since a data point can only belong to one cluster, the membership matrix $\mathbf{M}$ has the properties:

$$\begin{array}{l} -\text{the sum of each column is one, and} \\ -\text{the sum of all elements is } K. \end{array} \tag{11}$$

If the cluster centres $\mathbf{c}_i$ are fixed, the m_{ik} that minimise J_i can be derived as

$$m_{ik} = \begin{cases} 1 \text{ if } \|\mathbf{u}_k - \mathbf{c}_i\|^2 \leq \|\mathbf{u}_k - \mathbf{c}_j\|^2, & \text{for each } j \neq i \\ 0 \text{ otherwise} \end{cases} \tag{12}$$

That is, $\mathbf{u}_k$ belongs to cluster i if $\mathbf{c}_i$ is the closest centre among all centres. If, on the other hand, m_{ik} is fixed, then the optimal centre $\mathbf{c}_i$ that minimises (10) is the mean of all vectors

in cluster i,

$$\mathbf{c}_i = \frac{1}{|C_i|} \sum_{k, \mathbf{u}_k \in C_i} \mathbf{u}_k \tag{13}$$

where $|C_i|$ is the number of objects in C_i (its *cardinality*), and the summation is an element-by-element summation of vectors.

Algorithm (Jang, Sun and Mizutani, 1997) The hard c-means algorithm has five steps.

1. Initialise the cluster centres $\mathbf{c}_i$ $(i = 1, 2, \ldots, c)$. This is typically achieved by randomly selecting c points from the data points.
2. Determine the membership matrix **M** by (12).
3. Compute the objective function (10). Stop if either it is below a certain threshold value, or its improvement over the previous iteration is below a certain tolerance.
4. Update the cluster centres according to (13).
5. Go to step 2.

□

The algorithm is iterative, and there is no guarantee that it will converge to an optimum solution. The performance depends on the initial positions of the cluster centres, and it is advisable to employ some method to find good initial cluster centres. It is also possible to initialise a random membership matrix **M** first and then follow the iterative procedure.

Example 5 (rules) *The plot of the clusters in Fig.* 5 *suggests a relation between the variable x_1 on the horizontal axis and x_2 on the vertical axis. For example, the cluster in the upper right hand corner of the plot indicates, in very loose terms, that whenever x_1 is 'high', defined as near the right end of the horizontal axis, then x_2 is also 'high', defined as near the top end of the vertical axis. The relation can be described by the rule,*

$$\textit{if } x_1 \textit{ is high then } x_2 \textit{ is high} \tag{14}$$

Intuitively at least, it seems possible to make some sensible definitions of the two instances of the word 'high' in the rule, based on the location of the cluster centre. The cluster in the lower left part of the figure, could perhaps be described as

$$\textit{if } x_1 \textit{ is low then } x_2 \textit{ is low} \tag{15}$$

Again the accuracy of this rule depends on the two definitions of 'low'.

□

3.3 Fuzzy clusters (FCM algorithm)

It is reasonable to assume that points in the middle region of Fig. 5, between the two cluster centres, have a gradual membership of *both* clusters. Naturally this is accommodated by fuzzifying the definitions of 'low' and 'high' in (14) and (15). The fuzzified c-means algorithm (Bezdek in Jang et al., 1997) allows each data point to belong to a cluster to a degree specified by a membership grade, and thus each point may belong to several clusters.

The *fuzzy c-means* (FCM) algorithm partitions a collection of K data points specified by m-dimensional vectors $\mathbf{u}_k$ $(k = 1, 2, \ldots, K)$ into c fuzzy clusters, and finds a cluster centre

in each, minimising an objective function. Fuzzy c-means is different from hard c-means, mainly because it employs *fuzzy partitioning,* where a point can belong to several clusters with degrees of membership. To accommodate the fuzzy partitioning, the membership matrix **M** is allowed to have elements in the range $[0, 1]$. A point's total membership of all clusters, however, must always be equal to unity to maintain the properties (11) of the **M** matrix. The objective function is a generalisation of (10),

$$J(\mathbf{M}, \mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_c) = \sum_{i=1}^{c} J_i = \sum_{i=1}^{c} \sum_{k=1}^{K} m_{ik}^q d_{ik}^2, \tag{16}$$

where m_{ik} is a membership between 0 and 1, $\mathbf{c}_i$ is the centre of fuzzy cluster i, $d_{ik} = \|\mathbf{u}_k - \mathbf{c}_i\|$ is the Euclidean distance between the ith cluster centre and kth data point, and $q \in [1, \infty)$ is a weighting exponent. There are two necessary conditions for J to reach a minimum,

$$\mathbf{c}_i = \frac{\sum_{k=1}^{K} m_{ik}^q \mathbf{u}_k}{\sum_{k=1}^{K} m_{ik}^q}, \tag{17}$$

and

$$m_{ik} = \frac{1}{\sum_{j=1}^{c} \left(\frac{d_{ik}}{d_{jk}} \right)^{2/(q-1)}} \tag{18}$$

The algorithm is simply an iteration through the preceding two conditions.

Algorithm (Jang et al., 1997) In a batch mode operation, the fuzzy c-means algorithm determines the cluster centres $\mathbf{c}_i$ and the membership matrix **M** using the following steps:

1. Initialise the membership matrix **M** with random values between 0 and 1 within the constraints of (11).
2. Calculate c cluster centres $\mathbf{c}_i$ $(i = 1, 2, \ldots, c)$ using (17).
3. Compute the objective function according to (16). Stop if either it is below a certain threshold level or its improvement over the previous iteration is below a certain tolerance.
4. Compute a new **M** using (18).
5. Go to step 2.

□

The cluster centres can alternatively be initialised first, before carrying out the iterative procedure. The algorithm may not converge to an optimum solution and the performance depends on the initial cluster centres, just as in the case of the hard c-means algorithm.

Example 6 (FCM) *Getting back to the problem of modelling the test data in Example 1, the data were submitted to the FCM function in the Matlab Fuzzy Logic Toolbox. Asking for three clusters, all other settings default, it finds three cluster centres (Fig.* 6*). These can be used as indications of where to place the peaks of three fuzzy membership functions on the input axis.*

□

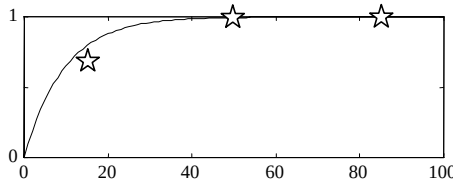

Figure 6: The FCM algorithm finds the cluster centres indicated by stars when asked to find three clusters.

3.4 Subtractive clustering

Fuzzy c-means is a supervised algorithm, because it is necessary to tell it how many clusters c to look for. If c is not known beforehand, it is necessary to apply an unsupervised algorithm. *Subtractive clustering* is based on a measure of the density of data points in the feature space (Chiu in Jang et al., 1997). The idea is to find regions in the feature space with high densities of data points. The point with the highest number of neighbours is selected as centre for a cluster. The data points within a prespecified, fuzzy radius are then removed (subtracted), and the algorithm looks for a new point with the highest number of neighbours. This continues until all data points are examined.

Consider a collection of K data points specified by m-dimensional vectors $\mathbf{u}_k$, $k = 1, 2, \ldots, K$. Without loss of generality, the data points are assumed normalised. Since each data point is a candidate for a cluster centre, a *density measure* at data point $\mathbf{u}_k$ is defined as

$$D_k = \sum_{j=1}^{K} \exp\left(-\frac{\|\mathbf{u}_k - \mathbf{u}_j\|}{(r_a/2)^2}\right), \tag{19}$$

where r_a is a positive constant. Hence, a data point will have a high density value if it has many neighbouring data points. Only the fuzzy neighbourhood within the radius r_a contributes to the density measure.

After calculating the density measure for each data point, the point with the highest density is selected as the first cluster centre. Let $\mathbf{u}_{c_1}$ be the point selected and D_{c_1} its density measure. Next, the density measure for each data point $\mathbf{u}_k$ is revised by the formula

$$D'_k = D_k - D_{C_1} \exp\left(-\frac{\|\mathbf{u}_k - \mathbf{u}_{C_1}\|}{(r_b/2)^2}\right), \tag{20}$$

where r_b is a positive constant. Therefore, the data points near the first cluster centre $\mathbf{u}_{c_1}$ will have significantly reduced density measures, thereby making the points unlikely to be selected as the next cluster centre. The constant r_b defines a neighbourhood to be reduced in density measure. It is normally larger than r_a to prevent closely spaced cluster centres; typically $r_b = 1.5 * r_a$.

After the density measure for each point is revised, the next cluster centre $\mathbf{u}_{c_2}$ is selected and all the density measures are revised again. The process is repeated until a sufficient

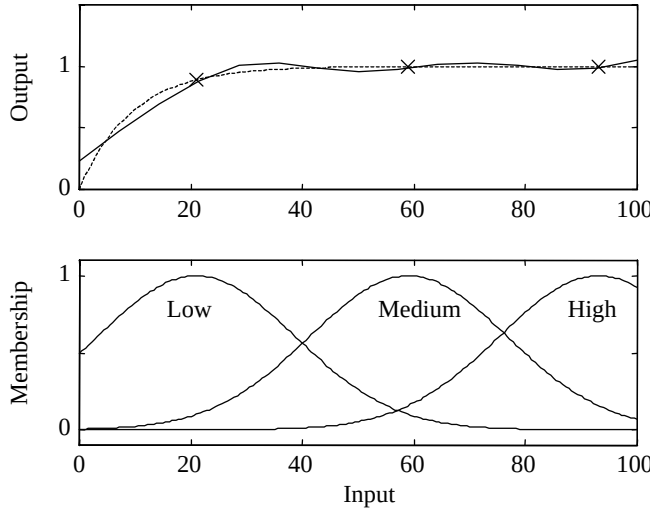

Figure 7: Result of applying subclustering to test data. The upper plot shows the target function (dashed) and the estimate (solid) as well as the clustere centres (x). The lower plot shows the three membership functions corresponding to the clusters.

number of cluster centres are generated.

When applying subtractive clustering to a set of input-output data, each of the cluster centres represents a rule. To generate rules, the cluster centres are used as the centres for the premise sets in a singleton type of rule base (or the radial basis functions in a radial basis function neural network, see later).

Example 7 (subclust) *The function* `genfis2` *in the Fuzzy Logic Toolbox uses subclustering to generate rules that approximate a function. Applying it to the data from the previous example produces cluster centres that are on the curve, and slightly shifted to the right relative to FCM, cp. Figs.* 6 *and* 7*. The rules are*

$$\textit{If } input \textit{ is Low then } output = 0.0328 * input - 0.943 \quad (21)$$

$$\textit{If } input \textit{ is Medium then } output = 0.0357 * input + 0.237 \quad (22)$$

$$\textit{If } input \textit{ is High then } output = 0.0206 * input - 1.11 \quad (23)$$

The rule extraction method first uses subclustering to determine the number of rules and input membership functions, then linear least squares estimation to determine each rule's output equations. Each fuzzy cluster is mapped into a generalised bell-shaped membership function defined as

$$bell\,(x; a, b, c) = \frac{1}{1 + \left|\frac{x-c}{a}\right|^{2b}}$$

The parameter c is taken to be the centre of the cluster, and a is the cluster radius, defined as

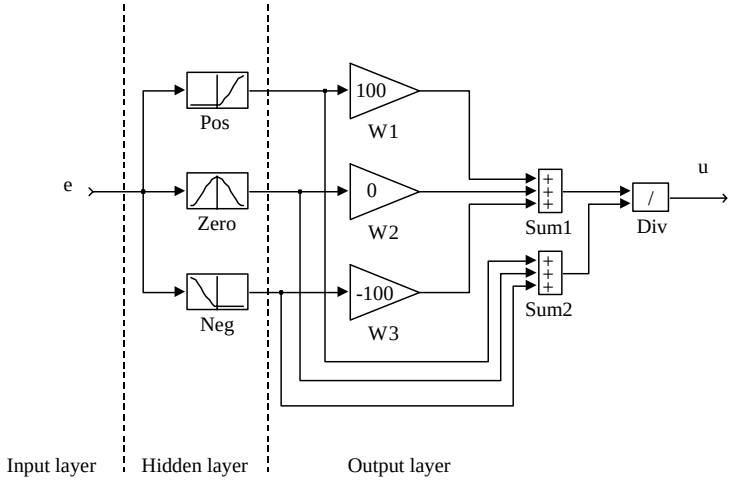

Figure 8: Three rules perceived as a network.

the longest distance from the centre to a point p with nonzero membership. The parameter b is a slope, determined as a linear function of the membership of the boundary point p.

□

4. Neurofuzzy function approximation

It immediately comes to mind, when looking at a neural network, that the activation functions look like fuzzy membership functions. Indeed, an early paper from 1975 treats the extension of the McCulloch-Pitts neuron to a fuzzy neuron (Lee & Lee, 1975; see also Keller & Hunt, 1985).

Consider a standard rule base for a fuzzy proportional controller with the error e as input and a control signal u with singleton membership functions as the output,

$$\text{If } e \text{ is Pos then } u \text{ is -100} \tag{24}$$

$$\text{If } e \text{ is Zero then } u \text{ is 0} \tag{25}$$

$$\text{If } e \text{ is Neg then } u \text{ is -100} \tag{26}$$

The inference mechanism can be drawn in a block diagram somewhat like a neural network (Fig. 8). The network has an input layer, one hidden layer, and one output layer. The input node connects to the neurons in the hidden layer, this corresponds to the if-part of the rules. Each neuron only consists of an activation function, there is no summation, because each neuron has only one input. The singleton control signals appear as weights on the outputs from the neurons. The one neuron in the output layer, with a rather odd appearance, calculates the weighted average corresponding to the *centre of gravity defuzzification* in the

rule base. The network can be generalised to multi-input-multi-output control, but then the diagram becomes very busy.

Backpropagation applies to this network since all layers are differentiable. Two possibilities for learning are apparent. One is to adjust the weights in the output layer, $i.e.$, all the singletons w_i until the error is minimized. The other is to adjust the shape of the membership functions, provided they are parametric.

The network can be described as a feedforward network with an input layer, a single hidden layer, and an output layer consisting of a single unit. The network performs a nonlinear mapping from the input layer to the hidden layer, followed by a linear mapping from the hidden layer to the output layer. Exactly such a topology occurs in *radial basis function* networks; the hidden units provide a 'basis' for the input patterns and their functions 'radially' surround a particular data point.

Radial basis function networks are used for curve-fitting in a multi-dimensional space. This is also called *function approximation,* and learning is equivalent to finding a function that best fits the training data. In its *strict* sense the function is constrained to pass through all the training data points. The radial basis functions technique consists of choosing a function F,

$$\mathrm{F}(\mathbf{u}) = \mathbf{w}^T \mathrm{f}\left(\|\mathbf{u}-\mathbf{u}_k\|\right) \tag{27}$$

$$= \begin{bmatrix} w_1 & w_2 & \dots & w_k \end{bmatrix} \begin{bmatrix} \mathrm{f}\left(\|\mathbf{u}-\mathbf{u}_1\|\right) \\ \mathrm{f}\left(\|\mathbf{u}-\mathbf{u}_2\|\right) \\ \dots \\ \mathrm{f}\left(\|\mathbf{u}-\mathbf{u}_K\|\right) \end{bmatrix} \tag{28}$$

Here $\mathbf{u} \in \mathbb{R}^m$ is a vector of inputs, $\mathbf{u}_k \in \mathbb{R}^m$ $(k = 1, 2, \dots, K)$ are vectors of training data, $\mathbf{w} \in \mathbb{R}^K$ is the vector of weights, $\mathrm{f}\left(\|\mathbf{u}-\mathbf{u}_k\|\right)$ is a set of (nonlinear) radial basis functions, and $\|\cdot\|$ is a norm, usually the Euclidean. The known data points $\mathbf{u}_k$ are taken to be the *centres* of the radial basis functions. The activation level of a function $\mathrm{f}\left(\mathbf{u}, \mathbf{u}_k\right)$ is *maximum* when the input $\mathbf{u}$ is at the centre $\mathbf{u}_k$ of the function, as demonstrated by the next example.

Example 8 (radial basis function) *The Gaussian is an example of a radial basis function*

$$\mathrm{f}(\mathbf{u}) = \exp(-\|\mathbf{u}-\mathbf{u}_1\|^2) \tag{29}$$

A two-dimensional sample, with two inputs $\mathbf{u} = (u_1, u_2)$*, is plotted in Fig.* 9 *with its centre at* $(u_1, u_2)_1 = (5, 5)$*. When* $\mathbf{u}$ *is near the centre, the exponent is near zero, and* $\mathrm{f}(\mathbf{u})$ *is nearly one, which is its maximum. The index 1 in (29) refers to the first training example. A second training example would be associated with a similar Gaussian function, but placed over another centre.*

□

Each known data point $\mathbf{u}_k$ must satisfy the equation,

$$\mathbf{w}^T \mathrm{f}\left(\mathbf{u}_k\right) = d_k \tag{30}$$

where d_k is the desired response corresponding to $\mathbf{u}_k$, therefore the unknown weights $\mathbf{w}$

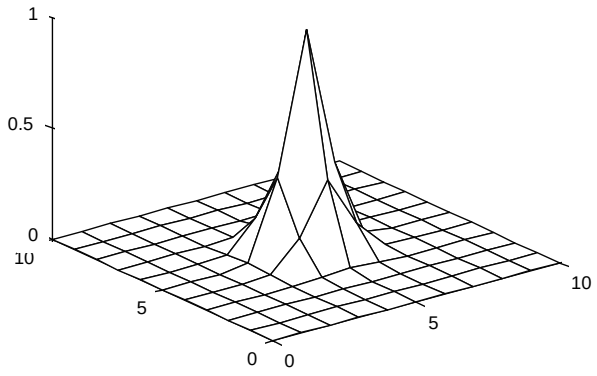

Figure 9: A Gaussian radial basis function.

must satisfy the following set of equations

$$\begin{bmatrix} w_1 & w_2 & \dots & w_K \end{bmatrix} \begin{bmatrix} \mathrm{f}_{11} & \mathrm{f}_{12} & \dots & \mathrm{f}_{1K} \\ \mathrm{f}_{21} & \mathrm{f}_{22} & \dots & \mathrm{f}_{2K} \\ \dots & \dots & \dots & \dots \\ \mathrm{f}_{K1} & \mathrm{f}_{K2} & \dots & \mathrm{f}_{KK} \end{bmatrix} = \begin{bmatrix} d_1 & d_2 & \dots & d_K \end{bmatrix} \tag{31}$$

In matrix notation

$$\mathbf{w}^T \mathbf{\Phi} = \mathbf{d}^T \tag{32}$$

where the vector **d** represents the *desired response* vector, and

$$\mathrm{f}_{jk} = \mathrm{f}\left(\|\mathbf{u}_j - \mathbf{u}_k\|\right), \qquad j,k = 1,2,\dots,K \tag{33}$$

The matrix $\mathbf{\Phi} = \{\mathrm{f}_{jk}\}$ is called the *interpolation matrix*. For a class of radial basis functions, Gaussian functions for instance, the interpolation matrix is invertible (it is positive definite). Provided the data points are all distinct, then we can solve for the weights directly, obtaining

$$\mathbf{w}^T = \mathbf{d}^T \mathbf{\Phi}^{-1} \tag{34}$$

Although in theory this means we can solve the *strict* interpolation problem where the function passes through all training points $\mathbf{u}_k$, in practice we cannot, if the interpolation matrix is close to singular.

The performance of the network depends only little on the type of function $\mathrm{f}(\cdot)$, according to theoretical investigations and practical experiences (Powell in Haykin, 1994). The performance may be improved by adjustments of the centre and the shape of the activation functions. Generally the radial basis function networks enjoy faster convergence than back-propagation networks.

4.1 Adaptive Neurofuzzy Inference System (ANFIS)

ANFIS (Adaptive Neuro Fuzzy Inference System) is an architecture which is functionally

equivalent to a Sugeno type fuzzy rule base (Jang, Sun & Mizutani, 1997; Jang & Sun, 1995). Under certain minor constraints the ANFIS architecture is also equivalent to a radial basis function network. Loosely speaking ANFIS is a method for tuning an existing rule base with a learning algorithm based on a collection of training data. This allows the rule base to adapt.

The network in Fig. 8 may be extended by assigning a linear function to the output weight of each neuron,

$$w_k = \mathbf{a}_k^T \mathbf{u} + b_k, \quad k = 1, 2, \ldots, K \tag{35}$$

where $\mathbf{a}_k \in \mathbb{R}^m$ is a parameter vector and b_k is a scalar parameter. The network is then equivalent to a first order Sugeno type fuzzy rule base (Takagi and Sugeno, 1985). The requirements for the radial basis function network to be equivalent to a fuzzy rule base is summarised in the following (Jang et al., 1997).

- Both must use the same aggregation method (weighted average or weighted sum) to derive their overall outputs.
- The number of activation functions must be equal to the number of fuzzy if-then rules.
- When there are several inputs in the rule base, each activation function must be equal to a composite input membership function. One way to achieve this is to employ Gaussian membership functions with the same variance in the rule base, and apply product for the **and** operation. The multiplication of the Gaussian membership functions becomes a multi-dimensional Gaussian radial basis function.
- Corresponding activation functions and fuzzy rules should have the same functions on the output side of the neurons and rules respectively.

If the training data are contained in a small region of the input space, the centres of the neurons in the hidden layer can be concentrated within the region and sparsely cover the remaining area. Thus only a local model will be formed and if the test data lie outside the region, the performance of the network will be poor. On the other hand, if one distributes the basis function centres evenly throughout the input space, the number of neurons depends exponentially on the dimension of the input space.

4.2 ANFIS architecture

Without loss of generality we assume two inputs, u_1 and u_2, and one output, y. Assume for now a first order Sugeno type of rule base with the following two rules

$$\text{If } u_1 \text{ is } A_1 \text{ and } u_2 \text{ is } B_1 \text{ then } y_1 = c_{11}u_1 + c_{12}u_2 + c_{10} \tag{36}$$

$$\text{If } u_1 \text{ is } A_2 \text{ and } u_2 \text{ is } B_2 \text{ then } y_2 = c_{21}u_1 + c_{22}u_2 + c_{20} \tag{37}$$

Incidentally, this fuzzy controller could interpolate between two linear controllers depending on the current state. If the firing strengths of the rules are α_1 and α_2 respectively, for two particular values of the inputs u_1 and u_2, then the output is computed as a weighted average

$$y = \frac{\alpha_1 y_1 + \alpha_2 y_2}{\alpha_1 + \alpha_2} = \bar{\alpha}_1 \, y_1 + \bar{\alpha}_2 \, y_2 \tag{38}$$

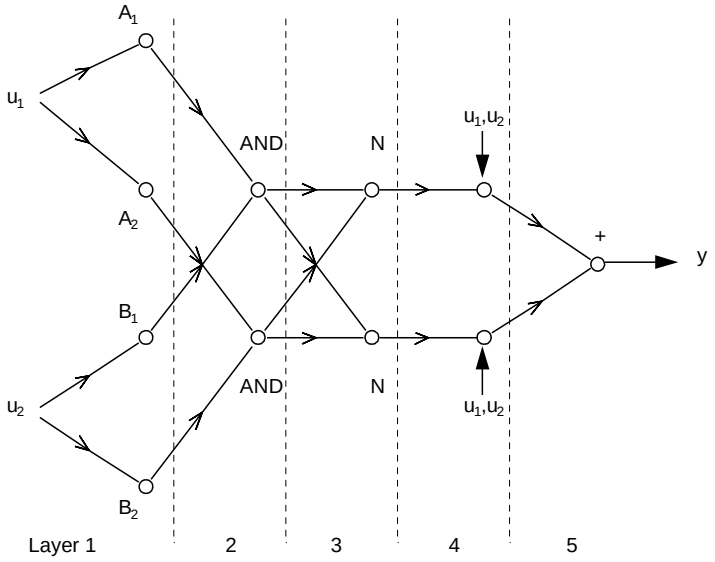

Figure 10: Structure of the ANFIS network.

The corresponding ANFIS network is shown in Fig. 10. A description of the layers in the network follows.

1. Each neuron i in layer 1 is adaptive with a parametric activation function. Its output is the grade of membership to which the given input satisfies the membership function, i.e., $\mu_{A_1}(u_1)$, $\mu_{B_1}(u_2)$, $\mu_{A_2}(u_1)$, or $\mu_{B_2}(u_2)$. An example of a membership function is the generalised *bell function*

$$\mu(x) = \frac{1}{1 + \left|\frac{x-c}{a}\right|^{2b}} \tag{39}$$

   where $\{a, b, c\}$ is the parameter set. As the values of the parameters change, the shape of the bell-shaped function varies. Parameters in that layer are called *premise parameters*.
2. Every node in layer 2 is a fixed node, whose output is the product of all incoming signals. In general, any other fuzzy AND operation can be used. Each node output represents the firing strength α_i of the ith rule.
3. Every node in layer 3 is a fixed node which calculates the ratio of the ith rule's firing strength relative to the sum of all rule's firing strengths,

$$\bar{\alpha}_i = \frac{\alpha_i}{\alpha_1 + \alpha_2}, \qquad i = 1, 2 \tag{40}$$

   The result is a *normalised firing strength.*

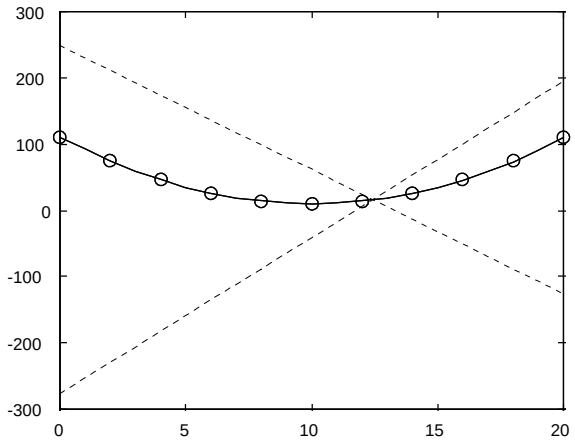

Figure 11: Approximation of data points (o) by an ANFIS network (solid). Two rules interpolate between two lines (dotted).

4. Every node in layer 4 is an adaptive node with a node output

$$\bar{\alpha}_i \; y_i = \bar{\alpha}_i \; (c_{i1} u_1 + c_{i2} u_2 + c_{i0}) \,, \qquad i = 1, 2 \tag{41}$$

   where $\bar{\alpha}_i$ is the normalised firing strength from layer 3 and $\{c_{i1}, c_{i2}, c_{i0}\}$ is the parameter set of this node. Parameters in this layer are called *consequent parameters*.
5. Every node in layer 5 is a fixed node which sums all incoming signals.

It is straight forward to generalise the ANFIS architecture in Fig. 10 to a rule base with more than two rules.

4.3 The ANFIS learning algorithm

When the premise parameters are fixed, the overall output is a linear combination of the consequent parameters. In symbols, the output y can be written as

$$y = \frac{\alpha_1}{\alpha_1 + \alpha_2} y_1 + \frac{\alpha_2}{\alpha_1 + \alpha_2} y_2 \tag{42}$$

$$= \bar{\alpha}_1 \; (c_{11} u_1 + c_{12} u_2 + c_{10}) + \bar{\alpha}_2 \; (c_{21} u_1 + c_{22} u_2 + c_{20}) \tag{43}$$

$$= (\bar{\alpha}_1 \; u_1) \, c_{11} + (\bar{\alpha}_1 \; u_2) \, c_{12} + \bar{\alpha_1} \; c_{10} + (\bar{\alpha}_2 \; u_2) \, c_{21} + (\bar{\alpha}_2 \; u_2) \, c_{22} + \bar{\alpha_2} \; c_{20} \tag{44}$$

which is linear in the consequent parameters $c_{ij} \;\; (i = 1, 2; j = 0, 1, 2)$. A hybrid algorithm adjusts the consequent parameters c_{ij} in a forward pass and the premise parameters $\{a_i, b_i, c_i\}$ in a backward pass (Jang et al., 1997). In the forward pass the network inputs propagate forward until layer 4, where the consequent parameters are identified by the least-squares method. In the backward pass, the error signals propagate backwards and the premise parameters are updated by gradient descent.

Because the update rules for the premise and consequent parameters are decoupled in

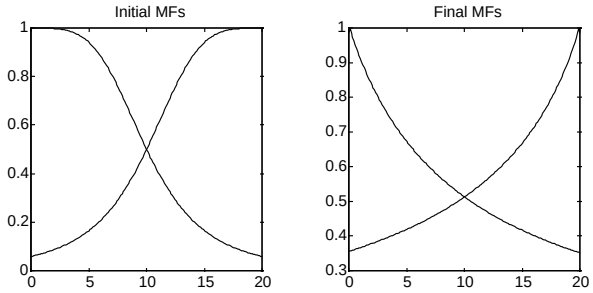

Figure 12: Membership functions before (left) and after (right) learning.

the hybrid learning rule, a computational speedup may be possible by using variants of the gradient method or other optimisation techniques on the premise parameters. Since ANFIS and radial basis function networks (RBFNs) are functionally equivalent under some minor conditions (p 17), a variety of learning methods can be used for both of them.

Example 9 (ANFIS) *To see how an ANFIS network can approximate a function, Fig.* 11 *shows a plot of a set of data-points and the resulting interpolating curve. Eleven data-points (circled in the figure) were presented to the ANFIS network. Initially two Gaussian input membership functions were chosen (Fig.* 12 *left). They cover the whole input range with 50 percent overlap. Another initial design choice was the number of rules, i.e., two. The result of the learning is a rule base with two rules*

$$\textit{If } x \textit{ is } A_1 \textit{ then } y_1 \quad = \quad -18.75x + 249.1 \tag{45}$$

$$\textit{If } x \textit{ is } A_2 \textit{ then } y_2 \quad = \quad 23.55x - 276.7 \tag{46}$$

The right hand side of the rules are two straight lines, also drawn on the figure, one with a negative slope and the other with a positive slope. The interpolating curve is the result of a nonlinear blend of the two straight lines. The weighting of the lines in each point of the interpolating curve is determined by the input membership functions A_1 *and* A_2 *(Fig.* 12 *right).*

The eleven data points were presented to the network 100 times, during which the ANFIS algorithm updated the premise parameters, that determine the shape and position of the two membership functions, and the consequent parameters, that determine the slope and the constant of the two lines on the output side (they were initially all zero).

□

4.4 Genetic algorithms

A problem with backpropagation and least squares optimisation is that they can be trapped in a local minimum of a nonlinear objective function, because they are derivative based. *Genetic algorithms* – survival of the fittest! – are derivative-free, stochastic optimisa-

tion methods, and therefore less likely to get trapped. They can be used to optimise both structure and parameters in neural networks. A special application for them is to determine fuzzy membership functions.

A genetic algorithm mimics the evolution of populations. First, different possible solutions to a problem are generated. They are tested for their performance, that is, how good a solution they provide. A fraction of the good solutions is selected, and the others are eliminated (survival of the fittest). Then the selected solutions undergo the processes of *reproduction*, *crossover*, and *mutation* to create a new *generation* of possible solutions, which is expected to perform better than the previous generation. Finally, production and *evaluation* of new generations is repeated until convergence. Such an algorithm searches for a solution from a broad spectrum of possible solutions, rather than where the results would normally be expected. The penalty is computational intensity. The elements of a genetic algorithm are explained next (Jang et al., 1997).

1. *Encoding*. The parameter set of the problem is encoded into a bit string representation. For instance, a point $(x, y) = (11, 6)$ can be represented as a *chromosome*, which is a concatenated bit string

1	0	1	1	0	1	1	0	(47)

   Each coordinate value is a *gene* of four bits. Other encoding schemes can be used, and arrangements can be made for encoding negative and floating point numbers.
2. *Fitness evaluation*. After creating a *population* the fitness value of each member is calculated. For a maximisation problem, the fitness value of the ith member is the value of the objective function at point i. Usually strictly positive objective functions are employed. Another possible fitness measure is to use a ranking of the members of the population, then the objective function can be inaccurate as long as it provides the correct ranking.
3. *Selection*. The algorithm selects which *parents* should participate in producing offsprings for the next generation. Usually the probability of selection for a member is proportional to its fitness value. The idea is to let members with above-average fitness reproduce and replace members with below-average fitness.
4. *Crossover*. Crossover operators generate new chromosomes that hopefully retain good features from the previous generation. Crossover is usually applied to selected pairs of parents with a probability equal to a given *crossover rate*. In one-point crossover a crossover point on the genetic code is selected at random and two parent chromosomes interchange their bit strings to the right of this point. Take for example two chromosomes

1	0	1	1	0	*1*	*1*	*0*	(48)
1	0	0	1	0	*0*	*0*	*0*	(49)

   If the crossover point is between the fifth bit and the sixth, the digits written in italics will swap places vertically. The two new chromosomes will be

1	0	1	1	0	*0*	*0*	*0*	(50)
1	0	0	1	0	*1*	*1*	*0*	(51)

In two-point crossover, two crossover points are selected and the part of the chromosome string between these two points is swapped to generate two children; and so on. In effect, parents pass segments of their own chromosomes on to their children, and some children will be able to outperform their parents if they get good genes from both parents.

5. *Mutation*. A mutation operator can spontaneously create new chromosomes. The most common way is to flip a bit with a probability equal to a very low, given *mutation rate*. The mutation prevents the population from converging towards a local minimum. The mutation rate is low in order to preserve good chromosomes.

Note that the above is only a general description of the basics of a genetic algorithm; detailed implementations vary considerably. For a textbook on the subject of genetic algorithms, see for example Goldberg (1989).

Algorithm (Jang et al., 1997) An example of a simple genetic algorithm for a maximisation problem is the following.

1. Initialise the population with randomly generated individuals and evaluate the fitness of each individual.
   (a) Select two members from the population with probabilities proportional to their fitness values.
   (b) Apply crossover with a probability equal to the crossover rate.
   (c) Apply mutation with a probability equal to the mutation rate.
   (d) Repeat (a) to (d) until enough members are generated to form the next generation.
3. Repeat steps 2 and 3 until a stopping criterion is met.

□

If the mutation rate is high (above 0.1), the performance of the algorithm will be as bad as a primitive random search.

Example 10 (line fit) *(Ross, 1995) This is an example of how a line may be fit to a given data set using a genetic algorithm. Consider the data set*

x	y
1	*1*
2	*2*
3	*3*
6	*6*

(52)

The line to be fitted is

$$y = ax + b \tag{53}$$

The parameters (a, b) *must be encoded in the form of bit strings with a random assignment of* 0*s and* 1*s at different bit locations, for example,*

0	*0*	*0*	*1*	*1*	*1*	*0*	*1*	*0*	*1*	*0*	*0*

(54)

Each chromosome is 12 *bits long, the first gene encode* a *and the second gene encode* b*. We start with an initial population of four chromosomes. The binary values of the genes must be converted and mapped to decimal numbers that make sense in (53). A suitable affine*

mapping is

$$\begin{aligned} c &= c_{\min} + \frac{c_{bin}}{2^6 - 1}\,(c_{\max} - c_{\min}) && (55)\\ &= -2 + \frac{c}{2^6 - 1}\,(5 + 2) && (56)\\ &= c/9 - 2 && (57) \end{aligned}$$

where c_{bin} is the decimal value of the binary gene, and $c_{\max} = 5$ and $c_{\min} = -2$ are assumed upper and lower limits of c. Thus, in the first iteration, the decimal value of gene a is 7, which is mapped into $a = 7/9 - 2 = -1.22$ and the decimal value of gene b is 20, which is mapped into $b = 0.22$. The initial population is four, and the following table shows the first iteration

Member	*Chromosomes*	a	b	$\mathcal{E}(\mathbf{y})$	$\mathcal{E}/\operatorname{mean}(\mathcal{E})$	*Copies*
1	*000111 010100*	-1.22	0.22	147	0.48	0
2	*010010 001100*	0.00	-0.67	332	1.08	1
3	*010101 101010*	0.33	2.67	391	1.27	2
4	*100100 001001*	2.00	-1.00	358	1.17	1

(58)

The objective function is

$$\mathcal{E}(\mathbf{y}) = 400 - \sum (d - \mathbf{y})^2 \qquad (59)$$

*The sum of the squared errors is subtracted from a large number 400 to convert the problem into a maximisation problem. The variable d is the desired y-value from (52), and the vector **y** is the result of inserting all x-values from (52) into the equation for the line (53). The value of the objective function is the value of the fitness of each string. Each fitness value is divided by the average of the fitness values for the whole population to give an estimate of the relative fitness of each string. All strings with a relative fitness less than 0.8 is eliminated, and the rest of the fitnesses are scaled and rounded such that their sum is four; the new numbers are the number of copies to use of each individual for the next generation.*

The next table concerns the second generation. The first column shows the individuals selected and they are aligned for crossover at the locations shown by a vertical bar (|).

Selected	*New chroms*	a	b	$\mathcal{E}(\mathbf{y})$	$\mathcal{E}/\operatorname{mean}(\mathcal{E})$	*Copies*
0101|01 101010	*010110 001100*	0.44	-0.67	376	1.15	1
0100|10 001100	*010001 101010*	-0.11	2.67	381	1.17	2
010101 101|010	*010101 101001*	0.33	2.56	292	0.90	1
100100 001|001	*100100 001010*	2.00	-0.89	256	0.78	0

(60)

Not shown in the two tables is the average fitness of the first generation 307 increasing to 326 in the second generation. The iterations should be continued until convergence to a solution within a generation.

□

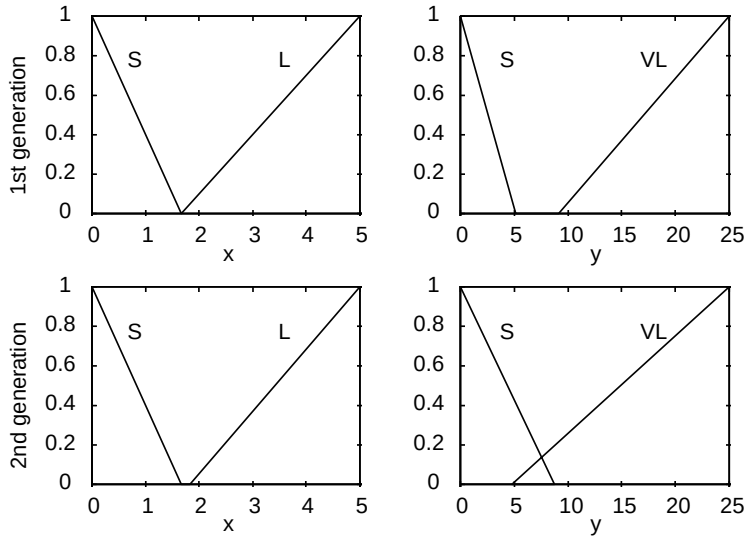

Figure 13: Best individuals in the first generation (top) and the second generation (bottom).

4.5 Computing membership functions

Genetic algorithms can be used to find membership functions (Karr & Gentry in Ross, 1995). Initially some membership functions and their shapes are assumed for the various fuzzy variables in the system. The membership functions are encoded as bit strings and concatenated. A fitness function is used to evaluate the fitness of each set of parameters that define the membership functions. The process is illustrated for a simple problem in the next example.

Example 11 (membership functions) *(Ross, 1995) Given a single-input, single-output rule base*

$$\textit{If } x \textit{ is S then } y \textit{ is S} \tag{61}$$

$$\textit{If } x \textit{ is L then } y \textit{ is VL} \tag{62}$$

The input x uses two fuzzy terms S (small) and L (large), and the output y uses S (small) and VL (very large). The problem is to try and fit the rule base, by adjusting the membership functions, to the following mapping

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 4 & 9 & 16 & 25 \end{bmatrix} \tag{63}$$

We assume triangular membership functions, with their right angle vertices fixed at the extremes of the respective universes of discourse. The input universe is assumed to be $[0, 5]$ and the output universe $[0, 25]$. For each membership function, the only adjustable

parameter is the length of the base (the leg of the triangle on the horizontal axis, see Fig. 13*).*

We use six-bit genes to define the base of each membership function. The genes are concatenated to give a 6 *bits/variable* $\times$ 4 *variables* $=$ 24 *bit chromosome. We start with a population of four and compute the base lengths using the conversion scheme (55)-(57) from the previous example,*

Member	*Chromosomes*	*Base S*	*Base L*	*Base S*	*Base VL*
1	*000111 010100 010110 110011*	0.56	1.59	8.73	20.2
2	*010010 001100 101100 100110*	1.43	0.95	17.5	15.1
3	*010101 101010 001101 101000*	1.67	3.33	5.16	15.9
4	*100100 001001 101100 100011*	2.86	0.71	17.5	13.9

(64)

The calculations proceed as in Example 10, details can be found in Ross (1995). After two iterations there is a slight change in the membership functions, see Figure 13*.*

□

5. Test results and discussion

For the sake of comparison, consider a very simple, nonlinear, static single-input-single-output plant is considered,

$$y = \tanh(u) + \sin(u) + e \tag{65}$$

The plant output y is governed by the input u and some Gaussian white noise e with zero mean and standard deviation 0.1. The noise is included to show the ability to extract parameters from noisy training data. The reason for choosing this simple example is to compare results with Sørensen (1994); he used it with a multi-layer perceptron to compare four different learning algorithms.

From the plant a training set of $N = 63$ samples of matching inputs and outputs $\{u(k), y(k)\}, (k = 1, 2, \cdots, N)$ is collected. The training set is equally spaced in the interval $u \in [-\pi, \pi]$. Unlike dynamic plants, k denotes the sample number rather than time, implying that the order in which the training examples are presented to the model is unimportant. The initial model has two membership functions (Neg and Pos), and ANFIS is asked to train for 500 epochs. ANFIS returns an adjusted rule base,

$$\begin{aligned} &\text{If input is } \textit{Neg} \text{ then output is } -1.921 * input - 8.065 \\ &\text{If input is } \textit{Pos} \text{ then output is } -2.819 * input + 12.41 \end{aligned} \tag{66}$$

The training data and the ANFIS approximation is shown in Fig. 14, top. The adjusted membership functions, which we have called *Neg* and *Pos*, are in the same figure (bottom).

A qualitative comparison with Sørensen (1994) is possible by means of Fig. 15, which shows how the approximation error develops over the training epochs. The result is that ANFIS converges faster than Sørensen's two simple algorithms, backpropagation and steepest descent. It is, however, a little bit slower than his Gauss Newton method, and it has about the same convergence rate as his parallel Gauss Newton method. Also shown in Fig 15 is

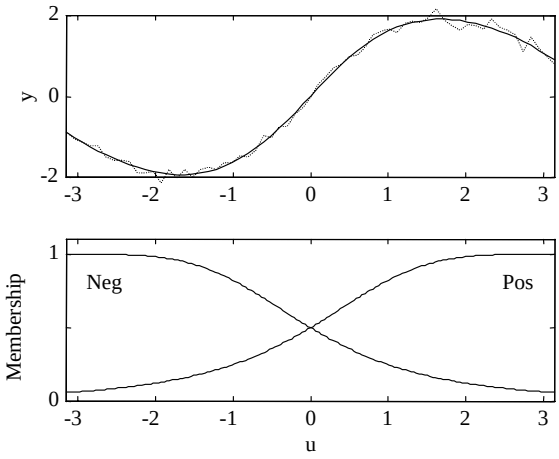

Figure 14: Top: ANFIS approximation (solid) of training data (dotted). Bottom: Two membership functions that cover the input range $[-\pi, \pi]$.

a dashed line, showing the optimal error, i.e.,

$$std\left(\tanh(u) - \sin(u) - y\right) = 0.113 \tag{67}$$

This is the actual standard deviation of the training data from the ideal function. ANFIS dives below the line, more than Sørensen's methods, indicating that it is trying to learn the noise (overfitting).

It is interesting to notice, that a run with ANFIS' default settings, results in a similar model with two input membership functions, but it stops after only 10 iterations. At this point the standard deviation on the error is almost optimal (0.119) and the error curve, cp. Fig. 15, is a straight line.

□

The example shows some strengths and shortcomings of neurofuzzy modelling for control purposes:

- We do not need to know the plant dynamics in advance; identification of them is embedded in the training of ANFIS.
- The example was based on off-line learning only. It is possible to turn on on-line learning to cope with time-varying plant dynamics. In general, the best approach is to use off-line learning to find a working controller for a nominal plant, and then use on-line learning to fine-tune the controller if the plant is time-varying.
- The distribution of the training data could pose a problem. Ideally, we would like to see the training data distributed across the input space of the plant in a somewhat uniform manner. This may not be possible due either to the scarcity of the data (especially when there are many inputs) or to limits imposed by the physical plant.

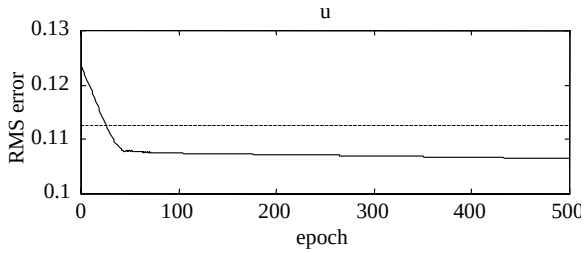

Figure 15: The learning curve for ANFIS is rather steep, and it drops below the optimal error indicating overtraining.

In traditional linear system identification it is well known that the frequency spectrum of the input signal must be sufficiently wide. With neurofuzzy methods this is required too, but also the amplitude spectrum of the input signal must be sufficiently wide, because the networks are nonlinear. It is thus necessary to train the network with a number of different amplitudes in order to learn the nonlinearities of the plant. It can be difficult, however, to determine beforehand how to manipulate the inputs in such a way that the internal variables contain sufficiently varied amplitudes.

ANFIS uses least squares optimisation which may favour local linear models which are *not* local linear approximations. Figure 11 shows the two lines between which the rule base interpolates, they are absolutely not local approximations to the function. Such a model can hardly be interpreted in terms of the individual rules. This can be achieved by having membership functions with less overlap and wider peaks (larger *cores*), because the linear models are then forced to represent local behaviour; other rules are not interfering. Babuška (1998) has studied this carefully, and his *product-space clustering* method is based on the philosophy that the local models should be local linearisations. He also makes the point, that the fuzzy c-means algorithm, as it is presented here, prefers clusters that are spherical; elongated clusters can be difficult to detect, especially if they are not perpendicular to the feature axes. He proposes to use a cluster algorithm which adapts to the shape of the cluster (Gustafson & Kessel in Babuska, 1998, p 60).

6. Conclusions

Compared to neural networks, the neurofuzzy methods provide models which can be interpreted by human beings. The models are in the form of the familiar if-then rules, implying easy integration with operators' (expert) rules. The ANFIS model shows good performance, but sometimes produces spurious rules, that make little sense. For automatic control purposes our interest lies in the application of these methods to system identification and adaptive control systems.

References

Babuška, R. (1998). *Fuzzy Modeling For Control*, Kluwer Academic Publishers.

Bezdek, J. and Pal, S. K. (1992). *Fuzzy models for pattern recognition*, IEEE Press, New York. (Selected reprints).

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimisation, and Machine Learning*, Addison-Wesley, Reading, MA, USA.

Haykin, S. (1994). *Neural Networks: A Comprehensive Foundation*, Macmillan College Publishing Company, Inc., 866 Third Ave, New York, NY 10022.

Hill, G., Horstkotte, E. and Teichrow, J. (1990). *Fuzzy-C development system – userś manual*, Togai Infralogic, 30 Corporate Park, Irvine, CA 92714, USA.

Jang, J.-S. R. and Gulley, N. (1995). *Fuzzy Logic Toolbox*, The MathWorks Inc., 24 Prime Park Way, Natick, Mass. 01760-1500.

Jang, J.-S. R. and Sun, C.-T. (1995). Neuro-fuzzy modeling and control, *Proceedings of the IEEE* **83**(3): 378–406.

Jang, J.-S. R., Sun, C.-T. and Mizutani, E. (1997). *Neuro-Fuzzy and Soft Computing*, number isbn 0-13-261066-3 in *Matlab Curriculum Series*, Prentice Hall, Upper Saddle River, NJ, USA.

Keller, J. M. and Hunt, D. J. (1985). Incorporating fuzzy membership functions into the perceptron algorithm, in *Fuzzy models for pattern recognition* (Bezdek and Pal, 1992), pp. 468–474. (Selected reprints).

Kosko, B. (1992). *Neural Networks and Fuzzy Systems. A Dynamical Systems Approach to Machine Intelligence*, Prentice–Hall, Englewood Cliffs.

Lee, S. C. and Lee, E. T. (1975). Fuzzy neural networks, in *Fuzzy models for pattern recognition* (Bezdek and Pal, 1992), pp. 448–467. (Selected reprints).

Lewis, R. (1990). *Practical Digital Image Processing*, Ellis Horwood Series in Digital and Signal Processing, Ellis Horwood Ltd, New York, etc.

MIT (1995). *CITE Literature and Products Database*, MIT GmbH / ELITE, Promenade 9, D-52076 Aachen, Germany.

Nørgaard, P. M. (1996). *System Identification and Control with Neural Networks*, PhD thesis, Technical University of Denmark, Dept. of Automation, Denmark.

Ross, T. (1995). *Fuzzy Logic with Engineering Applications*, McGraw-Hill, New York, N.Y.

Takagi, T. and Sugeno, M. (1985). Fuzzy identification of systems and its applications to modeling and control, *IEEE Trans. Systems, Man & Cybernetics* **15**(1): 116–132.

Zimmermann, H.-J. (1993). *Fuzzy set theory - and its applications*, second edn, Kluwer, Boston. (1. ed. 1991).