ELSEVIER

Fuzzy Sets and Systems 79 (1996) 69-91

FUZZY sets and systems

The representation of fuzzy algorithms used in adaptive modelling and control schemes

M. Brown, D.J. Mills, C.J. Harris*

Image, Speech and Intelligent Systems research group, Department of Electronics and Computer Science, University of Southampton, Southampton, SO17 1BJ, UK

Abstract

This paper will compare and contrast two apparently different approaches for representing linguistic fuzzy algorithms as well as discussing their relevance to neurofuzzy adaptive modelling and control schemes. *Discrete* fuzzy implementations which store the relational information and set definitions at discrete points have traditionally been used within the control community, whereas *continuous* fuzzy systems which store and manipulate functional relationships have recently gained in popularity due to their strong links with neural networks. It is shown that when algebraic operators are used to implement the underlying fuzzy logic, a simplified defuzzification calculation can be used in both cases, although the continuous fuzzy systems have a lower computational cost and generally a smoother output surface. Several neurofuzzy training rules are investigated and links are made with standard optimisation algorithms. The merits of adapting weights rather than rule confidences or relational elements are discussed and it is shown to be more efficient to train the neurofuzzy system in weight space. This paper's aim is to present a *consistent* and *computationally efficient* approach to implementing neurofuzzy algorithms and to relate it to more conventional systems.

1. Introduction

Research into fuzzy systems is currently receiving a lot of attention, thus mimicking the revival of interest in artificial neural networks which occurred in the late 1980s. Both fields are aimed at developing systems, motivated by current biological and physiological understanding of the brain, which possess the ability to learn, reason and generalise about noisy, redundant information. Although a wide variety of algorithms have been proposed, many are similar and often it is only the implementation method that differs. This paper compares several neural and fuzzy networks, using the notation developed for representing fuzzy systems, and shows that the two apparently different implementation methods: *continuous* and *discrete* representations, are equivalent under certain conditions and the relationship between these fuzzy networks and radial basis function and B-spline neural networks is explained [5, 34]. It then concentrates on the continuous approach and develops and discusses several training algorithms.

The term "fuzzy" is widely (ab)used in the literature. It has been said that fuzzy systems are inherently robust with respect to both plant parametric variations and measurement uncertainty. Similarly, it has been claimed that fuzzy systems are easily understood by the non-specialist because they use vague terms to explain their control actions, and that fuzzy systems can be verified and validated quicker than

* Corresponding author. E-mail: mqb@ecs.soton.ac.uk.

conventional models and controllers. The power of a fuzzy approach lies in the way vague, imprecise rules such as:

IF (*error is small*) THEN (*output is small*)

can be given a specific meaning and the manner in which the fuzzy system produces outputs for inputs which only partially match the rules used for initialisation/training. The partial matching ensures *local generalisation* i.e. that similar inputs produce similar outputs whereas dissimilar inputs produce independent outputs.

Traditionally fuzzy modelling and control systems have been implemented using a discrete approach where the fuzzy sets and relations are represented by the fuzzy membership value at a set of discrete points [16, 19, 31]. This representation is very flexible in that an explicit membership function does not need to be stored, although generally the number of sample points required to represent the functions is large and depends on the method used to fuzzify the information presented to the network. This paper also argues that the type of operators which are inappropriate for use within continuous fuzzy systems reduce the quality of the decision surfaces and would not be used because of this factor alone.

Continuous fuzzy systems have recently gained in popularity [5, 35], partially due to their link with certain neural networks. In a continuous fuzzy network, the fuzzy membership functions are stored as continuous mappings that depend on a set of parameters, and a rule confidence (or rule weight) is used to denote the strength with which a particular rule fires. The memory requirements are generally much less than that required for a discrete fuzzy implementation, although the computational cost is sometimes slightly greater. One significant advantage with this approach is that it allows concepts such as: *normalised fuzzy sets*, *rule confidences*, *least mean square learning* and *universal approximation* to be investigated and analysed within a consistent framework. Previously, fuzzy systems have been shown to outperform standard non-linear modelling and control designs on certain problems although there has been a lack of theoretical rigour associated with this approach. The results that can be derived using continuous fuzzy systems allow new insight to be gained into how vague linguistic knowledge is implemented in a rule base and how approximate reasoning algorithms *generalise*.

This paper proposes a common framework for the investigation of discrete and continuous fuzzy systems and outlines the similarities that exist between the two representations. The role of the fuzzy membership functions and operators are discussed and arguments that support the adoption of *algebraic* rather than *truncation* fuzzy operators are expounded. It is shown that under certain conditions the continuous and discrete fuzzy implementations are *equivalent* and that a continuous approach generally requires a smaller number of parameters as well as reducing the computational cost. The results derived for the continuous approach illustrate the strengths and weaknesses of using fuzzy networks within modelling and control systems, and show that they can reproduce exactly any bounded linear mapping. Finally, several training algorithms are proposed for which convergence to a minimum mean square error (MSE) can be proved. Training in *weight* and *rule confidence* space is compared, and it is shown that it is computationally cheaper to train the fuzzy network in weight space and use fuzzy algorithms (with their associated rule confidences) only for network initialisation, validation and verification purposes.

2. Fuzzy representation

Fuzzy logic is widely used in intelligent control to reason about vague rules which describe the relationship between imprecise, qualitative, linguistic assessments of the system's input and output states. These production rules are generally natural language representations of a human's (or expert's) knowledge, and provide an easily understood knowledge representation scheme for explaining information learnt by a computer or for initialising a particular system. A *fuzzy algorithm* is defined to be the set of these rules which describes a mapping between the system's input and output states. If the fuzzy algorithm has n input $(x_1, \ldots, x_n)^T$ and m output states $(y_1, \ldots, y_m)^T$, it can be written as m fuzzy algorithms each with n inputs and one output. These fuzzy algorithms are composed of shallow production rules of the form:

IF (x_1 is A_1^i) AND $\cdots$ AND (x_n is A_n^i)
THEN (y is B^j) c_{ij}

where A_k^i, $k = 1, \ldots, n$ and B^j are linguistic variables which represent vague terms such as *small*, *medium* or *large*. The fuzzy rule maps the antecedent, formed by the intersection (AND) of n univariate linguistic statements (x_k is A_k^i), to the consequent formed by a single univariate linguistic statement (y is B^j). It can therefore be rewritten as:

IF ($\boldsymbol{x}$ is $\boldsymbol{A}^i$) THEN (y is B^j) c_{ij}.

Associated with each rule is a variable $c_{ij} \in [0, 1]$ that denotes the *confidence* in the rule being true. A confidence of $c_{ij} = 0$ means the rule will never fire whereas if $c_{ij} > 0$ the rule will partially fire when the input is a partial member of the antecedent of the rule.

In general, the fuzzy algorithm consists of a set of these fuzzy rules which are connected together (using a union operator or the fuzzy OR) to form the rule base. For example, the rules of a fuzzy algorithm might have the form:

IF ($\boldsymbol{x}$ is $\boldsymbol{A}^1$) THEN (y is B^1) c_{11}
OR IF ($\boldsymbol{x}$ is $\boldsymbol{A}^1$) THEN (y is B^2) c_{12}
OR IF ($\boldsymbol{x}$ is $\boldsymbol{A}^2$) THEN (y is B^1) c_{21}
$\vdots$ $\vdots$
OR IF ($\boldsymbol{x}$ is $\boldsymbol{A}^p$) THEN (y is B^q) c_{pq}.

A fuzzy algorithm that maps p input sets to q output sets will have pq rules and associated confidences, many of which will be zero.

To implement a fuzzy algorithm, the fuzzy sets that represent the linguistic statements such as (x_k is A_k^i) need to be defined and the functions used to implement the fuzzy operators AND, IF($\cdot$) THEN($\cdot$) and OR must be chosen. This paper will examine various different schemes. The relationship between the input and output variables is then no longer uncertain or vague, it is a *deterministic*, non-linear multivariable function. Therefore, a distinction is made between a *fuzzy algorithm* which contains a set of imprecise, qualitative, linguistic rules, and a *fuzzy system* which refers to a *specific*, context-dependent implementation of the rules.

3. Fuzzy sets

To represent linguistic statements such as (x_k is A_k^i) Zadeh, [38], introduced the concept of a fuzzy set. A fuzzy set A is a collection of elements defined in a *universe of discourse* labelled X when the universe is continuous and X^{d} when it is discrete.

The support of a fuzzy set A is said to be *compact* if it is a (strict) subset of the original universe of discourse. For on-line, adaptive modelling and control applications it is often important to use fuzzy sets which have a compact support as this means only a small proportion of all the rules will contribute to the output (a localised response region), and only these rules need to be modified when the network is trained using instantaneous gradient descent algorithms. If the fuzzy sets are also evenly distributed across the input space, it reduces the computation time required for calculating the system's response as efficient algorithms can be developed to produce the addresses of the relevant fuzzy sets (non-zero memberships). This does not require the output of *every* membership function to be calculated as would generally occur if the membership functions did not have a compact support or were unevenly placed in the input space. Also fuzzy sets which do not have a compact support can be artificially endowed with this property by setting the membership function value to zero when it is less than some predefined value α (performing an α-cut).

Traditionally, fuzzy algorithms have been implemented using a discrete approach [19,31] where the fuzzy sets used to form a rule base are represented by membership functions defined at a set of discrete points. This representation is very flexible in that a functional relationship need not be stored, although the number of points used to represent the membership functions can be large. However, continuous fuzzy systems have recently gained in popularity, partially due to their links with certain classes of neural network. In a continuous fuzzy system, the membership functions are stored as continuous mappings that depend on a set of parameters. For example, consider using B-splines to define the kth univariate membership function $\mu_{A_k}(\cdot)$ [10]. The recursive evaluation algorithm for basis functions of order r is given

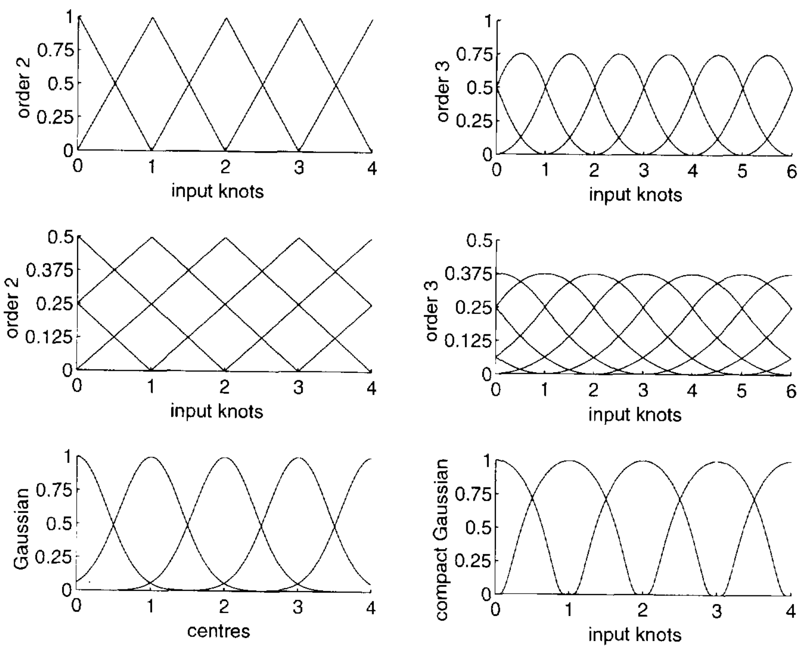


Fig. 1. Six different possible types of continuous fuzzy sets: B-splines (top), dilated B-splines (middle), Gaussian and a compact support Gaussian type functions (bottom).

by:

$$N_k^r(x) = \left(\frac{x - \lambda_{k-r}}{\lambda_{k-1} - \lambda_{k-r}}\right) N_{k-1}^{r-1}(x) + \left(\frac{\lambda_k - x}{\lambda_k - \lambda_{k-r+1}}\right) N_k^{r-1}(x)$$

$$N_k^1(x) = \begin{cases} 1 & \text{if } x \in I_k, \\ 0 & \text{otherwise,} \end{cases}$$

where $\mu_{A_k} = N_k^r$ is the kth basis function of order r, and I_k is the kth interval $[\lambda_{k-1}, \lambda_k)$. The parameters that define each basis (fuzzy membership) function are its order and the knot vector on which it is defined. An order r B-spline basis function has a knot vector of length $r + 1$, and only has a non-zero response over the r intervals that form its compact support (see Fig. 1).

Another popular choice [35] is the *Gaussian* membership function:

$$\mu_{A_k}(x) = \exp\left(-\left(\frac{x - c_k}{\sigma_k}\right)^2\right),$$

where the parameters defining the kth basis function are the centre c_k and the variance σ_k. Gaussian fuzzy sets do not have a compact support as they always have a positive response, but they can be modified (using the α-cut) so that this property is incorporated into the fuzzy sets. Alternatively, a compact support, Gaussian-type fuzzy set which is *infinitely* differentiable [36] can be used. Both these Gaussian membership functions have an input in their supports for which $\mu_A(x) = 1$ (see Fig. 1), and although this is generally required for a basis function to be a fuzzy set, it can be shown, [5, 36], that it is probably more important for the input sets to satisfy:

$$\sum_k \mu_{A_k}(x) \equiv 1 \quad \forall x.$$

If the fuzzy sets satisfy this property they are said to form a *partition of unity* or a *fuzzy partition*. Fuzzy sets that form a partition of unity generally produce smoother output surfaces which are partially invariant to shift and linear deformation of the data set [36]. Fuzzy sets which do not form a partition of unity can be endowed with this property by dividing them by the sum over all the fuzzy sets at each point. The normalised defuzzification algorithms (centre of gravity, etc.) generally produce fuzzy systems with this property, even if the original fuzzy sets do not, see Section 6, and this is an important concept for understanding the modelling abilities of a fuzzy system.

Instead of storing a functional relationship that describes $\mu_A(\cdot)$, a discrete fuzzy set stores the value at each of the points in X^{d}. A possible advantage of using discrete instead of continuous fuzzy sets is the freedom they allow the designer to choose set shapes that are not described by simple functions. Despite this, many discrete fuzzy systems are implemented using simple set shapes such as the discrete triangular or quadratic. In general, the memory requirements for a continuous fuzzy set are much less than that required for a discrete approach, although the computational cost of evaluating the membership is sometimes slightly greater.

4. Fuzzy set operations

To implement the fuzzy algorithms described in Section 2, a set of operators is required to manipulate the fuzzy quantities described by the membership functions. A fuzzy production rule is formed using three operations: intersection (AND), implication (IF($\cdot$) THEN($\cdot$)) and union (OR), and all these need to be appropriately represented within a fuzzy system. They determine how the univariate fuzzy sets interact and therefore influence the smoothness of the output surface, and this will be a prime consideration in the following discussion.

4.1. Fuzzy intersection

The fuzzy intersection (AND) of n univariate linguistic statements

$(x_1 \text{ is } \mu_{A_1^i})$ AND $\cdots$ AND $(x_n \text{ is } \mu_{A_n^i})$

is represented by the n-dimensional fuzzy membership function $\mu_{A_1^i \cap \cdots \cap A_n^i}(x_1, \ldots, x_n)$ or $\mu_{A^i}(\boldsymbol{x})$ which is defined in the product space $A_1 \times \cdots \times A_n$ by:

$$\mu_{A^i}(\boldsymbol{x}) = t\left(\mu_{A_1^i}(x_1), \ldots, \mu_{A_n^i}(x_n)\right),$$

where t is a class of functions called *triangular norms*. Triangular norms provide a wide range of functions to implement intersection but the two most popular are the *min* and the *product* operators. Generally using the *product* operator gives a *smoother* output surface, a desirable attribute in modelling and control systems.

When the univariate fuzzy membership functions $\mu_{A_k^i}(x_k)$ are expressed as functions, the multivariate membership function generated by the fuzzy intersection, $\mu_{A^i}(\boldsymbol{x})$, will require the storage of n parameter sets, one for each dimension. For example, the B-splines discussed earlier must store n knot vectors, each of length $r + 1$. The fuzzy intersection of the discrete univariate fuzzy membership functions, $\mu_{A_k^i}(x_k^{\mathrm{d}})$, can be calculated by either storing the discrete univariate functions and forming the fuzzy intersection on-line, or by storing a discrete representation of the multivariate membership function $\mu_{A^i}(\boldsymbol{x}^{\mathrm{d}})$. For the former method n discrete fuzzy sets must be stored, whereas the storage of the discrete multivariate membership function would generate an n-dimensional matrix, where the number of discrete values used to store the univariate fuzzy membership functions determines the size of the matrix.

4.2. Fuzzy implication

To represent a relation (IF *antecedent* THEN *consequent*), let the rule that maps the ith multivariate fuzzy input set $\boldsymbol{A}^i$ to the jth univariate output set with confidence c_{ij} be labelled by R_{ij}:

R_{ij}: IF ($\boldsymbol{x}$ is $\boldsymbol{A}^i$) THEN (y is B^j) c_{ij}.

Then the degree to which element $\boldsymbol{x}$ is *related* to element y by the ijth rule is represented by the membership function $\mu_{R_{ij}}(\boldsymbol{x}, y)$ that is defined in the product space $A_1 \times \cdots \times A_n \times B$ by:

$$\mu_{R_{ij}}(\boldsymbol{x}, y) = t\left(\mu_{A^i}(\boldsymbol{x}), c_{ij}, \mu_{B^j}(y)\right),$$

where again t is a triangular norm usually chosen to be the *min* or the *product* operator. It is equivalent to forming the fuzzy intersection of $(n + 1)$ univariate membership functions as again a t-norm is employed to combine the fuzzy knowledge, and from this viewpoint, c_{ij} represents the confidence in the following statements being true:

$$(x_1 \text{ is } A_1^i) \text{ AND } (x_2 \text{ is } A_2^i) \text{ AND } \cdots \text{ AND } (x_n \text{ is } A_n^i) \text{ AND } (y \text{ is } B^j)$$

or

$$\text{IF } (x_1 \text{ is } A_1^i) \text{ THEN IF } (x_2 \text{ is } A_2^i) \cdots \text{THEN IF } (x_n \text{ is } A_n^i) \text{ THEN } (y \text{ is } B^j).$$

The order in which these statements are evaluated and the position of the unknown quantity (y in this case) are immaterial. Hence for a known output and one unknown input, the knowledge stored in the above rules can be *inverted* to infer the value of the unknown variable. This is analogous to forming a fuzzy plant model (mapping present state and control to next state) and inverting the rules to infer the control necessary to move the plant from its present to the desired state [21].

To explain the differences between continuous and discrete fuzzy implication, the same arguments used for fuzzy intersection will apply (with n replaced by $n+1$). This is because a fuzzy relation is formed by applying fuzzy intersection.

4.3. *Fuzzy union*

If p multivariate fuzzy input sets A^i map to q univariate fuzzy output sets B^j, there will be pq $(n+1)$-dimensional membership functions, one for each relation. The pq relations may then be connected to form a fuzzy rule base by taking the union (OR) of the individual membership functions to form a *fuzzy relational surface*, given by:

$$\mu_R(\boldsymbol{x}, y) = \bigcup_{ij} \mu_{R_{ij}}(\boldsymbol{x}, y).$$

This operation is defined by:

$$\mu_R(\boldsymbol{x}, y) = s\,(\mu_{R_{11}}(\boldsymbol{x}, y), \ldots, \mu_{R_{1q}}(\boldsymbol{x}, y), \ldots, \mu_{R_{p1}}(\boldsymbol{x}, y), \ldots, \mu_{R_{pq}}(\boldsymbol{x}, y)),$$

where s is a class of functions called the *triangular co-norm*. Triangular co-norms also provide a wide range of suitable functions but the two most popular are the *max* and the *addition* operators.

To illustrate the differences between continuous and discrete fuzzy union consider the following single input, single output fuzzy system which has four triangular membership functions defined on each variable that represent the linguistic terms *almost zero*, *small*, *medium* and *large*, as shown in Fig. 2.

The fuzzy algorithm for this system might be given by:

```
     IF (x is almost zero)
        THEN (y is almost zero)    (c11)
OR   IF (x is small)
        THEN (y is almost zero)    (c21)
OR   IF (x is small)
        THEN (y is small)          (c22)
OR   IF (x is medium)
        THEN (y is small)          (c32)
OR   IF (x is medium)
        THEN (y is medium)         (c33)
OR   IF (x is large)
        THEN (y is large)          (c44)
```

$$\begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{pmatrix} = \begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.4 & 0.6 & 0.0 & 0.0 \\ 0.0 & 0.2 & 0.8 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

for which the rule confidence vectors associated with each input membership function have been normalised. All the fuzzy production rules that have a zero rule confidence do not influence the output of the system. The set of all the rule confidences forms the rule confidence matrix which relates each fuzzy input set to every possible fuzzy output set. Note that the rule confidences are not binary, as this allows for greater modelling flexibility (see Section 6.1).

A continuous fuzzy system represents the fuzzy algorithm by choosing the membership functions, which implement the fuzzy linguistic statements, and the fuzzy operators. The rule confidence matrix is stored *separately*. If the *product* operator is used as the t-norm, the fuzzy system will be normalised which justifies the use of the *addition* operator to implement fuzzy union. The fuzzy relational surface is shown in Fig. 3(a).

The above non-binary fuzzy algorithm is now represented by the relational surface shown in Fig. 3(b) except that this time *all* fuzzy operators are implemented using truncation operators (*min* and *max*). The surface is very regular as for each input/output pair only one rule is active. However, it also means that the surface has severe derivative discontinuities both along lines parallel to the input axes (a consequence of using triangular input/output membership functions) and along the major and minor diagonals (which is due to using the *min* intersection operator). The relational surface also has large areas where the system is not sensitive to any change in x (around 0.75 for instance), so the output of the fuzzy system will be *constant* in these regions. This implies that the overall network is insensitive to measurement noise, but in the regions where one rule is dropped from the calculation and another is substituted, the output will be extremely sensitive to the input. Therefore fuzzy systems which use truncation operators are not inherently robust, rather the information lost when they are used produces a very regular but undesirable fuzzy relational surface.

Discrete fuzzy systems store the input/output information in the form of a *fuzzy relational matrix* which is simply the continuous relational surface *sampled* at appropriate points. The relational matrix is therefore an $(n + 1)$-dimensional matrix where the number of

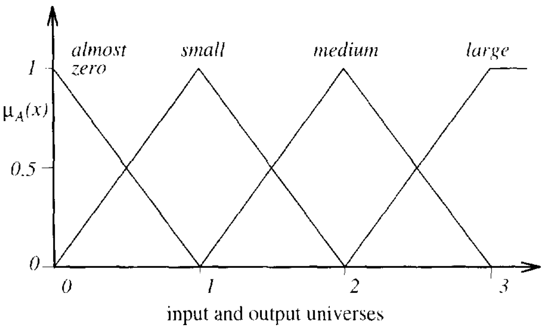


Fig. 2. Fuzzy input and output membership function definitions.

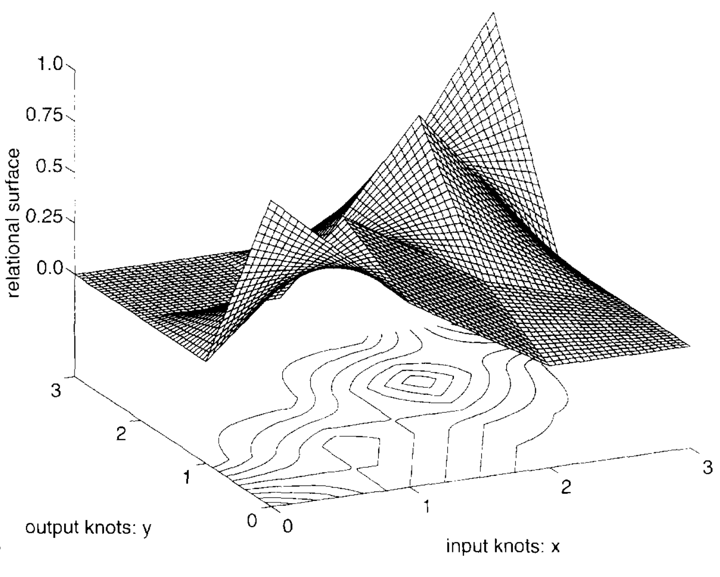


Fig. 3. The fuzzy relational surfaces and associated contour plots for a single input, single output fuzzy system with normalised rule confidence vectors where (a) truncation and (b) algebraic operators are used to implement all the fuzzy operators.

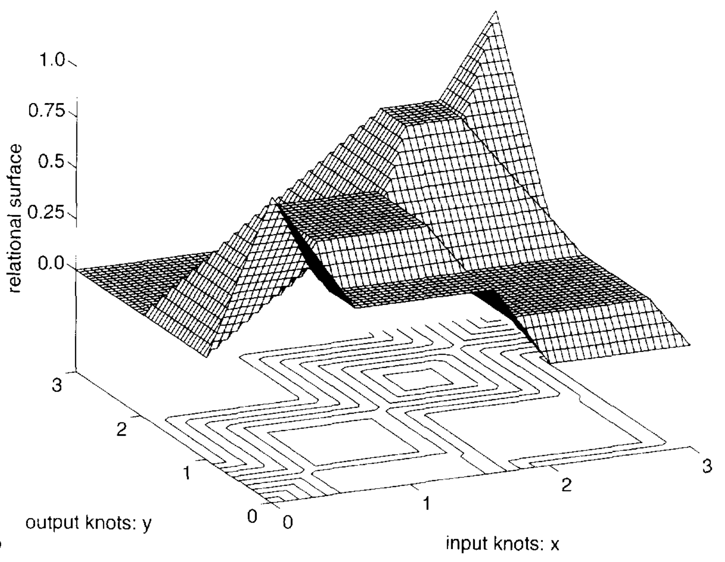


Fig. 3b.

discrete points used to represent each variable determines the size of each axis. It contains information about the shape of the fuzzy input and output sets, and the operators used to implement the underlying fuzzy logic. The ijth element is given by:

$$r_{ij} = \bigcup_{kl} \mu_{R_{kl}}(x_i^{\mathrm{d}}, y_j^{\mathrm{d}}) = \bigcup_{kl} t(\mu_{A^k}(x_i^{\mathrm{d}}), c_{kl}, \mu_{B^l}(y_j^{\mathrm{d}})) \tag{1}$$

for discrete input/output points $x_i^{\mathrm{d}}, y_j^{\mathrm{d}}$, respectively. The relational matrix attempts to approximate the form of the relational surface by sampling it at a large number of discrete points and manipulating this information directly rather than the rule confidence matrix. This can be extremely costly in terms of the storage requirements, as can be seen by considering the relational matrix shown in Fig. 4 that has been generated by the single input, single output fuzzy system represented by Fig. 3(a). Each variable is sampled at the points $\{0, 0.33, 0.67, 1.0, 1.33, 1.67, 2.0, 2.33, 2.67, 3.0\}$ which produces the following relational matrix of size (10×10). This is substantially larger than the rule confidence matrix (size (4×4)) as the relational matrix also contains information about the discrete representations of the fuzzy input and output sets used to represent the linguistic variables.

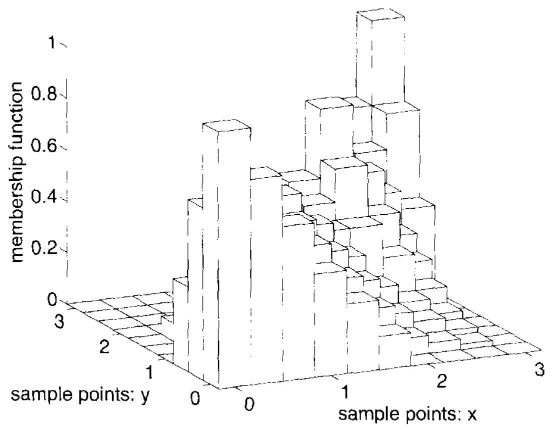
The relational matrix contains the linguistic rules used to either initialise the system or to explain its actions after training has ceased. Eq. (1) shows how the elements of the relational matrix are generated from the rule confidences and to be *consistent*, the know ledge stored in the relational matrix must be interpretable as a set of linguistic rules. This immediately imposes constraints on the method for updating neighbouring elements of the relational matrix, and this is discussed further in Section 7.

5. Compositional rule of inference

Once a fuzzy rule base has been formed, it can be used to generate an output set for a given fuzzy input set using a procedure known as *compositional rule of inference*. Let $\mu_R(x, y)$ be a fuzzy rule base, then the fuzzy output set $\mu_B(y)$ induced by the fuzzy input set $\mu_A(x)$ is given by

$$\mu_B(y) = \mu_A(x) \circ \mu_R(x, y),$$

where $\circ$ is the composition operator. Composition is the procedure that allows a fuzzy model to produce *sensible* outputs for previously unseen inputs and it will depend on whether the fuzzy sets are defined on a



3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.33	0.67	1.0
	0.0	0.0	0.0	0.0	0.09	0.18	0.27	0.4	0.53	0.67
y	0.0	0.0	0.0	0.0	0.18	0.36	0.53	0.47	0.4	0.33
2.0	0.0	0.0	0.0	0.0	0.27	0.53	0.8	0.53	0.27	0.0
	0.0	0.07	0.13	0.2	0.33	0.47	0.6	0.4	0.2	0.0
	0.0	0.13	0.27	0.4	0.4	0.4	0.4	0.27	0.13	0.0
1.0	0.0	0.2	0.4	0.6	0.47	0.33	0.2	0.13	0.07	0.0
	0.33	0.4	0.47	0.53	0.40	0.27	0.13	0.09	0.04	0.0
	0.67	0.6	0.53	0.47	0.33	0.2	0.07	0.04	0.02	0.0
0.0	1.0	0.8	0.6	0.4	0.27	0.13	0.0	0.0	0.0	0.0
	0.0			1.0			2.0	x		3.0

Fig. 4. The sampled fuzzy relational surface (top) and the corresponding relational matrix (bottom). The relational elements at the knot points correspond to the rule confidences as the input/output sets have a unity membership at these values.

continuous or a discrete universe and on the functions used to implement the fuzzy operators. Before composition can be applied, the input to the fuzzy model must first be represented as the fuzzy set $\mu_A(\mathbf{x})$. The process is called *fuzzification* and there are a variety of techniques that are used to represent the uncertainty associated with each input.

For fuzzy sets defined on a continuous universe, the composition operator is defined by

$$\mu_B(y) = s\,(t\,(\mu_A(\mathbf{x}), \mu_R(\mathbf{x}, y))), \tag{2}$$

where the s-norm is taken over *all* possible values of $\mathbf{x}$ in the domain $\mathbf{X}$, and the t-norm computes a match between two membership functions for each value of $\mathbf{x}$. When s and t are chosen to be the *integration* and the *product* operators respectively, then:

$$\mu_B(y) = \int_{\mathbf{X}} \mu_A(\mathbf{x})\,\mu_R(\mathbf{x}, y)\,\mathrm{d}\mathbf{x}$$

which for an arbitrary fuzzy input set requires an n-dimensional integral to be evaluated. *Discrete* numerical integration routines can be used to approximate the integrand $\mu_A(\mathbf{x})\,\mu_R(\mathbf{x}, y)$, and in this case the continuous and the discrete fuzzy composition algorithms are equivalent (if the sample/evaluation points are the same). If s and t are chosen to be the *max* and *min* operators the expression becomes:

$$\mu_B(y) = \max_{\mathbf{X}}\,(\min\,(\mu_A(\mathbf{x}), \mu_R(\mathbf{x}, y)))$$

which also introduces numerical difficulties since a global, non-linear optimisation problem in n dimensions must be solved to obtain the maximum over $\mathbf{X}$.

The numerical difficulties of performing continuous fuzzy composition are avoided if a *singleton* fuzzifier is used. Eq. (2) then reduces to $\mu_B(y) = \mu_R(\boldsymbol{x}^s, y)$ which is simply a *slice* taken through the relational surface and does not involve calculating an n-dimensional integral or solving an optimisation problem. It also should be noted that both implementation algorithms require knowledge of the fuzzy rule base $\mu_R(\boldsymbol{x}, y)$ which is stored as $(n+1)$ parameter sets.

Similar results can be obtained when the fuzzy sets are defined on a discrete universe except that because the input domain $\boldsymbol{X}^d$ is discrete, the compositional rule of inference is easier to calculate than the continuous case but it still requires knowledge of the discrete relational matrix. For a fuzzy singleton input, the output distribution is given by the appropriate *column* of the discrete relational matrix.

6. Defuzzification

If a single output variable is required from the fuzzy rule base rather than the fuzzy output set, $\mu_B(y)$, a process known as *defuzzification* is used to *compress* this information. The crisp output is generally obtained using a *mean of maxima* or a *centre of gravity* defuzzification strategy. It may seem that a lot of information about the uncertainty associated with the input measurements is lost when the fuzzy output set is compressed to a single value, however the *distributed* fuzzy representations (such as storing a *single* value as the membership of *several* fuzzy sets) do not always contain any *extra* information. Both defuzzification algorithms are easier to calculate if a discrete fuzzy system is used, although an extremely important insight into fuzzy systems is obtained when the centre of gravity method is applied to certain continuous fuzzy systems since it:

- allows a direct link to be made with artificial neural networks,
- provides an implementation method which greatly reduces both the computational cost and the storage requirements of the algorithm and
- explains how fuzzy systems *generalise* and the role of the fuzzification algorithm.

The results and insights do not directly apply to *all* fuzzy systems, although it has been the authors' experience that adopting any of the proposed implementation methods generally results in a smoother output surface and improved system performance.

Consider the centre of gravity defuzzification algorithm for a continuous fuzzy system:

$$y(\boldsymbol{x}) = \frac{\int_Y \mu_B(y)\, y \,\mathrm{d}y}{\int_Y \mu_B(y)\,\mathrm{d}y}.$$

Let s and t be the *addition* and *product* operators, respectively, then for the case of bounded and symmetric fuzzy output sets, suppose that the multivariate fuzzy input sets form a partition of unity, i.e. $\sum_i \mu_{A^i} = 1$ and that the ith rule confidence vector $\boldsymbol{c}_i = (c_{i1}, \ldots, c_{iq})^{\mathrm{T}}$ is normalised, i.e. $\sum_j c_{ij} = 1$, then the defuzzified output becomes

$$y(\boldsymbol{x}) = \frac{\int_X \mu_A(\boldsymbol{x}) \sum_i \mu_{A^i}(\boldsymbol{x}) w_i \,\mathrm{d}\boldsymbol{x}}{\int_X \mu_A(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}}, \tag{3}$$

where $w_i = \sum_j c_{ij}\, y_j^c$ is the *weight* associated with the ith fuzzy input set and y_j^c is the centre of the jth output set. The transformation from the weight w_i to the vector of rule confidences $\boldsymbol{c}_i$ is a many-to-one mapping, although for fuzzy sets defined by symmetric B-splines of order $r \geqslant 2$, it can be inverted in the sense that for a given w_i there exists a *unique* $\boldsymbol{c}_i$ that will generate the desired output. This will be explained further in Section 6.1 and is proved in [5]. It should also be emphasised that using weights in place of rule confidence vectors provides a considerable reduction in both the storage requirements and the computational cost, and is relevant to the discussion on training given in Section 7.

When the fuzzy input set $\mu_A(\boldsymbol{x})$ is a singleton the term $\sum_i \mu_{A^i}(\boldsymbol{x})\, w_i$ in Eq. (3) becomes constant and the numerator and denominator integrals cancel to give:

$$y^s(\boldsymbol{x}) = \sum_i \mu_{A^i}(\boldsymbol{x})\, w_i, \tag{4}$$

where $y^s(\boldsymbol{x})$ is called the fuzzy singleton output. This is an important and surprising result since $y^s(\boldsymbol{x})$ is a *linear* combination of the fuzzy input sets and does *not* depend on the choice of fuzzy output sets. It also provides a useful link between fuzzy and neural networks and allows both approaches to be treated within a unified framework, and this is discussed in Section 6.2. The reduction in the computational cost of implementing a fuzzy system in this manner and the overall algorithmic simplification is illustrated in Fig. 5.

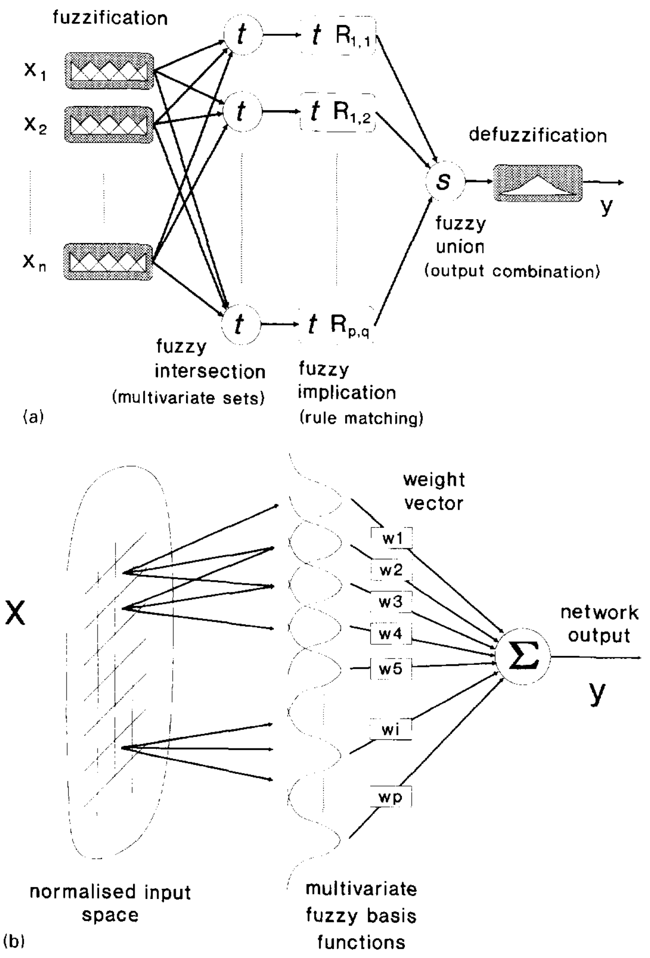


Fig. 5. An illustration of the information flow through a fuzzy system (top) and the resulting simplification (bottom) when algebraic operators are used in conjunction with a centre of gravity defuzzification algorithm, and the singleton input is represented by a crisp fuzzy set.

The analysis also illustrates how the centre of gravity defuzzification procedure *implicitly* imposes a partition of unity on the fuzzy input membership functions. Consider the above system when the fuzzy input sets do not sum to unity, which could be due to their univariate shape or the operator used to represent fuzzy intersection. The output is then given by:

$$\begin{aligned} y^s(\boldsymbol{x}) &= \frac{\sum_i \mu_{A^i}(\boldsymbol{x})\, w_i}{\sum_j \mu_{A^j}(\boldsymbol{x})} \\ &= \sum_i \mu^*_{A^i}(\boldsymbol{x})\, w_i, \end{aligned} \qquad (5)$$

where the normalised fuzzy input membership functions $\mu^*_{A^i}(\boldsymbol{x})$ form a partition of unity. This normalisation step is very important because it determines the *actual* influence of the fuzzy set on the system's output and can make previously convex sets, non-convex.

When the input to a fuzzy system is a fuzzy distribution rather than a singleton, it is possible to substitute Eq. (4) into (3) giving:

$$y(\boldsymbol{x}) = \frac{\int_X \mu_A(\boldsymbol{x})\, y^s(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x}}{\int_X \mu_A(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x}}. \qquad (6)$$

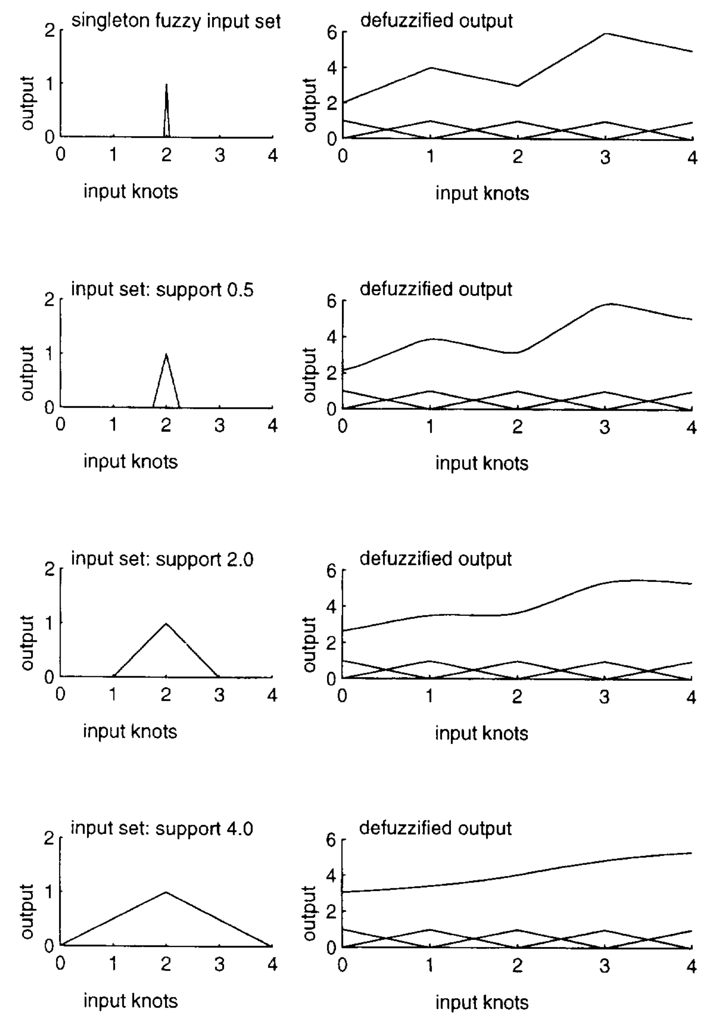


Fig. 6. Four fuzzy input sets and their corresponding defuzzified outputs, when the fuzzy rule base consists of triangular membership functions. The original triangular membership functions used to represent the linguistic terms are shown on the bottom of the graphs on the right, and it can be clearly seen that as the width of the input set increases the system becomes less sensitive to the input variable and the set shapes.

The defuzzified output is a weighted average of the fuzzy singleton outputs over the support of the fuzzy input set $\mu_A(x)$, and the effect is to *smooth* or *low pass filter* the system's output y which is illustrated in Fig. 6. It can be seen that as the width of the fuzzy input set increases, the overall output of the system becomes *less sensitive* to the shape of either the input set or the sets used to represent the linguistic terms. However, this is not always desirable as the output also becomes less sensitive to individual rules and the input variable, and in the limit as the input set shape has an arbitrarily large width (representing complete uncertainty about the measurement) the system's output is constant everywhere. It is common to see papers that use truncation fuzzy operators together with wide fuzzy input distributions, whose effect is to smooth out the small non-linear kinks that would otherwise appear in the system's output.

An important consequence of the above analysis is that using centre of area defuzzification in conjunction with the *addition* and *product* operators has reduced fuzzy composition and defuzzification to a single operation. It is no longer necessary to calculate and store $\mu_R(\boldsymbol{x}, y)$.

A similar analysis can be performed for discrete centre of gravity defuzzification where if algebraic operators are used, the defuzzified output for a singleton input becomes

$$y^{\mathrm{s}}(\boldsymbol{x}^{\mathrm{d}}) = \sum_i \mu_{A^i}(\boldsymbol{x}^{\mathrm{d}})\, w_i$$

and again, $y^{\mathrm{s}}(\boldsymbol{x}^{\mathrm{d}})$ is a weighted linear sum of the fuzzy input sets. For a general input set, $\mu_A(\boldsymbol{x}^{\mathrm{d}})$, the defuzzified output,

$$y(\boldsymbol{x}^{\mathrm{d}}) = \frac{\sum_{\boldsymbol{X}^{\mathrm{d}}} \mu_A(\boldsymbol{x}^{\mathrm{d}})\, y^{\mathrm{s}}(\boldsymbol{x}^{\mathrm{d}})}{\sum_{\boldsymbol{X}^{\mathrm{d}}} \mu_A(\boldsymbol{x}^{\mathrm{d}})}, \qquad (7)$$

is given by a weighted average of the fuzzy singleton outputs over the support of the fuzzy input set. As the number of points in $\boldsymbol{X}^{\mathrm{d}}$ increases, this quantity becomes closer to the output of the equivalent continuous fuzzy system.

6.1. *The rule confidences*

Many applications of fuzzy logic to problems in modelling and control use binary rule confidences, i.e. $c_{ij} \in \{0, 1\}$, so that the belief in the rule that maps the ith multivariate fuzzy input set to the jth univariate fuzzy output set is either one or zero. Using binary rule confidences means that only one fuzzy output set will fire for each fuzzy input set. The fuzzy algorithm will then consist of rules with the form:

IF ($\boldsymbol{x}$ is A^i) THEN (y is B^j) $\quad c_{ij} = 1$.

However, binary rule confidences impose severe limitations on the defuzzified output and restrict the class of functions that can be represented. A binary fuzzy system cannot reproduce certain functions, even if the network structure is correct. Problems also exist when there are more fuzzy input sets than output sets ($p > q$). This generally occurs for multivariable inputs since the basis functions are defined on a lattice across the input space, and their number increases exponentially with the input dimension. For $p > q$, the situation arises where several input sets could possibly map to the *same* output set.

These restrictions are overcome by allowing the rule confidences to take any value in the interval $[0, 1]$. The fuzzy system then has the flexibility to allow each input set to activate more than one output set and any defuzzified output lying in the support of the output sets can be obtained.

The main problem is then how to generate the rule confidences so that the desired behaviour can be incorporated into a fuzzy system. It was shown in Eq. (4) that the output of a continuous fuzzy system that uses algebraic operators and receives a crisp fuzzy input set is given by

$$y^{\mathrm{s}}(\boldsymbol{x}) = \sum_{i=1}^{p} \mu_{A^i}(\boldsymbol{x})\, w_i,$$

where $w_i = \sum_{j=1}^{q} c_{ij}\, y_j^{\mathrm{c}}$ is a weight that represents a local estimate of the system's output. There should be no restrictions on the weights and it is desirable for there to be no interdependency between the different input sets. This generates one constraint for each of the rule confidence vectors:

$$\hat{w}_i = \sum_{j=1}^{q} c_{ij}\, y_j^{\mathrm{c}} \qquad (8)$$

where $\hat{w}_i$ is the *desired* weight associated with each fuzzy input set. The weight can be considered as the possible *output* associated with the fuzzy input set, and it is therefore consistent to evaluate their grade of memberships in the fuzzy output sets; *fuzzifying* its value. This constraint ensures that the data stored and retrieved from the fuzzy system is *equivalent*, and unless relationships such as these are embodied in the learning rules, convergence, rates of convergence and stability results are very difficult to establish. The rule confidences are still not uniquely determined as Eq. (8) gives only one equation in q unknowns, therefore other constraining equations must be generated. A second constraint which has been assumed to exist is that the rule confidence vectors are *normalised*, and so the following expression must hold:

$$\sum_{j=1}^{q} c_{ij} \equiv 1 \quad \text{for each } i. \qquad (9)$$

It is also generally required that the rule base should be as *sparse* as possible, to minimise the computational cost and storage requirements, and to keep the rule base *simple* for user verification and validation. This can be achieved by restricting the non-zero rule confidences to coincide with the fuzzy output sets for which the desired weight has a non-zero degree of membership. The rule confidence vector should also be *convex* (a single maximum value) as this will ensure that the fuzzy output distribution is also convex.

Consider the commonly occurring case when there are two overlapping sets on the output universe and each desired weight has a non-zero degree of membership in two output sets. For convenience assume that these are the first two output sets, then Eqs. (8) and (9) give:

$$\hat{w}_i = c_{i1} y_1^c + c_{i2} y_2^c,$$
$$1 = c_{i1} + c_{i2}.$$

There exists two independent equations in two variables, and solving for the rule confidences gives:

$$c_{i1} = \frac{y_2^c - \hat{w}_i}{y_2^c - y_1^c}, \qquad c_{i2} = \frac{\hat{w}_i - y_1^c}{y_2^c - y_1^c}. \tag{10}$$

These functions are simply B-splines of order 2 or (equivalently) the commonly used triangular fuzzy membership functions. Therefore, if the fuzzy output sets are B-splines of order 2, the rule confidences can be calculated from:

$$c_{ij} = \mu_{B^j}(\hat{w}_i).$$

This provides a *direct, invertible* relationship between the weights, rule confidences and fuzzy output sets. Given a particular weight to store, a rule confidence vector can be evaluated using the fuzzy output sets which produces the original weight when the fuzzy information is defuzzified. It is possible to use other fuzzy output membership functions which have different shapes, although a simple and consistent method must be found for generating the rule confidences, and to the authors' knowledge none have been proposed apart from this algorithm.

For higher order fuzzy output sets it might be hoped that this relationship can be generalised, and indeed if the fuzzy output membership functions are *symmetric* B-splines of order $\geqslant 2$, the relation in Eq. (8) is satisfied. This is proved in [5]. The B-spline membership functions are also normalised and produce convex membership values, thus the rule confidences produced satisfy all of the constraints.

The *invertible* relationship which exists between weights and rule confidences is very important as it makes it possible to implement and train the network in weight space (with a considerable reduction in the computational cost), while explaining their responses using linguistic rules and the associated rule confidences. These systems are therefore *transparent* to the designer, and the network's performance (fuzzy rules) can be modified after training and re-implemented in weight space.

6.2. *A comparison with neural networks*

In the simplest form, fuzzy systems calculate their response by taking a *linear* combination of the input membership functions, which is a strategy adopted by several neural networks commonly used within learning modelling and control architectures. Radial basis function neural networks were first proposed by Broomhead and Lowe in 1988 [4] and the network's output is given by:

$$y(\boldsymbol{x}) = \sum_{i=1}^{p} f_i(\|\boldsymbol{x} - \boldsymbol{c}_i\|_2)\, w_i,$$

where $\boldsymbol{c}_i$ is the centre of the ith radial basis function f_i, and $\|\cdot\|_2$ denotes the common Euclidean norm. Different univariate functions f_i, produce networks which generalise differently [6], although one of the most popular is the standard multivariate Gaussian function which can be obtained by *multiplying* together n univariate Gaussian functions. As noted in [34, 35], this is equivalent to forming the fuzzy AND of n univariate Gaussian membership functions. Therefore there is a direct equivalence between radial basis function networks and fuzzy systems.

The Cerebellar Model Articulation Controller (CMAC) [1, 5, 20, 33] was originally proposed by Albus in the mid-1970s, and in its most basic form, is a distributed look-up table which locally generalises. The network's response can again be expressed as:

$$y(\boldsymbol{x}) = \sum_{i=1}^{p} f_i(\boldsymbol{x})\, w_i,$$

where the non-linear function $f_i(\cdot)$ can take various forms. It can be expressed as $f_i(\|x - c_i\|_\infty)$ about a centre c_i, or it can be formed by combining n univariate basis functions (fuzzy membership functions) using a *product* or *min* operator. Again there is a direct relationship between the CMAC and fuzzy systems. The CMAC possesses a special feature which ensures that the number of basis functions (fuzzy rules) which contribute to the output is *not* a function of the input space dimension, so the cost of calculating the network's response depends *linearly* on the input space dimension. This method of distributing the basis functions can be used as an even, sparse coarse coding algorithm for fuzzy systems. The basic B-spline algorithm can also be expressed as a linear combination of piecewise polynomial basis functions which are formed by multiplying together n univariate basis functions [5].

This relationship between neural networks and fuzzy systems also unifies two apparently different schemes developed in each field. Sugeno [30] proposed a type of fuzzy system where instead of the consequence of the production rule being a linguistic term, a *numerical function* is stored. These production rules have the form:

$$\begin{aligned}&\text{IF } (x_1 \text{ is } A_1^i) \text{ AND } \cdots \text{ AND } (x_n \text{ is } A_n^i)\\ &\quad\text{THEN } y = w_0^i + w_1^i x_1 + \cdots + w_n^i x_n\end{aligned}$$

where w^i is the *linear* parameter (weight) vector associated with the ith multivariate fuzzy input set. The fuzzy input sets provide a *non-linear* scheduling of the linear parameters, while the output is smooth due to the linear functions which only have a local influence. It should be noted that the conventional fuzzy systems that have been extensively investigated in this paper (algebraic operators, normalised fuzzy membership functions and centre of area defuzzification) are a *special* case of these rules, as the output associated with each antecedent is simply a single weight, w_0^i, rather than a linear function. This type of representation has also been proposed for normalised Gaussian radial basis functions [29] where the system's output is given by:

$$y(x) = \frac{\sum_{i=1}^{p}(w_0^i + w_1^i x_1 + \cdots + w_n^i x_n) f_i(\|x - c_i\|_2)}{\sum_{j=1}^{p} f_j(\|x - c_j\|_2)}.$$

There is no first order term in the Taylor series expansion of a Gaussian function, so this modification to the standard radial basis function representation enables it to model a locally linear function using only a small number of basis functions. This is similar to Sugeno's fuzzy system, [30] where trapezoidal membership functions are generally employed. These membership functions are constant near their centre, so the linear function means that the overall system can model locally linear functions using a small number of rules [13, 23].

The relationship between neural and fuzzy systems is not solely restricted to singleton inputs. As shown in Eqs. (6) and (7), when the input is represented by a fuzzy distribution, the defuzzified network output is calculated by taking a locally weighted average of the outputs which correspond to singleton inputs. It is irrelevant whether a neural or a fuzzy system is used to store the desired function, rather research should be concentrating on uncertainty representations (membership function shapes) and knowledge processing operations (truncation and algebraic operators).

7. Training

An important step of the fuzzy model or controller design procedure is knowledge elicitation, where the fuzzy rule base is initialised or constructed. Like expert systems there are many reasons why it may not be possible, or desirable, to use the rules described by a human operator [8], so learning algorithms must be used to build up knowledge about the plant and its environment, and to adapt to new and unexpected changes. Self organising fuzzy controllers were first proposed in the late 1970s [27], in an attempt to reduce the dependency of the overall system on the environment, and although the approach was successful for many systems it is based on heuristic learning rules and little can be proved about parameter convergence or the overall system's stability. This paper has investigated the modelling abilities of fuzzy systems from a theoretical viewpoint and a similar approach will be used to investigate several learning rules, and to relate their form to more commonly used heuristic training rules.

Assume there exist a set of training pairs $\{x(t), \hat{y}(t)\}_{t=1}^{L}$ where $\hat{y}(t)$ is the *desired* output for a given input $x(t)$. When a desired output is available, the normal approach to training is to use a *supervised* technique that compares the actual and desired outputs and uses the resulting error to update the fuzzy system parameters. Supervised training algorithms can be divided into two distinct groups according to whether real-time operation is a prime concern. *Batch* learning rules use information provided by *all* the training pairs when adjusting the fuzzy system parameters whereas *instantaneous* training algorithms use only a single example or a small subset of the training data at each update.

The fact that the fuzzy singleton output is formed from a weighted sum of the fuzzy input sets:

$$y^s(\boldsymbol{x}) = \sum_i \mu_{A^i}(\boldsymbol{x}(t))\, w_i = \sum_i \mu_{A^i}(\boldsymbol{x}(t)) \sum_j c_{ij} y_j^c$$

means that *simple*, *linear* training algorithms can be derived for either the weights or the rule confidences for which convergence can be proven, and rates of convergence estimated. To adapt the linear parameters, an error cost function must first be constructed. For practical reasons, the MSE is often employed since it leads to training algorithms that are simple to implement and solutions that are relatively insensitive to small errors in the training data. It is given by:

$$J = E\left(\varepsilon_y^2(t)\right) = E\left((\hat{y}(t) - y(t))^2\right),$$

where E is the expectation operator taken over t for the set of training pairs. When $\varepsilon_y^2(t)$ is a *discrete* random variable which takes values given by $\varepsilon_y^2(1), \ldots, \varepsilon_y^2(L)$, then:

$$J = E\left(\varepsilon_y^2(t)\right) = \frac{1}{2}\sum_{t=1}^{L}(\hat{y}(t) - y(t))^2,$$

where the $1/2$ has been added for mathematical convenience. These cost functions use all the available training data, whereas instantaneous learning rules employ instantaneous approximations such as:

$$J(t) = \varepsilon_y^2(t) = (\hat{y}(t) - y(t))^2.$$

The cost function and the learning rule used both influence the value of the final weight vector, although it can be verified that when batch and instantaneous gradient descent learning rules are used to train the fuzzy system based on the two cost functions described above, the optimal solutions are equivalent.

Before proceeding to derive the learning algorithms, it is worthwhile considering what training is trying to achieve and how the rules work. The overall aim is to adapt the parameters (weights or rule confidences) so that they move closer to their optimal values, which is achieved by measuring the output error and estimating the corresponding parameter errors. Although parameter error and output error measures are related, their behaviour can be quite different (misleading) especially when instantaneous assessments are made of the output error cost function. The ability of a system to *generalise* depends on its parameter errors, and the MSE is generally a poor measure of this quantity. Alternatively, the instantaneous performance of a system depends on the output error and even though parameter errors may be decaying quickly this does not always imply that the output error is being reduced. This is not important for batch learning algorithms where the prime concern is parameter convergence, but for on-line adaptation the transient performance of the training rules must be considered. The most successful on-line learning rules consider both types of errors when the parameters are being updated, because if they are biased towards one scheme their performance is poor in certain situations.

7.1. Adapting weights, rule confidences and relational elements

In the previous section it was argued that a fuzzy system can be implemented more efficiently if weights are used to represent the knowledge instead of rule confidences. The memory requirements are lower and a smaller number of arithmetic steps are necessary to calculate its response. It is now explained why the weight representation is also desirable for learning fuzzy systems. The reasons why centre on the relationship between each weight and the corresponding rule confidence vector associated with each basis function. Since q rule confidences are used to store the information that can be represented by a single variable, it immediately implies that the rule confidences are *linearly dependent*. This can be illustrated by considering the following three normalised rule confidence vectors which all defuzzify to the same output value:

$$\boldsymbol{c}_i^1 = (0.0, 1.0, 0.0)^{\mathrm{T}}, \qquad \boldsymbol{c}_i^2 = (0.5, 0.0, 0.5)^{\mathrm{T}},$$
$$\boldsymbol{c}_i^3 = (0.4, 0.2, 0.4)^{\mathrm{T}}.$$

If the linguistic descriptions assigned to each fuzzy set are *negative small*, *almost zero* and *positive small*, the corresponding weight will be zero for either $\boldsymbol{c}_i^1, \boldsymbol{c}_i^2$ or $\boldsymbol{c}_i^3$. However, the second and third rule confidence vectors produce contradictory rules (the discrete set of rule confidences is *non-convex*) as they require the output to be both *negative small* and *positive small* but not to be *almost zero*. These apparent inconsistencies arise because the elements of the rule confidence vectors are linearly dependent and the only constraints which they are required to satisfy are that $c_{ij} \in [0,1], \forall i,j$ and those given in Eqs. (8) and (9). These equations do not uniquely determine the rule confidences for an arbitrary fuzzy system, and unless extra conditions are imposed on the system (such as evaluating their value by finding the membership of the desired weight of the fuzzy output sets), an infinite number of possible solutions will occur, many of which are *logically inconsistent*. These problems do not occur if the weights are trained directly and the rule confidences are only generated whenever the designer wishes to verify and validate the rule base.

It is possible to generate simple learning rules which train the rule confidences directly and ensure *output learning equivalence* between systems whose weights and rule confidences are adapted [5]. However, to be consistent the rule confidences should satisfy the three constraints mentioned above as well as the rule confidence vectors being sparse and convex, otherwise it is difficult to verify and validate the rule base once training has ceased. This imposes extra constraints on an otherwise simple optimisation problem and generates complex training algorithms. Thus, all the training algorithms described in the following sections update the weights directly, after which fuzzy rules can be inferred using the algorithm given in Section 6.1.

The discrete fuzzy systems can be obtained by sampling the corresponding continuous networks. Therefore when information is stored in a discrete system, the changes made to the relational matrix should reflect the alterations made to the rule confidence matrix. This implies that every discrete relational element which contributes to the output should be updated and the update rule should be highly constrained so that linguistic rules can be inferred from the updated relational matrix. The only reason why it may be desirable to train the relational matrix elements directly is because the overall system has more parameters and is potentially more flexible. However, this would imply that the relational matrix elements could be updated independently and would destroy the structure of the stored linguistic rules. The number of elements which have to be updated at each timestep again makes it desirable to train the weights directly, and to generate the relational matrix when required.

7.2. *Batch training algorithms*

When searching for the set of weights that minimises the MSE, each iteration of a batch training algorithm will use information provided by all the training pairs. Let $\boldsymbol{w} = (w_1, \ldots, w_p)^{\mathrm{T}}$ be the weight vector and $\boldsymbol{a}(t) = (\mu_{A^1}(\boldsymbol{x}(t)), \ldots, \mu_{A^p}(\boldsymbol{x}(t)))^{\mathrm{T}}$ the vector of transformed inputs, then the MSE for a fuzzy system is given by

$$J = \frac{1}{2}\sum_{t=1}^{L}(\hat{y}(t) - \boldsymbol{a}^{\mathrm{T}}(t)\boldsymbol{w})^2.$$

After expansion, this becomes

$$J = \frac{\boldsymbol{w}^{\mathrm{T}}\boldsymbol{R}\boldsymbol{w}}{2} - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{p} + \frac{\hat{\boldsymbol{y}}^{\mathrm{T}}\hat{\boldsymbol{y}}}{2}, \tag{11}$$

where $\boldsymbol{R} = \sum_{t=1}^{L}\boldsymbol{a}(t)\boldsymbol{a}^{\mathrm{T}}(t)$ is called the autocorrelation matrix, $\boldsymbol{p} = \sum_{t=1}^{L}\hat{y}(t)\boldsymbol{a}(t)$ is the cross correlation vector and $\hat{\boldsymbol{y}} = (\hat{y}(1), \ldots, \hat{y}(T))^{\mathrm{T}}$ is the vector of desired outputs. Eq. (11) is a non-negative, quadratic function of the weight vector that has a stationary point at $\boldsymbol{w}^{\star}$, where each element of the gradient vector $\partial J/\partial \boldsymbol{w}$ is identically equal to zero, i.e. it must hold that

$$\frac{\partial J}{\partial \boldsymbol{w}} = \boldsymbol{R}\boldsymbol{w}^{\star} - \boldsymbol{p} = 0.$$

Consequently, a stationary point must satisfy the linear system of normal equations

$$\boldsymbol{R}\boldsymbol{w}^{\star} = \boldsymbol{p}. \tag{12}$$

The autocorrelation matrix is by definition a real, symmetric, positive semi-definite matrix. If $\boldsymbol{R}$ is also non-singular, it can be inverted and the normal equations have a unique solution given by $\boldsymbol{w}^{\star} = \boldsymbol{R}^{-1}\boldsymbol{p}$. The MSE will then have a single, global minimum in weight space. By contrast, if $\boldsymbol{R}$ is singular there exist an infinite number of solutions and the MSE has an infinite number of global minima. Thus, the training problem can be viewed as either minimising the quadratic function (11) or solving the linear system of normal equations (12).

Methods for solving linear systems can be classified as being *direct* or *iterative*. Direct linear solvers perform a well defined sequence of arithmetic operations and it is known that after a certain number of steps the solution will be obtained. Iterative methods however, proceed by making a sequence of estimates of the solution and can be further subdivided according to whether or not they use gradient information. Gradient methods are preferred for training neural or fuzzy systems partly for historical reasons but also because they offer some advantages when compared with direct methods since

- they are normally more efficient for large order systems;
- implementation is easier;
- advantage can be taken of approximate solutions;
- low accuracy solutions can be easily obtained.

Most gradient methods for solving a general linear system are two stage iterative descent methods. They define a sequence of points $\{w(k)\}$ that impose the descent condition $J(w(k+1)) < J(w(k))$ at each iteration. At iteration k, the first stage selects a search direction $u(k)$ and the second stage determines the step size $\alpha(k)$. The solution at iteration $k+1$ is then given by the update

$$w(k+1) = w(k) + \alpha(k)u(k).$$

The methods differ according to the procedure by which they compute their search direction. Most also introduce vector $r(k)$, the residual for the trial solution $w(k)$, i.e. $r(k) = Rw(k) - p$. This measures the error in the trial solution and is often used as a test for convergence. A reliable criterion to terminate the iterations is

$$\|r(k)\|/\|p\| < \varepsilon,$$

where ε is a user-defined tolerance that can be adjusted according to the required solution accuracy.

Training therefore requires solving the linear system of normal equations (12) but for fuzzy sets defined by B-splines, the autocorrelation matrix R will be sparse. A sparse matrix is one that has a small proportion of its elements that are non-zero and by adopting some form of sparse storage scheme a considerable saving in both storage and computation can be obtained. The saving is made possible by not storing the zero elements and hence avoiding all trivial arithmetic operations. B-splines of order r will have (at most) r^n elements of $a(t)$ that are non-zero and it can be shown that each row of R will then have (at most) $(2r-1)^n$ non-zero elements. To illustrate the percentage sparsity that this involves, consider a fuzzy system with multivariate input sets formed by the intersection of seven order r univariate B-splines on each input axis. Table 7.2 shows the percentage sparsity obtained for different values of n and r where the maximum number of non-zero elements in each row of R is denoted by nnz and the number of row elements is ne. The table illustrates that for a fixed order the percentage sparsity increases with the input dimension and that for a fixed n, as the order increases and more B-splines overlap, the percentage sparsity is reduced.

Table 1
Percentage sparsity for different n and r.

n	r	nnz	ne	$(ne - nnz)/ne$
1	2	3	7	57%
	3	5	7	29%
2	2	9	49	82%
	3	25	49	49%
3	2	27	343	92%
	3	125	343	64%
4	2	81	2401	97%
	3	625	2401	74%

The conjugate gradient method is particularly recommended for the solution of large, sparse linear systems since it only references R through its multipli cation by the search direction u. There are no problems with sparse matrix fill-in. In the conjugate gradient method, a sequence of search directions $\{u(k)\}$ are constructed that are mutually conjugate with respect to R and hence satisfy the condition $u^T(k)Ru(j) = 0, \forall k \neq j$. With perfect arithmetic an exact answer should be obtained after at most p iterations and so, strictly speaking, it belongs to the class of direct methods. However, in cases where R is well conditioned with many nearly degenerate or clustered eigenvalues, convergence to the required accuracy can occur in much less than p iterations. This will be true if R does not differ very much from the identity matrix. In other cases, where the condition number of $\boldsymbol{R}$ is large and its eigenvalues are all distinct and evenly spread, the accumulation of rounding error can affect the arithmetic to such an extent that many more than p iterations are required. For this latter case, the conjugate gradient method in its simplest form performs very poorly and techniques to improve the convergence rate must be found. A common approach is to *precondition* $\boldsymbol{R}$ by a symmetric positive definite matrix $\boldsymbol{P}$ that is easy to invert [28]. The system of normal equations is then solved indirectly by solving

$$\boldsymbol{P}^{-1}\boldsymbol{R}\boldsymbol{w} = \boldsymbol{P}^{-1}\boldsymbol{p}. \qquad (13)$$

If the condition number of $\boldsymbol{P}^{-1}\boldsymbol{R}$ is much less than $\boldsymbol{R}$ or its eigenvalues are better clustered, then the conjugate gradient method will solve Eq. (13) much faster than Eq. (12). The preconditioned conjugate gradient algorithm is written as follows.

Conjugate gradient

```
choose $\boldsymbol{w}(0), \boldsymbol{r}(0) = \boldsymbol{p} - \boldsymbol{R}\boldsymbol{w}(0), \boldsymbol{u}(0) = \boldsymbol{P}^{-1}\boldsymbol{r}(0)$
for $k = 1, 2, \ldots$
    $\alpha(k) = \boldsymbol{r}^{\mathrm{T}}(k)\boldsymbol{P}^{-1}\boldsymbol{r}(k)/\boldsymbol{u}^{\mathrm{T}}(k)\boldsymbol{R}\boldsymbol{u}(k)$
    $\boldsymbol{w}(k+1) = \boldsymbol{w}(k) + \alpha(k)\boldsymbol{u}(k)$
    $\boldsymbol{r}(k+1) = \boldsymbol{r}(k) - \alpha(k)\boldsymbol{R}\boldsymbol{u}(k)$
    $\beta(k) = \boldsymbol{r}^{\mathrm{T}}(k+1)\boldsymbol{P}^{-1}\boldsymbol{r}(k+1)/\boldsymbol{r}^{\mathrm{T}}(k)\boldsymbol{P}^{-1}\boldsymbol{r}(k)$
    $\boldsymbol{u}(k+1) = \boldsymbol{P}^{-1}\boldsymbol{r}(k+1) + \beta(k)\boldsymbol{u}(k)$
```

where $\boldsymbol{r}(k)$ is called the residual vector for iteration k. The *perfect* preconditioner is obviously $\boldsymbol{P} = \boldsymbol{R}$ but since this requires solving the linear system of normal equations, it is of no use. A *good* preconditioner is one that approximates $\boldsymbol{R}$ well enough such that the cost of computing $\boldsymbol{P}^{-1}\boldsymbol{r}(k)$ is more than offset by the improved convergence rate of the conjugate gradient method. There are a large number of possibilities, some quite simple and others sophisticated. The simplest is *diagonal* preconditioning which just scales $\boldsymbol{R}$ so that its diagonal elements are identically equal to one. A more elaborate choice is *incomplete Cholesky* preconditioning which performs a Cholesky factorisation on the non-zero elements of $\boldsymbol{R}$. Whichever choice is made, it is generally accepted that for large, sparse linear systems, preconditioned conjugate gradients should always be used.

Also important is the behaviour of the conjugate gradient algorithm in the *rank deficient* case. The autocorrelation matrix $\boldsymbol{R}$ is rank deficient when the basis functions that define the fuzzy input sets have a compact support and are placed in regions of the input space where there exist no training pairs, or where the data is very sparse. The weights associated with these basis functions then have the freedom to take arbitrary values and by adjusting the remaining network weights the fuzzy network output will not change. This means there exist an *infinite* number of different optimal weight vectors and the final solution obtained depends on the chosen starting point. Thus, it is pertinent to investigate which optimal weight vector is most desirable, and how it is affected by the initial value of the weights.

It can be shown that a fuzzy network generalises more sensibly if the size of the optimum weight vector $\|\boldsymbol{w}^{\star}\|_2$ is minimised, that is the training problem should locate the minimum norm solution. Fortunately, both steepest descent and conjugate gradient converge to the *minimum norm solution* provided that $\boldsymbol{w}(0) \in Range(\boldsymbol{R})$, which holds if $\boldsymbol{w}(0) = 0$. Therefore, it is recommended that in the absence of a better starting point, the weight vector should be initialised to zero.

7.3. Instantaneous training algorithms

If the fuzzy system is required to self organise in real time, an *instantaneous* training algorithm must be adopted. Such algorithms adjust the fuzzy system's parameters using information provided by only a small subset of the training pairs and by making an estimate of the MSE at time t. If only the current training pair is used, the (unbiased) MSE estimate is given by

$$J(t) = \varepsilon_y^2(t) = (\hat{y}(t) - y(t))^2.$$

Most instantaneous training algorithms are two stage, iterative, descent methods. They define a sequence of points $\{\boldsymbol{w}(t)\}$ that impose the descent condition $J(\boldsymbol{w}(t+1)) < J(\boldsymbol{w}(t))$ and at time t the first stage selects the search direction $\boldsymbol{u}(t)$ and the second stage determines the step size $\alpha(t)$. The solution at time $(t+1)$ is given by the update

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \alpha(t)\boldsymbol{u}(t).$$

In contrast to the previous section, where each batch training algorithm used the same step size calculation and was characterised by the procedure used to compute the search direction, all instantaneous training algorithms described below update the weight vector along the same $\boldsymbol{u}(t)$ and are characterised by the way they set the step size.

The instantaneous learning rules attempt to minimise the true MSE cost function and to reduce the parameter error using only a small amount of performance related information at each update step. Reducing the instantaneous output error also reduces the parameter error, but it can result in the true MSE increasing. Similarly it may be that the parameter error is reduced but the instantaneous output error increases. An adaptive system which is based solely on minimising the current output error can also produce very slow parameter convergence especially when the inputs are highly correlated [5, 29]. Therefore it is sometimes necessary to store a (small) set of relevant training data and use the appropriate data pair at each update step, rather than simply using the current training example [26].

For most instantaneous training algorithms, $\boldsymbol{u}(t)$ is chosen to be the direction of steepest descent but as the only information available at time t is the instantaneous estimate of the MSE gradient

$$\frac{\partial J(t)}{\partial \boldsymbol{w}} = \varepsilon_y(t)\boldsymbol{a}(t)$$

at each time step, the *fixed* step size (learning rate) α is taken along the search direction $-\partial J(t)/\partial \boldsymbol{w}$, and so the weight update is given by

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \alpha\varepsilon_y(t)\boldsymbol{a}(t).$$

There are two popular instantaneous training algorithms; the least mean square (LMS) and the normalised least mean square (NLMS) [37], which differ according to the procedure used to set α.

With the LMS algorithm the update for each weight vector element becomes

$$w_i(t+1) = \begin{cases} \hat{y}_i & \text{if } w_i(t+1) \text{ is not initialised,} \\ w_i(t) + \alpha\varepsilon_y(t)a_i(t) & \text{otherwise,} \end{cases}$$

where the heuristic that sets an uninitialised weight vector to the desired output vector is justified when the fuzzy input sets have a compact support and are normalised. Using this learning rule, the output error after updating $\underline{\varepsilon}_y(t)$ becomes

$$\underline{\varepsilon}_y(t) = (1 - \alpha\|\boldsymbol{a}(t)\|_2^2)\varepsilon_y(t)$$

and stable learning (i.e. $\underline{\varepsilon}_y(t) < \varepsilon_y(t)$) will occur if $\alpha \in (0, 2/\|\boldsymbol{a}(t)\|_2^2)$. A possible drawback of using the LMS algorithm is that the reduction in output error depends on the size of the transformed input vector and if the variance in magnitude is large, small values of α are required for stable learning. This can greatly increase the time taken for training.

Alternatively, the NLMS algorithm may be derived where the dependency condition on the size of the transformed input when setting α is removed. The update is given by

$$w_i(t+1) = \begin{cases} \hat{y}_i & \text{if } w_i(t+1) \text{ is not initialised,} \\ w_i(t) + \dfrac{\alpha\varepsilon_y(t)}{\|\boldsymbol{a}(t)\|_2^2}a_i(t) & \text{otherwise,} \end{cases}$$

and this time the output error after updating becomes

$$\underline{\varepsilon}_y(t) = (1 - \alpha)\varepsilon_y(t).$$

Stable learning is assured if $\alpha \in (0, 2)$ which means the reduction in output error no longer depends on $\|\boldsymbol{a}(t)\|_2$. A potential problem with using NLMS occurs when $\|\boldsymbol{a}(t)\|_2 < \varepsilon$ since the step taken along search direction $\boldsymbol{a}(t)$ is inversely proportional to ε. If the output error does not depend on $\|\boldsymbol{a}(t)\|_2$, the magnitude of the weight change can become unbounded. This learning rule can also be derived using Lagrange multipliers or by using singular valued decomposition to solve a single example, p parameter matrix inversion problem [11].

The advantage of both algorithms is the economy of computation, especially when the transformed input vectors are sparse. The recommended update is simply a scalar multiplied by a transformed input vector and for fuzzy input sets defined on a compact support, only those weights that contribute to the output are updated. The disadvantage is the difficulty in choosing values of α appropriate for the training problem to be solved.

The algorithms can be given a simple geometric interpretation. A fuzzy network will exactly store the current training pair $(\boldsymbol{x}(t), \hat{y}(t))$ if the weight vector that solves

$$\hat{y}(t) = \boldsymbol{a}^{\mathrm{T}}(t)\boldsymbol{w}$$

is obtained. This equation generates a $(p-1)$-dimensional *solution hyperplane* in weight space that has a normal parallel to $\boldsymbol{a}(t)$. The LMS and NLMS algorithms use $\boldsymbol{a}(t)$ as a search direction and whether or not the weight update lies *on* the solution hyperplane is determined by the choice of step size. The current training pair will be stored exactly if the step size is set to

$$\alpha = \begin{cases} 1/\|\boldsymbol{a}(t)\|_2^2 & \text{for the LMS update,} \\ 1 & \text{for the NLMS update.} \end{cases}$$

For smaller values of α, the new weight vector does not reach the solution hyperplane while larger values will overshoot. Because the search directions are perpendicular to the solution hyperplane, the LMS and NLMS updates obey the principle of *minimal disturbance*, i.e. the weight change is the smallest that will store the current training pair exactly and there is minimal interference with the information already stored.

A proof of convergence for both algorithms can be obtained by assuming there is no modelling error or measurement noise. For this unrealistic case, the weight vector converges to a value that can exactly reproduce the training set. Although a convergence proof is very important, the algorithms are only effective if they converge rapidly, and this depends on the relative orientations of the solution hyperplanes. Fast convergence rates occurs when the transformed input vectors are mutually orthogonal since the information stored at each update will not overwrite that stored previously. Parameter convergence is then obtained after a finite number of iterations. However this condition is only achieved when the weight vector elements mapped to by each transformed input are unique, so there exists at most one input lying in the support of each basis function. But to generalise sensibly, especially in the case of training pairs that are corrupted by noise, it is both necessary and desirable to have several training pairs lying in the support of each basis function, even though this produces non-orthogonal transformed input vectors. In contrast, slow convergence rates will be obtained when the transformed input vector vectors are nearly parallel (or ill-conditioned).

For a fuzzy network, fast initial learning occurs when the training pairs are presented randomly and the fuzzy input sets are defined using B-splines. This is because the transformed input vectors are sparse and for many of the training pairs presented during the first stages of learning, they are mutually ortho gonal. Hence learning information about one part of the input space does not interfere with that stored in other parts. However, as training proceeds most transformed inputs map to the same weight vector elements and information learnt during previous updates is partially destroyed.

When there exists modelling error or measurement noise, an optimal weight vector that exactly reproduces the training set does not exist. Instead, the trained weight vector will lie in a *minimal capture zone* whose size and shape depends on the fuzzy network structure and the distribution of the training pairs [5, 25]. The mismatch can be filtered by using stochastic approximation where two simple changes to the LMS and NLMS algorithms are proposed; an individual learning rate is assigned to each basis function and the learning rate is reduced through time as the confidence in a particular weight increases. Let the ith weight have a learning rate $\alpha_i(t)$ then the necessary conditions for the weight vector to converge are given by

$$\alpha_i(t) > 0,$$

$$\sum_{t=1}^{\infty} \alpha_i(t) = \infty,$$

$$\sum_{t=1}^{\infty} \alpha_i^2(t) < \infty.$$

One function that satisfies these constraints is

$$\alpha_i(t) = \frac{\alpha_1}{1 + (t_i/\alpha_2)},$$

where α_1 and α_2 are positive constants that set the initial learning rate and rate of decay respectively and t_i is the number of times that the ith weight has been updated. These modifications allow the fuzzy network to retain a fast initial convergence rate while, in the long term, filtering out measurement and modelling noise. Stochastic approximation however does require a stationary training set otherwise $\alpha_i(t)$ must be reset to a relatively large value at regular intervals.

8. Conclusions

This paper has presented a computationally efficient method for implementing an adaptive fuzzy system as well as comparing it with more conventional schemes. The approach developed has been based on a *continuous* representation and it has been shown that discrete fuzzy systems are simply approximations to these networks. The use of fuzzy membership functions that form a partition of unity and algebraic operators has been proposed, and it has been shown that this leads to systems which produce smoother outputs, have smaller memory requirements and a reduced computational cost. The theory derived also makes it possible to interpret how fuzzy systems generalise, to establish their modelling capabilities and allows training algorithms to be developed for which convergence can be proved. Throughout this work it has been postulated that fuzzy systems are simply deterministic, non-linear mappings and despite the fact that they can be used to implement vague, heuristic rules, they are amenable to the same rigorous analysis as other techniques used for learning modelling and control. Finally, the link that has been established between neural and fuzzy systems is important to both fields. Neural networks have been criticised because their knowledge is stored in an *opaque* fashion, whereas this work shows how certain associative memory networks can generate fuzzy rules which explain their actions. Similarly the learning theory developed for neural networks can be applied to these fuzzy systems and more advanced network construction algorithms can be used to determine the structure of rule bases and allow these algorithms to be applied in higher dimensional input spaces [15, 3].

Acknowledgements

The authors are grateful for the financial support provided by Lucas Aerospace and the EPSRC during the preparation of this paper.

References

[1] J.S. Albus, A new approach to manipulator control: the cerebellar model articulation controller (CMAC), *Trans. ASME. J. Dyn. Sys. Meas. Control* **63** (1975) 220–227.

[2] H.R. Berenji, Fuzzy Logic Controllers, in: R.R. Yager and L.A. Zadeh, Eds., *An Introduction to Fuzzy Logic Applications and Intelligent Systems* (Kluwer Academic, Dordrecht, 1992): Chap. 4.

[3] K.M. Bossley, D.J. Mills, M. Brown and C.J. Harris, Neurofuzzy high dimensional approximation, in: J.G. Taylor, *Neural Networks* (Alfred Waller Limited, Henley on Thames, Ed., 1995) 297–332.

[4] D.S. Broomhead and D. Lowe, Multivariable functional interpolation and adaptive networks, *Complex Systems* **2** (1988) 321–355.

[5] M. Brown and C.J. Harris, *Neurofuzzy Adaptive Modelling and Control* (Prentice Hall, Hemel Hempstead, 1994).

[6] S. Chen, S.A. Billings and P.M. Grant, Recursive hybrid algorithm for non-linear system identification using radial basis function networks, *Int. J. Control* **55** (5) (1992) 1051–1070.

[7] M.G. Cox, Algorithms for spline curves and surfaces, *NPL Report DITC*, 166/90, 1990.

[8] J. Efstathiou, Expert systems, fuzzy logic and rule-based control explained at last, *Trans. Inst. Meas. Control* **26** (1988) 151–164.

[9] S.D Farrall and R.P. Jones, Energy management in an automative electric/heat engine hybrid powertrain using fuzzy decision making, in: *IEEE 8th Internat. Symp. on Intelligent Control*, Chicago, 1993.

[10] C.J. Harris, C.G. Moore and M. Brown, *Intelligent Control: Some Aspects of Fuzzy Logic and Neural Networks* (World Scientific, Singapore, 1993).

[11] S. Haykin, *Adaptive Filter Theory*, 2nd Edn. (Prentice-Hall Englewood Cliffs, NJ, 1991).

[12] R. Jager, Adaptive fuzzy control, in: L. Boullart and A. Krijgsman, Eds., *Application of Artificial Intelligence in Process Control* (Pergamon Press, Oxford, 1992) 368–387.

[13] T.A. Johansen, Fuzzy model based control: stability, robustness and performance issues, *IEEE Trans. Fuzzy Systems* **2** (3) (1994) 221–234.

[14] T. Kavli, *Learning Principles in Dynamic Control*, Dr. Scient Thesis, Institute for Informatics, University of Oslo, Norway (1992).

[15] T. Kavli, ASMOD: an algorithm for adaptive spline modelling of observation data, *Int. J. Control* **58** (4) (1993) 947–968.

[16] I. Kouatli and B. Jones, An improved design procedure for fuzzy control systems, *Int. J. Mach. Tools Manufact.* **31** (1991) 107–122.

[17] S.H. Lane, D.A. Handelman and J.J. Gelfand, Theory and development of higher order CMAC neural networks, *IEEE Cont. Sys. Mag.* (1992) 23–30.

[18] C.C. Lee, Fuzzy logic in control systems: fuzzy logic controller – Parts 1 and 2, *IEEE Trans. System Man and Cybernet.* **20** (2) (1990) 404–435.

[19] D.A. Linkens and M.F. Abbod, Supervisory intelligent control using a fuzzy logic hierarchy, *Trans. Inst. Meas. Control* **15** (3) (1993) 112–132.

[20] W.T. Miller, F.H. Glanz and L.G. Kraft, CMAC: An associative neural network alternative to backpropagation, *Proc. IEEE* **78** (10) (1990) 1561–1567.

[21] C.G Moore and C.J. Harris, Indirect adaptive fuzzy control, *Int. J. Control* **56** (2) (1992) 441–468.

[22] C.G. Moore, C.J. Harris and E. Rogers, Utilising fuzzy models in the design of estimators and predictors: An agile target tracking example, in: *2nd IEEE Internat. Conf. Fuzzy Systems*, San Francisco, California, Vol. 2 (1993) 679–684.

[23] R. Murray-Smith, A local model network approach to nonlinear modelling, Ph.D. Thesis, Department of Computer Science, University of Strathclyde (1994).

[24] P.J. Pacini and B. Kosko, Adaptive fuzzy systems for target tracking, *Intelligent Systems Engineering* **1** (1) (1992) 3–21.

[25] P.C. Parks and J. Militzer, Convergence properties of associative memory storage for learning control systems, *Autom. Remote Control*, Part 2 **50** (2) (1989) 254–286.

[26] P.C. Parks and J. Militzer, A comparison of five algorithms for the training of CMAC memories for learning control systems, *Automatica* **28** (5) (1992) 1027–1035.

[27] T.J. Procyk and E.H. Mamdani, A linguistic self-organizing process controller, *Automatica* **15** (1979) 15–30.

[28] J.R. Shewchuk, An introduction to the conjugate gradients method without the agonizing pain, School of Computer Science, Carnegie Mellon University, CMU-CS-94-125 (1994).

[29] J. Shao, Y.C. Lee and R. Jones, Orthogonal projection method for fast on-line learning algorithm of radial basis function neural networks, in: *INNS World Congress on Neural Networks*, Portland Oregon, Vol. 3 (1993) 520–535.

[30] M. Sugeno and M. Nishida, Fuzzy control of a model car, *Fuzzy Sets and Systems* **16** (1985) 103–113.

[31] R. Sutton and D.R. Towill, An introduction to the use of fuzzy sets in the implementation of control algorithms, *J. Inst. Elec. Radio Eng.* **55** (10) (1985) 357–367.

[32] R. Sutton and I.M. Jess, A design study of a self-organizing fuzzy autopilot for ship control, *IMechE, Proc. Instn. Mech. Engrs.* **205** (1991) 35–47.

[33] H. Tolle and E. Ersü, *Neurocontrol: Learning Control Systems Inspired by Neuronal Architectures and Human Problem Solving*, Lecture Notes in Control and Information Sciences, Vol. 172 (Springer, Berlin, 1992).

[34] L.X. Wang and J.M. Mendel, Fuzzy basis functions, universal approximation, and orthogonal least-squares learning, *IEEE Trans. Neural Networks* **3** (5) (1992) 807–814.

[35] L.X. Wang, *Adaptive Fuzzy Systems and Control: Design and Stability Analysis* (Prentice-Hall, Englewood Cliffs, NJ, 1994).

[36] H.W. Werntges, Partitions of unity improve neural function approximators, *IEEE Internat. Conf. on Neural Networks*, San Francisco, California, **2** (1993) 914–918.

[37] B. Widrow and M.A. Lehr, 30 years of adaptive neural networks: perceptron, madaline and backpropagation, *Proc. IEEE* **78** (9) (1990) 1415–1441.

[38] L.A. Zadeh, Outline of a new approach to the analysis of complex systems and decision processes, *IEEE Trans. System Man and Cybernet.* **3** (1) (1973) 28–44.