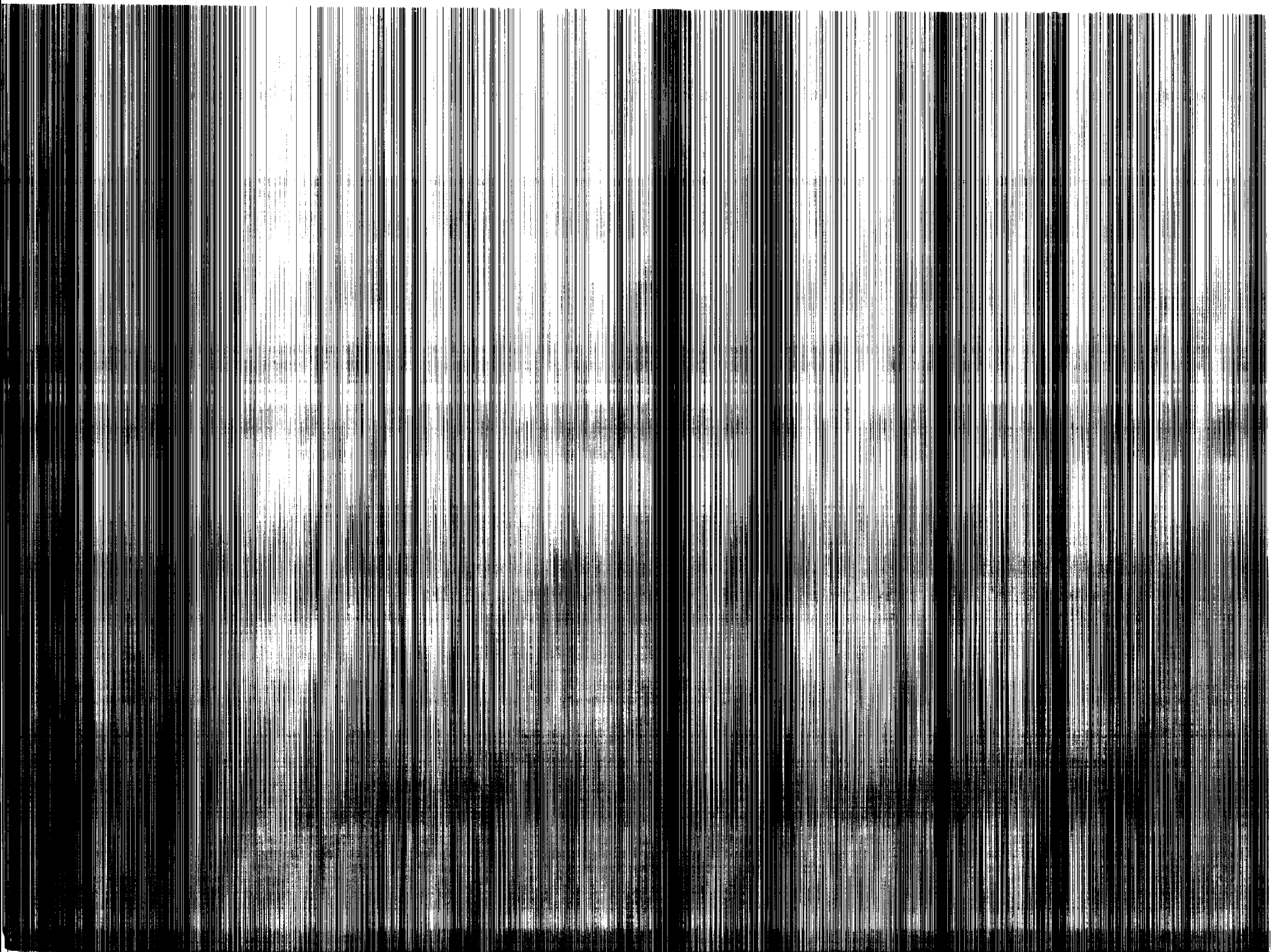
NASA Conference Publication 10061

Proceedings of the Second Joint Technology Workshop on Neural Networks and Fuzzy Logic

Volume I

Proceedings of a workshop held at
Lyndon B. Johnson Space Center
Houston, Texas
April 10 - 13, 1990



NASA Conference Publication 10061

Proceedings of the Second Joint Technology Workshop on Neural Networks and Fuzzy Logic

Volume I

Robert N. Lea and
James Villarreal, Editors
NASA Lyndon B. Johnson Space Center
Houston, Texas

Proceedings of a workshop sponsored by the
National Aeronautics and Space Administration,
Washington, D.C., and cosponsored by
Lyndon B. Johnson Space Center and
the University of Houston, Clear Lake
Houston, Texas
April 10 - 13, 1990

National Aeronautics and
Space Administration

February 1991

Proceedings of the Second Joint Technology Workshop

on Neural Networks and Fuzzy Logic

Program

April 10, 1990 PM

6:00-8:00 Registration

April 11, 1990 AM

7:30-8:00 Registration

8:00-8:30 **Robert T. Savely**, Branch Chief, Software Technology Branch, NASA/Lyndon B. Johnson Space Center, Houston, TX, **Welcoming Remarks**

8:30-9:00 **Jon Erickson**, Division Chief, Automation and Robotics Division, NASA/Lyndon B. Johnson Space Center, Houston, TX, **Overview of Space Station**

9:00-9:30 **Barney Roberts**, Planet Surface System Manager, NASA/Lyndon B. Johnson Space Center, Houston, TX, **Overview of Mars/Lunar Initiative**

9:30-9:45 Break

9:45-10:45 **Bernard Widrow**, Stanford University, **Neural Control Systems**

10:45-11:45 **Bart Kosko**, University of Southern California, Los Angeles, CA, **Fuzzy Sets and Associative Memories**

11:45-1:00 Lunch

April 11, 1990 PM

1:00-1:45 **Hal White**, University of California at San Diego, **Neural Network Representations and Learning of Mappings and their Derivatives**

1:45-2:15 **Masaki Togai**, Togai Infralogic, Inc., Irvine, CA, **Impact of Application of Fuzzy Theory to Industry**

2:15-2:30 Break

2:30-3:00 **Takeshi Yamakawa**, Kyushu Institute of Technology, Iizuka, Fukuoka, Japan, **Time-Sweeping Mode Fuzzy Computer Hardware System --- Forward and Backward Fuzzy Inference Engine ---**

3:00-3:30 **P. Z. Wang**, Institute of Systems Science, National University of Singapore, **The Simplification of Fuzzy Control Algorithm and Hardware Implementation**

3:30-3:45 Break

3:45-5:15 **Lotfi Zadeh**; University of California at Berkeley, moderator of panel discussion on

Fuzzy Sets, Neural Networks, and Intelligent Control

Panel members include:
Bernard Widrow, Stanford University
Bart Kosko, University of Southern California
Masaki Togai, Togai Infralogic Corporation
Takeshi Yamakawa, Kyushu Institute of Technology
P. Z. Wang. National University of Singapore
Hal White, University of California, San Diego
Elie Sanchez, Neural and Fuzzy Systems Institute, Marseilles, France
Paul Werbos, National Science Foundation.

6:00-7:00 Wine and Cheese Reception

7:00-9:00 Banquet and Keynote Speaker

Professor Lotfi Zadeh: The Role of Logic in Human and Machine Intelligence.

April 12, 1990 AM

8:00-8:30 **James Anderson**, Brown University, Providence, RI, **Experiments with Representations in Simple Neural Networks**

8:30-9:00 **James Bezdek**, University of West Florida, **Generalized Self Organizing Clustering Schemes**

9:00-9:15 Break

9:15-9:45 **Hiroyuki Watanabe**, University of North Carolina, Chapel Hill, NC, **A Single Board Fuzzy Inference System**

9:45-10:15 **Isao Hayashi**, Central Research Laboratories, Matsushita Electrical Industrial Co., **The Learning Function of NN-Driven Fuzzy Reasoning under Changes of Reasoning Environment**

10:15-10:30 Break

10:30-11:00 **Kaoru Hirota**, Hosei University, Tokyo, Japan, **A Solution of Inverse Problem of Fuzzy Relational Equation by using Perceptron Model**

11:00-11:45 **Masaki Togai**, Togai Infralogic, Inc., Irvine, CA, **Overview of LIFE (Laboratory for International Fuzzy Engineering) Research**

11:45-1:00 Lunch

April 12, 1990 PM

1:00-1:30 **James Keller**, University of Missouri, Columbia, MO, **Experiments on Neural Network Architectures for Fuzzy Logic**

1:30-2:00 **John Yen**, Texas A&M University, College Station, TX, **Using Fuzzy Logic to Integrate Neural Networks and Knowledge-based Systems**

2:00-2:30 **Hamid Berenji**, Ames Research Center, Palo Alto, CA, **An Architecture for Designing Fuzzy Controllers Using Neural Networks**

2:30-2:45 Break

2:45-3:15 **Rod Taber**, Center of Applied Optics, University of Alabama in Huntsville, Huntsville, Alabama, **Spatiotemporal Pattern Recognition with the Neuron Ring**

3:15-3:45 **Robert Shelton and James Villarreal**, NASA/Lyndon B. Johnson Space Center, Houston, TX, **Spatiotemporal Neural Networks**

3:45-4:15 **Yashvant Jani and Robert N. Lea**, NASA/Lyndon B. Johnson Space Center, Houston, TX, **Fuzzy Logic in Autonomous Spacecraft Operations**

4:15-4:45 **Kumpati (Bob) S. Narendra**, Yale University, New Haven, CT, **Identification and Control of Dynamical Systems using Neural Networks**

4:45-5:15 **Jacob Barhen**, Center for Space Microelectronics Technology, Jet Propulsion Laboratory, Pasadena, CA, **Non-Lipschitzian Dynamics**

April 13, 1990 AM

8:00-8:30 **Dan Greenwood**, Netrologic, San Diego, CA, **Diagnosis and Failure Prediction of the Space Shuttle Main Engine**

8:30-9:00 **Paul Werbos**, National Science Foundation, **Neural Nets for Control and the Link to Fuzzy Logic**

9:00-9:15 Break

9:15-9:45 **C.C. Lee**, University of California at Berkeley, Berkeley, CA, **An Intelligent Control System for Dynamic Processes**

9:45-10:15 **Ronald Yager**, Machine Intelligence Institute, Iona College, New Rochelle, NY, **A Neural Network Based Fuzzy Logic Controller**

10:15-10:45 **Sankar K. Pal**, NASA/Lyndon B. Johnson Space Center, Houston, TX and Indian Statistical Institute, Calcutta, India, **Fuzzy Geometry, Entropy and Image Information**

10:45-11:00 Break

11:00-11:30 **Enrique Ruspini**, Stanford Research Institute, Menlo Park, CA, **The Semantics of Fuzzy Logic**

11:30-12:00 **Robert Dawes**, Martingale Research Corporation, **Identification, Estimation and Control of Dynamical Systems with the Parametric Avalanche Neural Network**

CONTENTS

VOLUME II

An Architecture for Designing Fuzzy Logic Controllers Using Neural Networks 1

An Overview of the Neuron Ring Model 31

A Space - Time Neural Network 63

Fuzzy Logic in Autonomous Orbital Operations 81

Identification and Control of Dynamical Systems Using Neural Networks 111
(Paper not provided by publication date)

Non-Lipschitzian Dynamics 113
(Paper not provided by publication date)

Space Shuttle Main Engine Fault Detection Using Neural Networks 115

Neurocontrol and Fuzzy Logic: Connections and Designs 153

An Intelligent Control Based on Fuzzy Logic and Neural Networks 197

An Neural Network Based Fuzzy Logic Controller 209
(Paper not provided by publication date)

Fuzzy Geometry, Entropy and Image Information 211

The Semantics of Fuzzy Logic 233

Identification, Estimation and Control of Dynamical Systems with the Parametric Avalanche Neural Network 271
(Paper not provided by publication date)

CONTENTS

VOLUME I

Neural Control Systems 1
(Paper not provided by publication date)

Fuzzy Associative Memories 3

Neural Network Representation and Learning of Mappings and their Derivatives 59

Impact of Application of Fuzzy Theory to Industry 91
(Paper not provided by publication date)

Time-sweeping Mode Fuzzy Computer Hardware System -- Forward and Backward Fuzzy Inference Engine 93
(Paper not provided by publication date)

The Simplification of Fuzzy Control Algorithm and Hardware Implementation 95

PANEL: **Fuzzy Sets, Neural Networks, and Intelligent Control**
PANEL MODERATOR: **Lotfi Zadeh, University of California at Berkeley**
PANEL MEMBERS: **Bernard Widrow, Stanford University**
Bart Kosko, University of Southern California
Masaki Togai, Togai Infralogic Corporation
Takeshi Yamakawa, Kyushu Institute of Technology
P. Z. Wang, National University of Singapore
Hal White, University of California, San Diego
Elie Sanchez, Neural and Fuzzy Systems Institute, Marseilles, France
Paul Werbos, National Science Foundation

No papers were presented at this discussion.

Radar Signal Categorization Using a Neural Network 107

Self-organization and Clustering Algorithms 143

Single Board System for Fuzzy Inference 159

Learning Control of Inverted Pendulum System by Neural Network Driven Fuzzy Reasoning 169

Solution of Inverse Problem of Fuzzy Relational Equation by Using Perceptron Model 183

Overview of LIFE (Laboratory for International Fuzzy Engineering) Research 199
(Paper not provided by publication date)

Experiments on Neural Network Architectures for Fuzzy Logic 201

Using Fuzzy Logic to Integrate Neural Networks and Knowledge-based Systems 217

Neural Control Systems

(Paper not provided by publication date.)

In: Kosko, B., NEURAL NETWORKS AND FUZZY SYSTEMS, Prentice-Hall, 1990

CHAPTER 17

FUZZY ASSOCIATIVE MEMORIES

Fuzzy Systems as Between-Cube Mappings

In Chapter 16, we introduced continuous or fuzzy sets as points in the unit hypercube $I^n = [0,1]^n$. Within the cube we were interested in the distance between points. This led to measures of the size and fuzziness of a fuzzy set and, more fundamentally, to a measure of how much one fuzzy set is a subset of another fuzzy set. This *within-cube* theory directly extends to the continuous case where the space X is a subset of R^n or, in general, where X is a subset of products of real or complex spaces.

The next step is to consider mappings *between* fuzzy cubes. This level of abstraction provides a surprising and fruitful alternative to the propositional and predicate-calculus reasoning techniques used in artificial-intelligence (AI) expert systems. It allows us to reason with sets instead of propositions.

The fuzzy set framework is numerical and multidimensional. The AI framework is symbolic and one-dimensional, with usually only bivalent expert "rules" or propositions allowed. Both frameworks can encode structured knowledge in linguistic form. But the fuzzy approach translates the structured knowledge into a flexible *numerical* framework and processes it in a manner that resembles neural network processing. The numerical framework also allows fuzzy systems to be adaptively inferred and modified, perhaps with neural or statistical techniques, directly from problem domain sample data.

Between-cube theory is fuzzy systems theory. A fuzzy set is a point in a cube. A fuzzy system is a mapping between cubes. A fuzzy system S maps fuzzy sets to fuzzy sets. Thus a **fuzzy system** S is a transformation $S : I^n \rightarrow I^p$. The n-dimensional unit hypercube I^n houses all the fuzzy subsets of the domain space, or input *universe of discourse*, $X = \{x_1, \ldots, x_n\}$. I^p houses all the fuzzy subsets of the range space, or output universe of discourse, $Y = \{y_1, \ldots, y_p\}$. X and Y can also be subsets of R^n and R^p. Then the fuzzy power sets $F(2^X)$ and $F(2^Y)$ replace I^n and I^p.

In general a fuzzy system S maps families of fuzzy sets to families of fuzzy sets, thus $S : I^{n_1} \times \ldots \times I^{n_r} \rightarrow I^{p_1} \times \ldots \times I^{p_s}$. Here too we can extend the definition of a fuzzy system to allow arbitrary products of arbitrary mathematical spaces to serve as the domain or range spaces of the fuzzy sets.

(A technical comment is in order for sake of historical clarification. A tenet, perhaps the defining tenet, of the classical theory [Dubois, 1980] of fuzzy sets as functions concerns the fuzzy extension of any mathematical function. This tenet holds that any function $f : X \rightarrow Y$ that maps points in X to points in Y can be extended to map the fuzzy subsets of X to the fuzzy subsets of Y. The so-called *extension principle* is used to define the set-function $f : F(2^X) \rightarrow F(2^Y)$, where $F(2^X)$ is the fuzzy power set of X, the set of all fuzzy subsets of X. The formal definition of the extension principle is complicated. The key idea is a supremum of pairwise minima. Unfortunately, the extension principle achieves generality at the price of triviality. One can show [Kosko, 1986a-87] that in general the extension principle extends functions to fuzzy sets by stripping the fuzzy sets of their fuzziness, mapping the fuzzy sets into bit vectors of nearly all 1s. This shortcoming, combined with the tendency of the extension-principle framework to push fuzzy theory into largely inaccessible regions of abstract mathematics, led in part to the development of the alternative sets-as-points geometric framework of fuzzy theory.)

We shall focus on fuzzy systems $S : I^n \rightarrow I^p$ that map *balls* of fuzzy sets in I^n to balls of fuzzy sets in I^p. These continuous fuzzy systems behave as associative memories. They map close inputs to close outputs. We shall refer to them as **fuzzy associative memories**, or FAMs.

The simplest FAM encodes the **FAM rule** or association (A_i, B_i), which associates

the p-dimensional fuzzy set B_i with the n-dimensional fuzzy set A_i. These minimal FAMs essentially map one ball in I^n to one ball in I^p. They are comparable to simple neural networks. But the minimal FAMs need not be adaptively trained. As discussed below, structured knowledge of the form "If traffic is heavy in this direction, then keep the stop light green longer" can be directly encoded in a Hebbian-style FAM matrix. In practice we can eliminate even this matrix. In its place the user encodes the fuzzy-set association (HEAVY, LONGER) as a single linguistic entry in a FAM bank matrix.

In general a **FAM system** $F : I^n \rightarrow I^p$ encodes and processes in parallel a **FAM bank** of m FAM rules $(A_1, B_1), \ldots, (A_m, B_m)$. Each input A to the FAM system activates each stored FAM rule to different degree. The minimal FAM that stores (A_i, B_i) maps input A to B'_i, a partially activated version of B_i. The more A resembles A_i, the more B'_i resembles B_i. The corresponding output fuzzy set B combines these partially activated fuzzy sets $B'_1, \ldots, B'_m$. In the simplest case B is a weighted average of the partially activated sets:

$$B = w_1 B'_1 + \ldots + w_m B'_m ,$$

where w_i reflects the credibility, frequency, or strength of the fuzzy association (A_i, B_i). In practice we usually "defuzzify" the output waveform B to a single numerical value y_j in Y by computing the fuzzy centroid of B with respect to the output universe of discourse Y.

More general still, a FAM system encodes a bank of compound FAM rules that associate multiple output or consequent fuzzy sets $B_i^1, \ldots, B_i^s$ with multiple input or antecedent fuzzy sets $A_i^1, \ldots, A_i^r$. We can treat compound FAM rules as compound linguistic conditionals. Structured knowledge can then be naturally, and in many cases easily, obtained. We combine antecedent and consequent sets with logical conjunction, disjunction, or negation. For instance, we would interpret the compound association $(A^1, A^2;\ B)$ linguistically as the compound conditional "IF X^1 is A^1 AND X^2 is A^2 , THEN Y is B" if the comma in the fuzzy association $(A^1, A^2;\ B)$ stood for conjunction instead of, say, disjunction.

We specify in advance the numerical universes of discourse X^1, X^2, and Y. For each universe of discourse X, we specify an appropriate *library* of fuzzy set values, $A_1^r, \ldots, A_k^r$. Contiguous fuzzy sets in a library overlap. In principle a neural network can estimate these

libraries of fuzzy sets. In practice this is usually unnecessary. The library sets represent a weighted, though overlapping, quantization of the input space X. A different library of fuzzy sets similarly quantizes the output space Y. Once the library of fuzzy sets is defined, we construct the FAM by choosing appropriate combinations of input and output fuzzy sets. We can use adaptive techniques to make, assist, or modify these choices.

An **adaptive FAM** (AFAM) is a *time-varying* FAM system. System parameters gradually change as the FAM system samples and processes data. Below we discuss how neural network algorithms can adaptively infer FAM rules from training data. In principle learning can modify other FAM system components, such as the libraries of fuzzy sets or the FAM-rule weights w_i.

Below we propose and illustrate an unsupervised adaptive clustering scheme, based on competitive learning, for "blindly" generating and refining the bank of FAM rules. In some cases we can use supervised learning techniques, though we need additional information to accurately generate error estimates.

FUZZY AND NEURAL FUNCTION ESTIMATORS

Neural and fuzzy systems estimate sampled functions and behave as associative memories. They share a key advantage over traditional statistical-estimation and adaptive-control approaches to function estimation. They are *model-free* estimators. Neural and fuzzy systems estimate a function without requiring a mathematical description of how the output functionally depends on the input. They "learn from example." More precisely, they learn from samples.

Both approaches are numerical, can be partially described with theorems, and admit an algorithmic characterization that favors silicon and optical implementation. These properties distinguish neural and fuzzy approaches from the symbolic processing approaches of artificial intelligence.

Neural and fuzzy systems differ in how they estimate sampled functions. They differ in the kind of samples used, how they represent and store those samples, and how they

associatively "inference" or map inputs to outputs.

These differences appear during system construction. The neural approach requires the specification of a nonlinear dynamical system, usually feedforward, the acquisition of a sufficiently representative set of numerical training samples, and the encoding of those training samples in the dynamical system by repeated learning cycles. The fuzzy system requires only that a linguistic "rule matrix" be partially filled in. This task is markedly simpler than designing and training a neural network. Once we construct the systems, we can present the same numerical inputs to either system. The outputs will be in the same numerical space of alternatives. So both systems correspond to a surface or manifold in the input-output product space $X \times Y$. We present examples of these surfaces in Chapters 18 and 19.

Which system, neural or fuzzy, is more appropriate for a particular problem depends on the nature of the problem and the availability of numerical and structured data. To date fuzzy techniques have been most successfully applied to control problems. These problems often permit comparison with standard control-theoretic and expert-system approaches. Neural networks so far seem best applied to ill-defined two-class pattern recognition problems (defective or nondefective, bomb or not, etc.). The application of both approaches to new problem areas is just beginning, amid varying amounts of enthusiasm and scepticism.

Fuzzy systems estimate functions with *fuzzy set* samples (A_i, B_i). Neural systems use *numerical point* samples (x_i, y_i). Both kinds of samples are from the input-output product space $X \times Y$. Figure 17.1 illustrates the geometry of fuzzy-set and numerical-point samples taken from the function $f\colon X \rightarrow Y$.

The fuzzy-set association (A_i, B_i) is sometimes called a "rule." This is misleading since reasoning with sets is not the same as reasoning with propositions. Reasoning with sets is harder. Sets are multidimensional, and associations are housed in matrices, not conditionals. We must take care how we define each term and operation. We shall refer to the antecedent term A_i in the fuzzy association (A_i, B_i) as the **input associant** and the

consequent term B_i as the **output associant.**

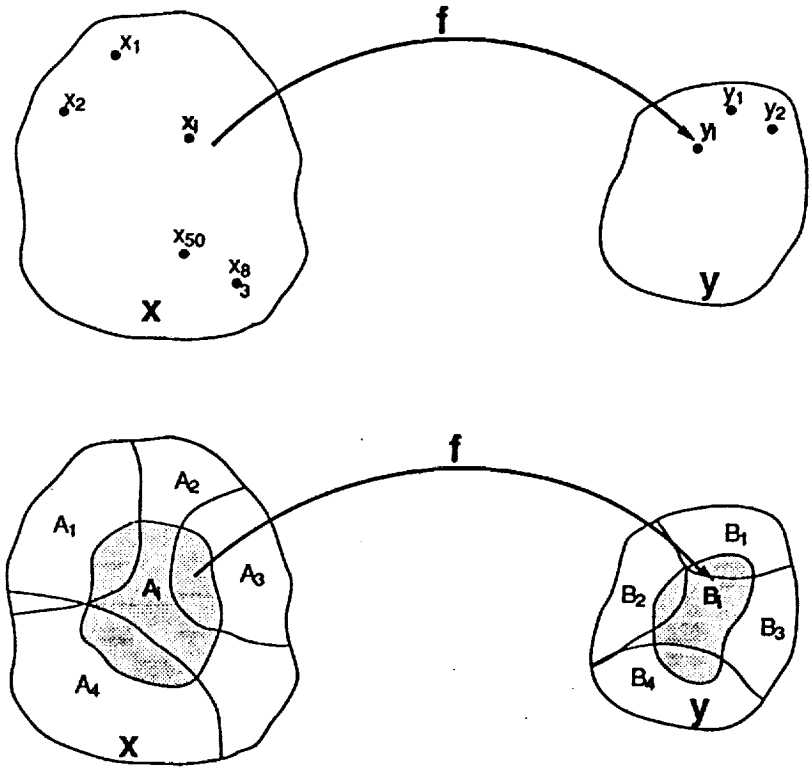


FIGURE 17.1 Function f maps domain X to range Y. In the first illustration we use several numerical point samples $(x_i,\ y_i)$ to estimate f: $X \longrightarrow Y$. In the second case we use only a few fuzzy subsets A_i of X and B_i of Y. The fuzzy association $(A_i,\ B_i)$ represents system structure, as an adaptive clustering algorithm might infer or as an expert might articulate. In practice there are

usually fewer different output associants or "rule" consequents B_i than input associants or antecedents A_i.

The fuzzy-set sample (A_i, B_i) encodes *structure*. It represents a mapping itself, a minimal *fuzzy association* of part of the output space with part of the input space. In practice this resembles a meta-rule—IF A_i, THEN B_i—the type of structured linguistic rule an expert might articulate to build an expert-system "knowledge base". The association might also be the result of an adaptive clustering algorithm.

Consider a fuzzy association that might be used in the intelligent control of a traffic light: "If the traffic is heavy in this direction, then keep the light green longer." The fuzzy association is (HEAVY, LONGER). Another fuzzy association might be (LIGHT, SHORTER). The fuzzy system encodes each linguistic association or "rule" in a numerical *fuzzy associative memory* (FAM) mapping. The FAM then numerically processes numerical input data. A measured description of traffic density (e.g., 150 cars per unit road surface area) then corresponds to a unique numerical output (e.g., 3 seconds), the "recalled" output.

The degree to which a particular measurement of traffic density is heavy depends on how we define the fuzzy set of heavy traffic. The definition may be obtained from statistical or neural clustering of historical data or from pooling the responses of experts. In practice the fuzzy engineer and the problem domain expert agree on one of many possible libraries of fuzzy set definitions for the variables in question.

The degree to which the traffic light is kept green longer depends on the degree to which the measurement is heavy. In the simplest case the two degrees are the same. In general they differ. In actual fuzzy systems the output control variables—in this case the single variable green light duration—depend on many FAM rule antecedents or associants that are activated to different degrees by incoming data.

Neural vs. Fuzzy Representation of Structured Knowledge

The functional distinction between how fuzzy and neural systems differ begins with how they represent structured knowledge. How would a neural network encode the same associative information? How would a neural network encode the structured knowledge "If the traffic is heavy in this direction, then keep the light green longer"?

The simplest method is to encode two associated numerical vectors. One vector represents the input associant HEAVY. The other vector represents the output associant LONGER. But this is too simple. For the neural network's fault tolerance now works to its disadvantage. The network tends to reconstruct partial inputs to complete sample inputs. It erases the desired partial degrees of activation. If an input is close to A_i, the output will tend to be B_i. If the output is distant from A_i, the output will tend to be some other sampled output vector or a spurious output altogether.

A better neural approach is to encode a mapping from the heavy-traffic subspace to the longer-time subspace. Then the neural network needs a representative sample set to capture this structure. Statistical networks, such as adaptive vector quantizers, may need thousands of statistically representative samples. Feedforward multi-layer neural networks trained with the backpropagation algorithm may need hundreds of representative numerical input-output pairs and may need to recycle these samples tens of thousands of times in the learning process.

The neural approach suffers a deeper problem than just the computational burden of training. *What* does it encode? How do we know the network encodes the original structure? What does it recall? There is no natural inferential audit trail. System nonlinearities wash it away. Unlike an expert system, we do not know which inferential paths the network uses to reach a given output or even which inferential paths exist. There is only a system of synchronous or asynchronous nonlinear functions. Unlike, say, the adaptive Kalman filter, we cannot appeal to a postulated mathematical model of how the output state depends on the input state. Model-free estimation is, after all, the central computational advantage of neural networks. The cost is system inscrutability.

We are left with an unstructured computational black box. We do not know what the neural network encoded during training or what it will encode or forget in further training. (For competitive adaptive vector quantizers we do know that sample-space centroids are asymptotically estimated.) We can characterize the neural network's behavior only by exhaustively passing all inputs through the black box and recording the recalled outputs. The characterization may be in terms of a summary scalar like mean-squared error.

This black-box characterization of the network's behavior involves a computational *dilemma*. On the one hand, for most problems the number of input-output cases we need to check is computationally prohibitive. On the other, when the number of input-output cases is tractable, we may as well store these pairs and appeal to them directly, and without error, as a look-up table. In the first case the neural network is unreliable. In the second case it is unnecessary.

A further problem is sample generation. Where did the original numerical point samples come from? Was an expert asked to give numbers? How reliable are such numerical vectors, especially when the expert feels most comfortable giving the original linguistic data? This procedure seems at most as reliable as the expert-system method of asking an expert to give condition-action rules with numerical uncertainty weights.

Statistical neural estimators require a "statistically representative" sample set. We may need to randomly "create" these samples from an initial small sample set by bootstrap techniques or by random-number generation of points clustered near the original samples. Both sample-augmentation procedures assume that the initial sample set sufficiently represents the underlying probability distribution. The problem of where the original sample set comes from remains. The fuzziness of the notion "statistically representative" compounds the problem. In general we do not know in advance how well a given sample set reflects an unknown underlying distribution of points. Indeed when the network is adapting on-line, we know only past samples. The remainder of the sample set is in the unsampled future.

In contrast, fuzzy systems directly encode the linguistic sample (HEAVY, LONGER) in a dedicated numerical matrix. The default encoding technique is the fuzzy Hebb procedure discussed below. For practical problems, as mentioned above, the numerical matrix need not be stored. Indeed it need not even be formed. Certain numerical inputs permit this

simplification, as we shall see below. In general we describe inputs by an uncertainty distribution, probabilistic or fuzzy. Then we must use the entire matrix.

For instance, if a *heavy traffic* input is simply the number 150, we can omit the FAM matrix. But if the input is a Gaussian curve with mean 150, then in principle we must process the vector input with a FAM matrix. (In practice we might use only the mean.) This difference is explained below. The dimensions of the linguistic FAM bank matrix are usually small. The dimensions reflect the quantization levels of the input and output spaces.

The fuzzy approach combines the purely numerical approaches of neural networks and mathematical modeling with the symbolic, structure-rich approaches of artificial intelligence. We acquire knowledge symbolically—or numerically if we use adaptive techniques —but represent it numerically. We also process data numerically. Adaptive FAM rules correspond to common-sense, often non-articulated, behavioral rules that improve with experience.

We can acquire structured expertise in the fuzzy terminology of the knowledge source, the "expert." This requires little or no force-fitting. Such is the expressive power of fuzziness. Yet in the numerical domain we can prove theorems and design hardware.

This approach does not abandon neural network techniques. Instead, it limits them to *unstructured* parameter and state estimation, pattern recognition, and cluster formation. The system *architecture* remains fuzzy, though perhaps adaptively so. In the same spirit, no one believes that the brain is a single unstructured neural network.

FAMS as Mappings

Fuzzy associative memories (FAMs) are transformations. *FAMs map fuzzy sets to fuzzy sets.* They map unit cubes to unit cubes. This is evident in Figure 17.1. In the simplest case the FAM consists of a single association, such as (HEAVY, LONGER). In general the FAM consists of a bank of different FAM associations. Each association is represented by a different numerical FAM matrix, or a different entry in a FAM-bank

matrix. These matrices are not combined as with neural network associative memory (outer-product) matrices. (An exception is the *fuzzy cognitive map* [Kosko, 1988; Taber, 1987, 1990].) The matrices are stored separately but accessed in parallel.

We begin with single-association FAMs. For concreteness let the fuzzy-set pair (A, B) encode the traffic-control association (HEAVY, LIGHT). We quantize the domain of traffic density to the n numerical variables $x_1, x_2, \ldots, x_n$. We quantize the range of green-light duration to the p variables $y_1, y_2, \ldots, y_p$. The elements x_i and y_j belong respectively to the ground sets $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_p\}$. x_1 might represent zero traffic density. y_p might represent 10 seconds.

The fuzzy sets A and B are fuzzy subsets of X and Y. So A is point in the n-dimensional unit hypercube $I^n = [0, 1]^n$, and B is a point in the p-dimensional fuzzy cube I^p. Equivalently, we can think of A and B as membership functions m_A and m_B mapping the elements x_i of X and y_j of Y to degrees of membership in [0, 1]. The membership values, or *fit* (fuzzy unit) values, indicate how much x_i belongs to or fits in subset A, and how much y_j belongs to B. We describe this with the abstract functions $m_A : X \longrightarrow [0, 1]$ and $m_B : Y \longrightarrow [0, 1]$. We shall freely view sets both as functions and as points.

The geometric *sets-as-points* interpretation of fuzzy sets A and B as points in unit cubes allows a natural vector representation. We represent A and B by the numerical *fit vectors* $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_p)$, where $a_i = m_A(x_i)$ and $b_j = m_B(y_j)$. We can interpret the identifications A = HEAVY and B = LONGER to suit the problem at hand. Intuitively the a_i values should increase as the index i increases, perhaps approximating a sigmoid membership function. Figure 17.2 illustrates three possible fuzzy subsets of the universe of discourse X.

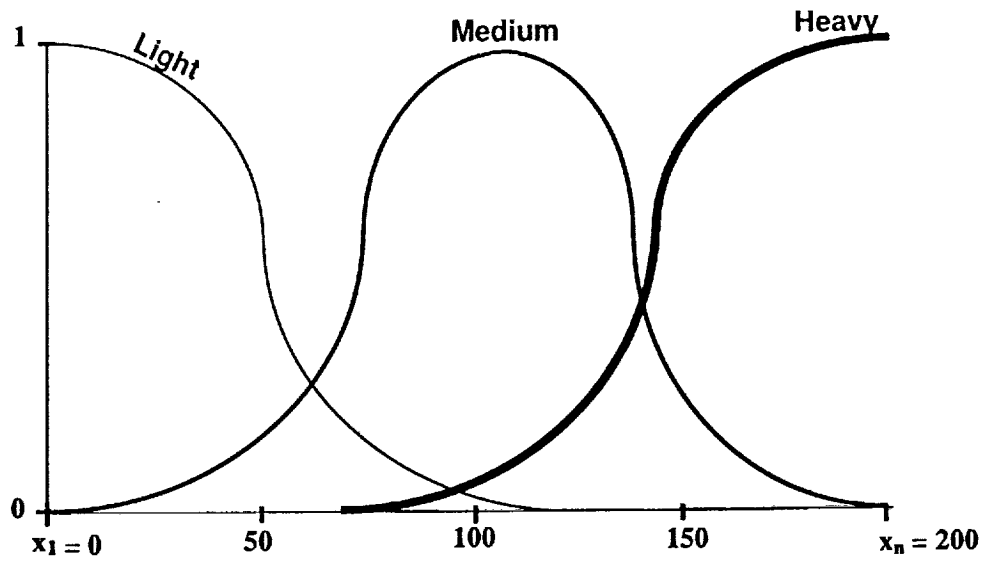


FIGURE 17.2 Three possible fuzzy subsets of traffic density space X. Each fuzzy sample corresponds to such a subset. We draw the fuzzy sets as continuous membership functions. In practice membership values are quantized. So the sets are points in the unit hypercube I^n. Each fuzzy sample corresponds to such a subset.

Fuzzy Vector-Matrix Multiplication: Max-Min Composition

Fuzzy vector-matrix multiplication is similar to classical vector-matrix multiplication. We replace pairwise multiplications with pairwise minima. We replace column (row) sums with column (row) maxima. We denote this **fuzzy vector-matrix composition** relation, or the **max-min composition** relation [Klir, 1988], by the composition operator "o". For row fit vectors A and B and fuzzy n-by-p matrix M (a point in $I^{n \times p}$):

$$A \circ M = B \quad , \tag{1}$$

where we compute the "recalled" component b_j by taking the fuzzy inner product of fit vector A with the jth column of M:

$$b_j = \max_{1 \le i \le n} \min(a_i, m_{ij}) \quad . \tag{2}$$

Suppose we compose the fit vector $A = (.3\ .4\ .8\ 1)$ with the fuzzy matrix M given by

$$M = \begin{pmatrix} .2 & .8 & .7 \\ .7 & .6 & .6 \\ .8 & .1 & .5 \\ 0 & .2 & .3 \end{pmatrix} .$$

Then we compute the "recalled" fit vector $B = A \circ M$ component-wise as

$$\begin{aligned} b_1 &= \max\{\min(.3,\ .2),\ \min(.4,\ .7),\ \min(.8,\ .8),\ \min(1,\ 0)\} \\ &= \max(.2,\ .4,\ .8,\ 0) \\ &= .8\ , \\ b_2 &= \max(.3,\ .4,\ .1,\ .2) \\ &= .4\ , \\ b_3 &= \max(.3,\ .4,\ .5,\ .3) \\ &= .5\ . \end{aligned}$$

So $B = (.8\ .4\ .5)$. If we somehow encoded $(A,\ B)$ in the FAM matrix M, we would say that the FAM system exhibits *perfect recall* in the forward direction.

The neural interpretation of max-min composition is that each neuron in field F_Y (or field F_B) generates its signal/activation value by fuzzy linear composition. Passing

information back through M^T allows us to interpret the fuzzy system as a bidirectional associative memory (BAM). The Bidirectional FAM Theorems below characterize successful BAM recall for fuzzy correlation or Hebbian learning.

For completeness we also mention the **max-product composition** operator, which replaces minimum with product in (2):

$$b_j = \max_{1 \le i \le n} a_i \, m_{ij} \quad .$$

In the fuzzy literature this composition operator is often confused with the fuzzy correlation encoding scheme discussed below. Max-product composition is a method for "multiplying" fuzzy matrices or vectors. Fuzzy correlation, which also uses pairwise products of fit values, is a method for constructing fuzzy matrices. In practice, and in the following discussion, we use only max-min composition.

FUZZY HEBB FAMs

Most fuzzy systems found in applications are fuzzy Hebb FAMs [Kosko, 1986b]. They are fuzzy systems $S : I^n \rightarrow I^p$ constructed in a simple neural-like manner. As discussed in Chapter 4, in neural network theory we interpret the classical Hebbian hypothesis of correlation synaptic learning [Hebb, 1949] as unsupervised learning with the signal product $S_i \, S_j$:

$$\dot{m}_{ij} = -m_{ij} + S_i(x_i) \, S_j(y_j) \quad . \tag{3}$$

For a given pair of bipolar vectors (X, Y), the neural interpretation gives the *outer-product* correlation matrix

$$M = X^T Y \quad . \tag{4}$$

The **fuzzy Hebb matrix** is similarly defined pointwise by the minimum of the "signals" a_i and b_j, an encoding scheme we shall call **correlation-minimum encoding**:

$$m_{ij} = \min(a_i, b_j) \quad , \tag{5}$$

given in matrix notation as the *fuzzy outer-product*

$$M = A^T \circ B \quad . \tag{6}$$

Mamdani [1977] and Togai [1986] independently arrived at the fuzzy Hebbian prescription (5) as a multi-valued logical-implication operator: $\text{truth}(a_i \rightarrow b_i) = \min(a_i, b_j)$. The min operator, though, is a symmetric truth operator. So it does not properly generalize the classical implication $P \rightarrow Q$, which is false if and only if the antecedent P is true and the consequent Q is false, $t(P) = 1$ and $t(Q) = 0$. In contrast, a like desire to define a "conditional possibility" matrix pointwise with continuous implication values led Zadeh [1983] to choose the Lukasiewicz implication operator: $m_{ij} = \text{truth}(a_i \rightarrow b_j) = \min(1, 1 - a_i + b_j)$. The problem with the Lukasiewicz operator is that it usually unity. For $\min(1, 1 - a_i + b_j) < 1$ iff $a_i > b_j$. Most entries of the resulting matrix M are unity or near unity. This ignores the information in the association (A, B). So $A' \circ M$ tends to equal the largest fit value a'_k for any system input A'.

We construct an *autoassociative* fuzzy Hebb FAM matrix by encoding the redundant pair (A, A) in (6), as the fuzzy auto-correlation matrix:

$$M = A^T \circ A \quad . \tag{7}$$

In the previous example the matrix M was such that the input $A = (.3 \;\; .4 \;\; .8 \;\; 1)$ recalled fit vector $B = (.8 \;\; .4 \;\; .5)$ upon max-min composition: $A \circ M = B$. Will B still be recalled if we replace the original matrix M with the fuzzy Hebb matrix found with (6)? Substituting A and B in (6) gives

$$M = A^T \circ B = \begin{pmatrix} .3 \\ .4 \\ .8 \\ 1 \end{pmatrix} \circ (.8 \;\; .4 \;\; .5) = \begin{pmatrix} .3 & .3 & .3 \\ .4 & .4 & .4 \\ .8 & .4 & .5 \\ .8 & .4 & .5 \end{pmatrix} \quad .$$

This fuzzy Hebb matrix M illustrates two key properties. First, the ith row of M is the pairwise minimum of a_i and the output associant B. Symmetrically, the jth column of M is the pairwise minimum of b_j and the input associant A:

$$M = \left[\begin{array}{c} a_1 \wedge B \\ \hline \vdots \\ \hline a_n \wedge B \end{array}\right] \tag{8}$$

$$= [b_1 \wedge A^T \mid \ldots \mid b_m \wedge A^T] \quad , \tag{9}$$

where the cap operator denotes pairwise minimum: $a_i \wedge b_j = \min(a_i, b_j)$. The term $a_i \wedge B$ indicates component-wise minimum:

$$a_i \wedge B = (a_i \wedge b_1, \ldots, a_i \wedge b_n) \quad . \tag{10}$$

Hence if some $a_k = 1$, then the kth row of M is B. If some $b_l = 1$, the lth column of M is A. More generally, if some a_k is at least as large as every b_j, then the kth row of the fuzzy Hebb matrix M is B.

Second, the third and fourth columns of M are just the fit vector B. Yet no column is A. This allows perfect recall in the forward direction, $A \circ M = B$, but not in the backward direction, $B \circ M^T \neq A$:

$$A \circ M = (.8 \ .4 \ .5) = B \quad ,$$

$$B \circ M^T = (.3 \ .4 \ .8 \ .8) = A' \subset A \quad .$$

A' is a proper subset of A: $A' \neq A$ and $S(A', A) = 1$, where S measures the degree of subsethood of A' in A, as discussed in Chapter 16. In other words, $a'_i \leq a_i$ for each i and $a'_k < a_k$ for at least one k. The Bidirectional FAM Theorems below show that this is a general property: If $B' = A \circ M$ differs from B, then B' is a proper subset of B. Hence fuzzy subsets truly map to fuzzy subsets.

The Bidirectional FAM Theorem for Correlation-Minimum Encoding

Analysis of FAM recall uses the traditional [Klir, 1988] fuzzy set notions of the *height* and the *normality* of fuzzy sets. The **height** $H(A)$ of fuzzy set A is the maximum fit value of A:

$$H(A) = \max_{1 \leq i \leq n} a_i \quad .$$

A fuzzy set is **normal** if $H(A) = 1$, if at least one fit value a_k is maximal: $a_k = 1$. In practice fuzzy sets are usually normal. We can extend a nonnormal fuzzy set to a normal fuzzy set by adding a dummy dimension with corresponding fit value $a_{n+1} = 1$.

Recall accuracy in fuzzy Hebb FAMs constructed with correlation-minimum encoding depends on the heights $H(A)$ and $H(B)$. Normal fuzzy sets exhibit perfect recall. Indeed (A, B) is a bidirectional fixed point—$A \circ M = B$ and $B \circ M^T = A$—if and only if $H(A) = H(B)$, which always holds if A and B are normal. This is the content of the Bidirectional FAM Theorem [Kosko, 1986a] for correlation-minimum encoding. Below we present a similar theorem for correlation-product encoding.

Correlation-Minimum Bidirectional FAM Theorem. If $M = A^T \circ B$, then

(i) $A \circ M = B$ iff $H(A) \geq H(B)$,

(ii) $B \circ M^T = A$ iff $H(B) \geq H(A)$,

(iii) $A' \circ M \subset B$ for any A' .

(iv) $B' \circ M^T \subset A$ for any B' .

Proof. Observe that the height $H(A)$ is the *fuzzy norm* of A:

$$A \circ A^T = \max_i a_i \wedge a_i = \max_i a_i = H(A) \ .$$

Then

$$\begin{aligned} A \circ M &= A \circ (A^T \circ B) \\ &= (A \circ A^T) \circ B \\ &= H(A) \circ B \\ &= H(A) \wedge B \ . \end{aligned}$$

So $H(A) \wedge B = B$ iff $H(A) \geq H(B)$, establishing (i). Now suppose A' is an arbitrary fit vector in I^n. Then

$$\begin{aligned} A' \circ M &= (A' \circ A^T) \circ B \\ &= (A' \circ A^T) \wedge B \ , \end{aligned}$$

which establishes (iii). A similar argument using $M^T = B^T \circ A$ establishes (ii) and (iv). Q.E.D.

The equality $A \circ A^T = H(A)$ implies an immediate corollary of the Bidirectional FAM Theorem. Supersets $A' \supset A$ behave the same as the encoded input associant A: $A' \circ M = B$ if $A \circ M = B$. Fuzzy Hebb FAMs ignore the information in the difference $A' - A$, when $A' \subset A'$.

Correlation-Product Encoding

An alternative fuzzy Hebbian encoding scheme is **correlation-product encoding**. The standard mathematical outer product of the fit vectors A and B forms the FAM matrix M. This is given pointwise as

$$m_{ij} \;=\; a_i \, b_j \quad , \tag{11}$$

and in matrix notation as

$$M \;=\; A^T \, B \quad . \tag{12}$$

So the ith row of M is just the fit-scaled fuzzy set $a_i \; B$, and the jth column of M is $b_j \; A^T$:

$$M \;=\; \left[\begin{array}{c} a_1 \; B \\ \hline \vdots \\ \hline a_n \; B \end{array}\right] \tag{13}$$

$$=\; [b_1 \; A^T \mid \ldots \mid b_m \; A^T] \quad , \tag{14}$$

If $A \;=\; (.3 \;\; .4 \;\; .8 \;\; 1)$ and $B \;=\; (.8 \;\; .4 \;\; .5)$ as above, we encode the FAM rule $(A, \; B)$ with correlation-product in the following matrix M:

$$M \;=\; \begin{pmatrix} .24 & .12 & .15 \\ .32 & .16 & .2 \\ .64 & .32 & .4 \\ .8 & .4 & .5 \end{pmatrix} \quad .$$

Note that if $A' \;=\; (0 \;\; 0 \;\; 0 \;\; 1)$, then $A' \circ M \;=\; B$. The output associant B is recalled to maximal degree. If $A' \;=\; (1 \;\; 0 \;\; 0 \;\; 0)$, then $A' \circ M \;=\; (.24 \;\; .12 \;\; .15)$. The output B is recalled only to degree .3.

Correlation-minimum encoding produces a matrix of clipped B sets. Correlation-product encoding produces a matrix of scaled B sets. In membership function plots, the scaled fuzzy sets $a_i \; B$ all have the same shape as B. The clipped fuzzy sets $a_i \; \wedge \; B$ are largely flat. In this sense correlation-product encoding preserves more information than correlation-minimum encoding, an important point in fuzzy applications when output fuzzy sets are added together as in equation (17) below. In the fuzzy-applications literature this often leads to the selection of correlation-product encoding.

Unfortunately, in the fuzzy-applications literature the correlation-product *encoding* scheme is invariably confused with the max-product composition method of recall or *inference*, as mentioned above. This confusion is so widespread it warrants formal clarification.

In practice, and in the fuzzy control applications developed in Chapters 18 and 19, the input fuzzy set A' is a binary vector with one 1 and all other elements 0—a row of the n-by-n identity matrix. A' represents the occurrence of the crisp measurement datum x_i, such as a traffic density value of 30. When applied to the encoded FAM rule (A, B), the measurement value x_i activates A to degree a_i. This is part of the max-min composition recall process, for $A' \circ M = (A' \circ A^T) \circ B = a_i \wedge B$ or $a_i B$ depending on whether correlation-minimum or correlation-product encoding is used. We activate or "fire" the output associant B of the "rule" to degree a_i.

Since the values a_i are binary, $a_i\ m_{ij} = a_i \wedge m_{ij}$. So the max-min and max-product composition operators coincide. We avoid this confusion by referring to both the recall process and the correlation encoding scheme as **correlation-minimum inference** when correlation-minimum encoding is combined with max-min composition, and as **correlation-product inference** when correlation-product encoding is combined with max-min composition.

We now prove the correlation-product version of the Bidirectional FAM Theorem.

Correlation-Product Bidirectional FAM Theorem. If $M = A^T B$ and A and B are non-null fit vectors, then

(i) $A \circ M = B$ iff $H(A) = 1$,

(ii) $B \circ M^T = A$ iff $H(B) = 1$,

(iii) $A' \circ M \subset B$ for any A' .

(iv) $B' \circ M^T \subset A$ for any B' .

Proof.

$$\begin{aligned} A \circ M &= A \circ (A^T B) \\ &= (A \circ A^T) B \\ &= H(A) B \quad . \end{aligned}$$

Since B is not the empty set, $H(A) B = B$ iff $H(A) = 1$, establishing (i). ($A \circ M = B$ holds trivially if B is the empty set.) For an arbitrary fit vector A' in I^n:

$$\begin{aligned} A' \circ M &= (A' \circ A^T) B \\ &\subset H(A) B \\ &\subset B \quad , \end{aligned}$$

since $A' \circ A \leq H(A)$, establishing (iii). (ii) and (iv) are proved similarly using $M^T = B^T A$. Q.E.D.

Superimposing FAM Rules

Now suppose we have m FAM rules or associations $(A_1, B_1), \ldots, (A_m, B_m)$. The fuzzy Hebb encoding scheme (6) leads to m FAM matrices $M_1, \ldots, M_m$ to encode the associations. The natural neural-network temptation is to add, or in this case maximum, the m matrices pointwise to distributively encode the associations in a single matrix M:

$$M = \max_{1 \leq k \leq m} M_k \quad . \tag{15}$$

This superimposition scheme fails for fuzzy Hebbian encoding. The superimposed result tends to be the matrix $A^T \circ B$, where A and B are the pointwise maximum of the respective m fit vectors A_k and B_k. We can see this from the pointwise inequality

$$\max_{1 \leq k \leq m} \min(a_i^k, b_j^k) \leq \min(\max_{1 \leq k \leq m} a_i^k, \max_{1 \leq k \leq m} b_j^k) \ . \tag{16}$$

Inequality (16) tends to hold with equality as m increases since all maximum terms approach unity. We lose the information in the m associations (A_k, B_k).

The fuzzy approach to the superimposition problem is to *additively superimpose the m recalled vectors* B_k' instead of the fuzzy Hebb matrices M_k. B_k' and M_k are given by

$$\begin{aligned} A \circ M_k &= A \circ (A_k^T \circ B_k) \\ &= B_k' \ , \end{aligned}$$

for any fit-vector input A applied in parallel to the bank of FAM rules (A_k, B_k). This requires separately storing the m associations (A_k, B_k), as if each association in the FAM bank were a separate feedforward neural network.

Separate storage of FAM associations is costly but provides an "audit trail" of the FAM inference procedure. The user can directly determine which FAM rules contributed how much membership activation to a "concluded" output. Separate storage also provides knowledge-base modularity. The user can add or delete FAM-structured knowledge without disturbing stored knowledge. Both of these benefits are advantages over a pure neural-network architecture for encoding the same associations (A_k, B_k). Of course we can use neural networks exogenously to estimate, or even individually house, the associations (A_k, B_k).

Separate storage of FAM rules brings out another distinction between FAM systems and neural networks. A fit-vector input A activates all the FAM rules (A_k, B_k) in parallel but to different degrees. If A only partially "satisfies" the antecedent associant A_k, the consequent associant B_k is only partially activated. If A does not satisfy A_k at all, B_k does not activate at all. B_k' is the null vector.

Neural networks behave differently. They try to reconstruct the entire association (A_k, B_k) when stimulated with A. If A and A_k mismatch severely, a neural network will

tend to emit a non-null output B'_k, perhaps the result of the network dynamical system falling into a "spurious" attractor in the state space. This may be desirable for metrical classification problems. It is undesirable for inferential problems and, arguably, for associative memory problems. When we ask an expert a question outside his field of knowledge, in many cases it is more prudent for him to give no response than to give an educated, though wild, guess.

Recalled Outputs and "Defuzzification"

The recalled fit-vector output B is a weighted sum of the individual recalled vectors B'_k:

$$B = \sum_{k=1}^{m} w_k B'_k \quad , \tag{17}$$

where the nonnegative weight w_k summarizes the credibility or strength of the kth FAM rule (A_k, B_k). The credibility weights w_k are immediate candidates for adaptive modification. In practice we choose $w_1 = \ldots = w_m = 1$ as a default.

In principle, though not in practice, the recalled fit-vector output is a normalized sum of the B'_k fit vectors. This keeps the components of B unit-interval valued. We do not use normalization in practice because we invariably "defuzzify" the output distribution B to produce a single numerical output, a single value in the output universe of discourse $Y = \{y_1, \ldots, y_p\}$. The information in the output waveform B resides largely in the relative values of the membership degrees.

The simplest defuzzification scheme is to choose that element $y_{\max}$ that has maximal membership in the output fuzzy set B:

$$m_B(y_{\max}) = \max_{1 \le j \le k} m_B(y_j) \quad . \tag{18}$$

The popular probabilistic methods of maximum-likelihood and maximum-a-posteriori parameter estimation motivate this **maximum-membership defuzzification** scheme. The

maximum-membership scheme (18) is also computationally light.

There are two fundamental problems with the maximum-membership defuzzification scheme. First, the mode of the B distribution is not unique. This is especially troublesome with correlation-minimum encoding, as the representation (8) shows, and somewhat less troublesome with correlation-product encoding. Since the minimum operator clips off the top of the B_k fit vectors, the additively combined output fit vector B tends to be flat over many regions of universe of discourse Y. For continuous membership functions this leads to infinitely many modes. Even for quantized fuzzy sets, there may be many modes.

In practice we can average multiple modes. For large FAM banks of "independent" FAM rules, some form of the Central Limit Theorem (whose proof ultimately depends on Fourier transformability not probability) tends to apply. The waveform B tends to resemble a Gaussian membership function. So a unique mode tends to emerge. It tends to emerge with fewer samples if we use correlation-product encoding.

Second, the maximum-membership scheme ignores the information in much of the waveform B. Again correlation-minimum encoding compounds the problem. In practice B is often highly asymmetric, even if it is unimodal. Infinitely many output distributions can share the same mode.

The natural alternative is the **fuzzy centroid defuzzification** scheme. We directly compute the real-valued output as a normalized convex combination of fit values, the *fuzzy centroid* $\bar{B}$ of fit-vector B with respect to output space Y:

$$\bar{B} = \frac{\sum_{j=1}^{p} y_j \, m_B(y_j)}{\sum_{j=1}^{p} m_B(y_j)} \quad . \tag{19}$$

The fuzzy centroid is unique and uses all the information in the output distribution B. For symmetric unimodal distributions the mode and fuzzy centroid coincide. In many cases we must replace the discrete sums in (19) with integrals over continuously infinite spaces. We show in Chapter 19, though, that for libraries of trapezoidal fuzzy sets we can replace such a ratio of integrals with a ratio of simple discrete sums.

Note that computing the centroid (19) is the only step in the FAM inference procedure

that requires division. All other operations are inner products, pairwise minima, and additions. This promises realization in a fuzzy optical processor. Already some form of this FAM-inference scheme has led to digital [Togai, 1986] and analog [Yamakawa, 1987-88] VLSI circuitry.

FAM System Architecture

Figure 17.3 schematizes the architecture of the nonlinear FAM system F. Note that F maps fuzzy sets to fuzzy sets: $F(A) = B$. So F is in fact a fuzzy-system transformation $F : I^n \rightarrow I^p$. In practice A is a bit vector with one unity value, $a_i = 1$, and all other fit values zero, $a_j = 0$.

The output fuzzy set B is usually defuzzified with the centroid technique to produce an exact element y_j in the output universe of discourse Y. In effect defuzzification produces an output binary vector O, again with one element 1 and the rest 0s. At this level the FAM system F maps sets to sets, reducing the fuzzy system F to a mapping between Boolean cubes, $F : \{0,1\}^n \rightarrow \{0,1\}^p$. In many applications we model X and Y as continuous universes of discourse. So n and p are quite large. We shall call such systems **binary input-output FAMs**.

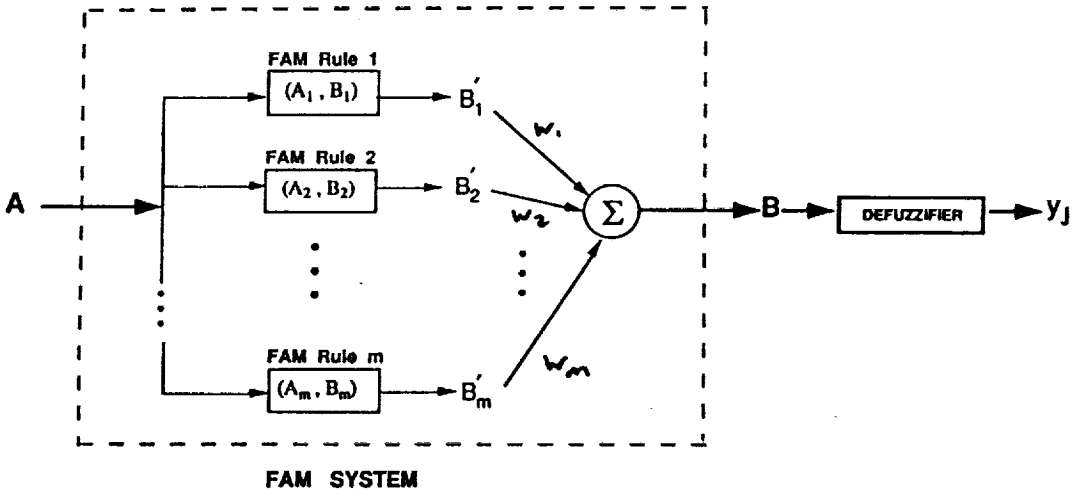


FIGURE 17.3 FAM system architecture. The FAM system F maps fuzzy sets in the unit cube I^n to fuzzy sets in the unit cube I^p. Binary input fuzzy sets are often used in practice to model exact input data. In general only an uncertainty estimate of the system state is available. So A is a proper fuzzy set. The user can defuzzify output fuzzy set B to yield exact output data, reducing the FAM system to a mapping between Boolean cubes.

Binary Input-Output FAMs: Inverted Pendulum Example

Binary input-output FAMs (BIOFAMs) are the most popular fuzzy systems for applications. BIOFAMs map system state-variable data to control data. In the case of traffic control, a BIOFAM maps traffic densities to green (and red) light durations.

BIOFAMs easily extend to multiple FAM rule antecedents, to mappings from product cubes to product cubes. There has been little theoretical justification for this extension,

aside from Mamdani's [1977] original suggestion to multiply relational matrices. The extension to multi-antecedent FAM rules is easier applied than formally explained. In the next section we present a general explanation for dealing with multi-antecedent FAM rules. First, though, we present the BIOFAM algorithm by illustrating it, and the FAM construction procedure, on an archetypical control problem.

Consider an inverted pendulum. In particular, consider how to adjust a motor to balance an inverted pendulum in two dimensions. The inverted pendulum is a classical control problem. It admits a math-model control solution. This provides a formal benchmark for BIOFAM pendulum controllers.

There are two state variables and one control variable. The first state variable is the *angle* θ that the pendulum shaft makes with the vertical. Zero angle corresponds to the vertical position. Positive angles are to the right of the vertical, negative angles to the left.

The second state variable is the *angular velocity* $\Delta\theta$. In practice we approximate the instantaneous angular velocity $\Delta\theta$ as the difference between the present angle measurement θ_t and the previous angle measurement θ_{t-1}:

$$\Delta\theta_t \;\; = \theta_t \; - \; \theta_{t-1} \quad .$$

The control variable is the motor current or *angular velocity* v_t. The velocity can also be positive or negative. We expect that if the pendulum falls to the right, the motor velocity should be negative to compensate. If the pendulum falls to the left, the motor velocity should be positive. If the pendulum successfully balances at the vertical, the motor velocity should be zero.

The real line R is the universe of discourse of the three variables. In practice we restrict each universe of discourse to a comparatively small interval, such as $[-90, 90]$ for the pendulum angle, centered about zero.

We can quantize each universe of discourse into five overlapping fuzzy sets. We know that the system variables can be positive, zero, or negative. We can quantize the magnitudes of the system variables finely or coarsely. Suppose we quantize the magnitudes as small, medium, and large. This leads to seven linguistic fuzzy set values:

NL: Negative Large

NM: Negative Medium

NS: Negative Small

ZE: Zero

PS: Positive Small

PM: Positive Medium

PL: Positive Large

For example, θ is a fuzzy *variable* that takes NL as a fuzzy set *value*. Different fuzzy quantizations of the angle universe of discourse allow the fuzzy variable θ to assume different fuzzy set values. The expressive power of the FAM approach stems from these fuzzy-set quantizations. In one stroke we reduce system dimensions, and we describe a nonlinear numerical process with linguistic common-sense terms.

We are not concerned with the exact shape of the fuzzy sets defined on each of the three universes of discourse. In practice the quantizing fuzzy sets are usually symmetric triangles or trapezoids centered about representive values. (We can think of such sets as *fuzzy numbers.*) The set ZE may be a Gaussian curve for the pendulum angle θ, a triangle for the angular velocity $\Delta\theta$, and a trapezoid for the velocity v. But all the ZE fuzzy sets will be centered about the numerical value zero, which will have maximum membership in the set of zero values.

How much should contiguous fuzzy sets overlap? This design issue depends on the problem at hand. Too much overlap blurs the distinction between the fuzzy set values. Too little overlap tends to resemble bivalent control, producing overshoot and undershoot. In Chapter 19 we determine experimentally the following default heuristic for ideal overlap: *Contiguous fuzzy sets in a library should overlap approximately* 25%.

FAM rules are triples, such as $(NM, Z;\ PM)$. They describe how to modify the control variable for observed values of the pendulum state variables. A FAM rule associates a motor-velocity fuzzy set value with a pendulum-angle fuzzy set value and an angular-velocity fuzzy set value. So we can interpret the triple $(NM, Z;\ PM)$ as the set-level

implication

IF the pendulum angle θ is negative but medium

AND the angular velocity $\Delta\theta$ is about zero ,

THEN the motor velocity should be positive but medium .

These commonsensical FAM rules are comparatively easy to articulate in natural language. Consider a terser linguistic version of the same three-antecedent FAM rule:

IF $\theta = NM$ AND $\Delta\theta = ZE$,

THEN $v = PM$.

Even this mild level of formalism may inhibit the knowledge acquisition process. On the other hand, the still terser FAM triple $(NM, ZE;\ PM)$ allows knowledge to be acquired simply by filling in a few entries in a linguistic FAM-bank matrix. In practice this often allows a working system to be developed in hours, if not minutes.

We specify the pendulum FAM system when we choose a FAM bank of two-antecedent FAM rules. Perhaps the first FAM rule to choose is the *steady-state FAM rule*: $(ZE, ZE;\ ZE)$. The steady-state FAM rule describes what to do in equilibrium. For the inverted pendulum we should do nothing.

This is typical of many control problems that require nulling a scalar error measure. We can control multivariable problems by nulling the norms of the system error vector and error-velocity vectors, or, better, by directly nulling the individual scalar variables. (Chapter 19 shows how error nulling can control a realtime target tracking system.) Error nulling tractably extends the FAM methodology to nonlinear estimation, control, and decision problems of high dimension.

The pendulum FAM bank is a 7-by-7 matrix with linguistic fuzzy-set entries. We index the columns by the seven fuzzy sets that quantize the angle θ universe of discourse. We index the rows by the seven fuzzy sets that quantize the angular velocity $\Delta\theta$ universe of discourse.

Each matrix entry is one of seven motor-velocity fuzzy-set values. Since a FAM rule is a mapping or function, there is exactly one output velocity value for every pair of angle and angular-velocity values. So the 49 entries in the FAM bank matrix represent the 49 possible two-antecedent FAM rules. In practice most of the entries are blank. In the adaptive FAM case discussed below, we adaptively generate the entries from process sample data.

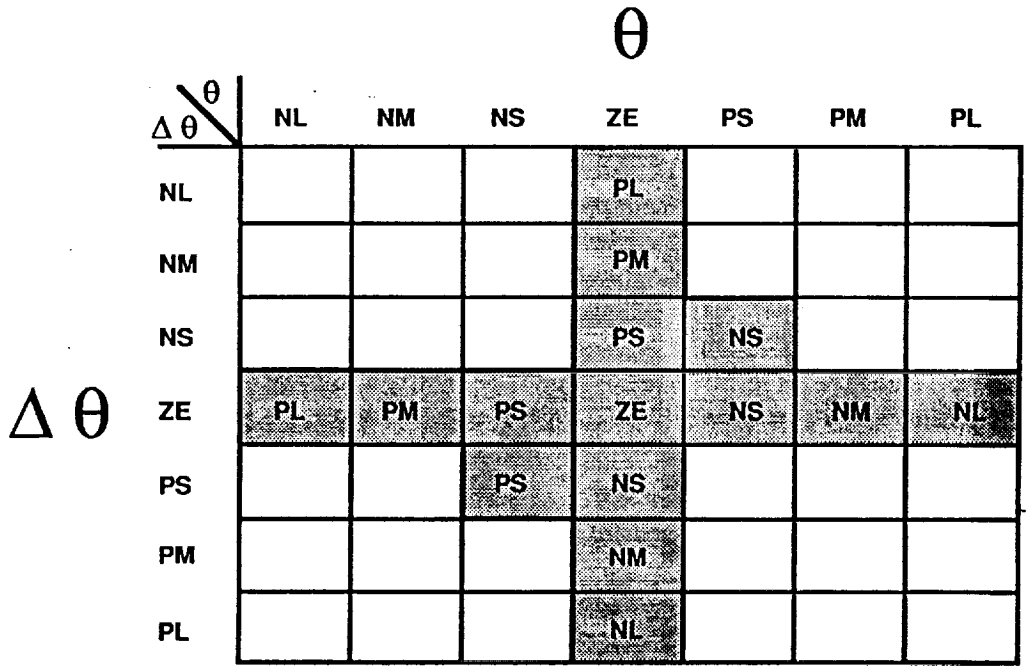
Commonsense dictates the entries in the pendulum FAM bank matrix. Suppose the pendulum is not changing. So $\Delta\theta = ZE$. If the pendulum is to the right of vertical, the motor velocity should be negative to compensate. The farther the pendulum is to the right, the larger the negative motor velocity should be. The motor velocity should be positive if the pendulum is to the left. So the fourth row of the FAM bank matrix, which corresponds to $\Delta\theta = ZE$, should be the ordinal inverse of the θ row values. This assignment includes the steady-state FAM rule $(ZE, ZE;\ ZE)$.

Now suppose the angle θ is zero but the pendulum is moving. If the angular velocity is negative, the pendulum will overshoot to the left. So the motor velocity should be positive to compensate. If the angular velocity is positive, the motor velocity should be negative. The greater the angular velocity is in magnitude, the greater the motor velocity should be in magnitude. So the fourth column of the FAM bank matrix, which corresponds to $\theta = ZE$, should be the ordinal inverse of the $\Delta\theta$ column values. This assignment also includes the steady-state FAM rule.

Positive θ values with negative $\Delta\theta$ values should produce negative motor velocity values, since the pendulum is heading toward the vertical. So $(PS, NS;\ NS)$ is a candidate FAM rule. Symmetrically, negative θ values with positive $\Delta\theta$ values should produce positive motor velocity values. So $(NS, PS;\ PS)$ is another candidate FAM rule.

This gives 15 FAM rules altogether. In practice these rules are more than sufficient to successfully balance an inverted pendulum. Different, and smaller, subsets of FAM rules may also successfully balance the pendulum.

We can represent the bank of 15 FAM rules as the 7-by-7 linguistic matrix



The BIOFAM system F also admits a geometric interpretation. The set of all possible input-outpairs $(\theta, \Delta\theta;\ F(\theta, \Delta\theta))$ defines a *FAM surface* in the input-output product space, in this case in R^3. We plot examples of these control surfaces in Chapters 18 and 19.

The BIOFAM *inference procedure* activates in parallel the antecedents of all 15 FAM rules. The binary or pulse nature of inputs picks off single fit values from the quantizing fuzzy sets. We can use either the correlation-minimum or correlation-product inferencing technique. For simplicity we shall illustrate the procedure with correlation-minimum inferencing.

Suppose the current pendulum angle θ is 15 degrees and the angular velocity $\Delta\theta$ is -10. This amounts to passing two bit vectors of one 1 and all else 0 through the BIOFAM system. What is the corresponding motor velocity value $v = F(15, -10)$?

Consider first how the input data pair (15, -10) activates steady-state FAM rule $(ZE, ZE;\ ZE)$. Suppose we define the antecedent and consequent fuzzy sets for ZE with the triangular fuzzy set membership functions in Figure 17.4. Then the angle datum 15 is a zero angle value to degree .2 : $m^{\theta}_{ZE}(15) = .2$. The angular velocity datum -10 is a zero

angular velocity value to degree .5 : $m_{ZE}^{\Delta\theta}(-10) = .5$.

We combine the antecedent fit values with minimum or maximum according as the antecedent fuzzy sets are combined with the conjunctive AND or the disjunctive OR. Intuitively, it should be at least as difficult to satisfy both antecedent conditions as to satisfy either one separately.

The FAM rule notation $(ZE, ZE;\ ZE)$ implicitly assumes that antecedent fuzzy sets are combined conjunctively with AND. So the data satisfy the compound antecedent of the FAM rule $(ZE, ZE;\ ZE)$ to degree

$$\begin{aligned} \min(m_{ZE}^{\theta}(15),\ m_{ZE}^{\Delta\theta}(-10)) &= \min(.2,\ .5) \\ &= .2 \quad . \end{aligned}$$

Clearly this methodology extends to any number of antecedent terms connected with arbitrary logical (set-theoretical) connectives.

The system should now activate the consequent fuzzy set of zero motor velocity values to degree .2. This is not the same as activating the ZE motor velocity fuzzy set 100% with probability .2, and certainly not the same as $\text{Prob}\{v = 0\} = .2$. Instead a deterministic 20% of ZE should result and, according to the additive combination formula (17), should be added to the final output fuzzy set.

The correlation-minimum inference procedure activates the angular velocity fuzzy set ZE to degree .2 by taking the pairwise minimum of .2 and the ZE fuzzy set m_{ZE}^{v}:

$$\min(m_{ZE}^{\theta}(15),\ m_{ZE}^{\Delta\theta}(-10)) \wedge m_{ZE}^{v}(v) = .2 \wedge m_{ZE}^{v}(v)$$

for all velocity values v. The correlation-product inference procedure would simply multiply the zero angular velocity fuzzy set by .2 : $.2\ m_{ZE}^{v}(v)$ for all v.

The data similarly activate the FAM rule $(PS, ZE;\ NS)$ depicted in Figure 17.4. The angle datum 15 is a small but positive angle value to degree .8. The angular velocity datum -10 is a zero angular velocity value to degree .5. So the output motor velocity fuzzy set of small but negative motor velocity values is scaled by .5, the lesser of the two antecedent fit values:

$$\min(m_{PS}^{\theta}(15),\ m_{ZE}^{\Delta\theta}(-10)) \wedge m_{NS}^{v}(v) \ = \ .5 \wedge m_{NS}^{v}(v)$$

for all velocity values v. So the data activate the FAM rule $(PS, ZE;\ NS)$ to greater degree than the steady-state FAM rule $(ZE, ZE;\ ZE)$ since in this example an angle value of 15 degrees is more a small but positive angle value than a zero angle value.

The data similarly activate the other 13 FAM rules. We combine the resulting minimum-scaled consequent fuzzy sets according to (17) by summing pointwise. We can then compute the fuzzy centroid with equation (19), with perhaps integrals replacing the discrete sums, to determine the specific output motor velocity v. In Chapter 19 we show that, for symmetric fuzzy sets of quantization, the centroid can always be computed exactly with simple discrete sums even if the fuzzy sets are continuous. In many realtime applications we must repeat this entire FAM inference procedure hundreds, perhaps thousands, of times per second. This requires fuzzy VLSI or optical processors.

Figure 17.4 illustrates this equal-weight additive combination procedure for just the FAM rules $(ZE, ZE;\ ZE)$ and $(PS, ZE;\ NS)$. The fuzzy-centroidal motor velocity value in this case is -3.

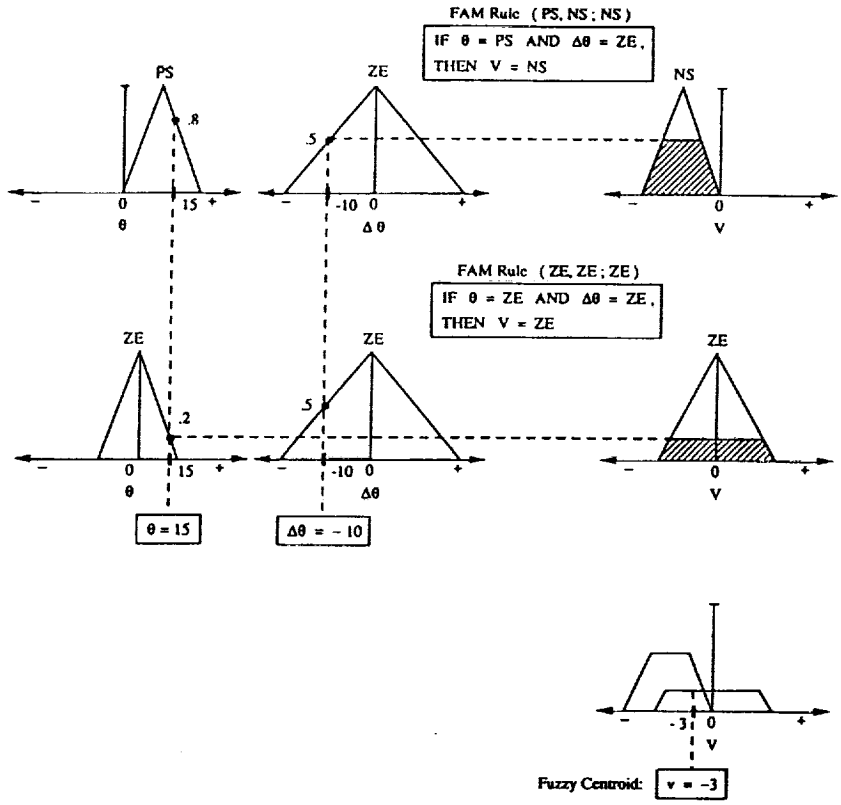


FIGURE 17.4 FAM correlation-minimum inference procedure. The FAM system consists of the two two-antecedent FAM rules $(PS, ZE;\ NS)$ and $(ZE, ZE;\ ZE)$. The input angle datum is 15, and is more a small but positive angle value than a zero angle value. The input angular velocity datum is -10, and is only a zero angular velocity value to degree .5. Antecedent fit values are combined with minimum since the antecedent terms are combined conjunctively with AND. The combined fit value then scales the consequent fuzzy set with pairwise minimum. The minimum-scaled output fuzzy sets are added pointwise. The fuzzy centroid of this output waveform is computed and yields the system output velocity value -3.

Multi-Antecedent FAM Rules: Decompositional Inference

The BIOFAM inference procedure treats antecedent fuzzy sets as if they were propositions with fuzzy truth values. This is because fuzzy logic corresponds to 1-dimensional

fuzzy set theory and because we use binary or exact inputs. We now formally develop the connection between BIOFAMs and the FAM theory presented earlier.

Consider the compound FAM rule "IF X is A AND Y is B , THEN C is Z," or $(A, B;\ C)$ for short. Let the universes of discourse X, Y, and Z have dimensions n, p, and q: $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_p\}$, and $Z = \{z_1, \ldots, z_q\}$. We can directly extend this framework to multiple antecedent and consequent terms.

In our notation X, Y, and Z are both universes of discourse and fuzzy variables. The fuzzy *variable* X can assume the fuzzy set *values* $A_1, A_2, \ldots$, and similarly for the fuzzy variables Y and Z. When controlling an inverted pendulum, the identification "X is A" might represent the natural-language description "The pendulum angle is positive but small."

What is the matrix representation of the FAM rule $(A, B;\ C)$? The question is nontrivial since A, B, and C are fuzzy subsets of different universes of discourse, points in different unit cubes. Their dimensions and interpretations differ. Mamdani [1977] and others have suggested representing such rules as fuzzy multidimensional relations or arrays. Then the FAM rule $(A, B;\ C)$ would be a fuzzy subset of the product space $X \times Y \times Z$. This representation is not used in practice since only exact inputs are presented to FAM systems and the BIOFAM procedure applies. If we presented the system with a genuine fuzzy set input, we would no doubt preprocess the fuzzy set with a centroidal or maximum-fit-value technique so we could still apply the BIOFAM inference procedure.

We present an alternative representation that decomposes, then recomposes, the FAM rule $(A, B;\ C)$ in accord with the FAM inference procedure. This representation allows neural networks to adaptively estimate, store, and modify the decomposed FAM rules. The representation requires far less storage than the multidimensional-array representation.

Let the fuzzy Hebb matrices M_{AC} and M_{BC} store the simple FAM associations (A, C) and (B, C):

$$M_{AC} = A^T \circ C \quad , \tag{20}$$

$$M_{BC} = B^T \circ C \quad . \tag{21}$$

The fuzzy Hebb matrices M_{AC} and M_{BC} *split* the compound FAM rule $(A, B;\ C)$. We can construct the splitting matrices with correlation-product encoding.

Let $I_X^i = (0 \ldots 0\ 1\ 0 \ldots 0)$ be an n-dimensional bit vector with ith element 1 and all other elements 0. I_X^i is the ith row of the n-by-n identity matrix. Similarly, I_Y^j and I_Z^k are the respective jth and kth rows of the p-by-p and q-by-q identity matrices. The bit vector I_X^i represents the occurrence of the exact input x_i.

We will call the proposed FAM representation scheme **FAM decompositional inference**, in the spirit of the max-min compositional inference scheme discussed above. FAM decompositional inference *decomposes* the compound FAM rule $(A, B;\ C)$ into the component rules (A, C) and (B, C). The simpler component rules are processed in parallel. New fuzzy set inputs A' and B' pass through the FAM matrices M_{AC} and M_{BC}. Max-min composition then gives the recalled fuzzy sets $C_{A'}$ and $C_{B'}$:

$$C_{A'} = A' \circ M_{AC} \quad , \tag{22}$$

$$C_{B'} = B' \circ M_{BC} \quad . \tag{23}$$

The trick is to *recompose* the fuzzy sets $C_{A'}$ and $C_{B'}$ with intersection or union according as the antecedent terms "X is A" and "Y is B" are combined with AND or OR. The negated antecedent term "X is NOT A" requires forming the set complement $C_{A'}^c$ for input fuzzy set A'.

Suppose we present the new inputs A' and B' to the single-FAM-rule system F that stores the FAM rule $(A, B;\ C)$. Then the recalled output fuzzy set C' equals the intersection of $C_{A'}$ and $C_{B'}$:

$$\begin{aligned} F(A',\ B') &= [A' \circ M_{AC}] \cap [B' \circ M_{BC}] \\ &= C_{A'} \cap C_{B'} \\ &= C' \quad . \end{aligned} \tag{24}$$

We can then defuzzify C', if we wish, to yield the exact output I_Z^k.

The logical connectives apply to the antecedent terms of different dimension and meaning. Decompositional inference applies the set-theoretic analogues of the logical connectives to subsets of Z. Of course all subsets C' of Z have the same dimension and meaning.

We now prove that decompositional inference generalizes BIOFAM inference. This generalization is not simply formal. It opens an immediate path to adaptation with arbitrary neural network techniques.

Suppose we present the exact inputs x_i and y_j to the single-FAM-rule system F that stores $(A, B;\ C)$. So we present the unit bit vectors I_X^i and I_Y^j to F as nonfuzzy set inputs. Then

$$\begin{aligned} F(x_i, y_j) \;=\; F(I_X^i,\ I_Y^j) \;&=\; [I_X^i \circ M_{AC}] \;\cap\; [I_Y^j \circ M_{BC}] \\ &=\; a_i \wedge C \;\cap\; b_j \wedge C \qquad (25) \\ &=\; \min(a_i,\ b_j) \wedge C \quad . \qquad (26) \end{aligned}$$

(25) follows from (8). Representing C with its membership function m_C, (26) is equivalent to the BIOFAM prescription

$$\min(a_i,\ b_j) \wedge m_C(z) \qquad (27)$$

for all z in Z.

If we encode the simple FAM rules (A, C) and (B, C) with correlation-product encoding, decompositional inference gives the BIOFAM version of correlation-product inference:

$$\begin{aligned} F(I_X^i, I_Y^j) \;&=\; [I_X^i \circ A^T C] \;\cap\; [I_Y^j \circ B^T C] \\ &=\; a_i\, C \;\cap\; b_j\, C \qquad (28) \\ &=\; \min(a_i,\ b_j)\, C \qquad (29) \\ &=\; \min(a_i,\ b_j)\, m_C(z) \qquad (30) \end{aligned}$$

for all z in Z. (13) implies (28). $\min(a_i\, c_k,\ b_j\, c_k) \;=\; \min(a_i,\ b_j)\, c_k$ implies (29).

Decompositional inference allows arbitrary fuzzy sets, waveforms, or distributions A' and B' to be applied to a FAM system. The FAM system can house an arbitrary FAM bank of compound FAM rules. If we use the FAM system to control a process, the input fuzzy sets A' and B' can be the output of an independent state-*estimation* system, such as a Kalman filter. A' and B' might then represent probability distributions on the exact input spaces X and Y. The filter-controller cascade is a common engineering architecture.

We can split compound consequents as desired. We can split the compound FAM rule "IF X is A AND Y is B, THEN Z is C OR W is D," or $(A, B;\ C, D)$, into the FAM rules $(A, B;\ C)$ and $(A, B;\ D)$. We can use the same split if the consequent logical connective is AND.

We can give a propositional-calculus justification for the decompositional inference technique. Let A, B, and C be bivalent propositions with truth values $t(A)$, $t(B)$, and $t(C)$ in $\{0, 1\}$. Then we can construct truth tables to prove the two consequent-splitting tautologies that we use in decompositional inference:

$$[A \longrightarrow (B \text{ OR } C)] \longrightarrow [(A \longrightarrow B) \text{ OR } (A \longrightarrow C)] , \tag{31}$$

$$[A \longrightarrow (B \text{ AND } C)] \longrightarrow [(A \longrightarrow B) \text{ AND } (A \longrightarrow C)] , \tag{32}$$

where the arrow represents logical implication.

In bivalent logic, the implication $A \rightarrow B$ is false iff the antecedent A is true and the consequent B is false. Equivalently, $t(A \rightarrow B) = 1$ iff $t(A) = 1$ and $t(B) = 0$. This allows a "brief" truth table to be constructed to check for validity. We chose truth values for the terms in the consequent of the overall implication (31) or (32) to make the consequent false. Given those restrictions, if we cannot find truth values to make the antecedent true, the statement is a tautology. In (31), if $t((A \rightarrow B) \text{ OR } (A \rightarrow C)) = 0$, then $t(A) = 1$ and $t(B) = t(C) = 0$, since a disjunction is false iff both disjuncts are false. This forces the antecedent $A \rightarrow (B \text{ OR } C)$ to be false. So (31) is a **tautology**: It is true in all cases.

We can also justify splitting the compound FAM rule "IF X is A OR Y is B, THEN Z is C" into the disjunction (union) of the two simple FAM rules "IF X is A,

THEN Z is C " and "IF Y is B , THEN Z is C " with a propositional tautology:

$$[(A \text{ OR } B) \longrightarrow C] \longrightarrow [(A \longrightarrow C) \text{ OR } (B \longrightarrow C)] \ . \tag{33}$$

Now consider splitting the original compound FAM rule "IF X is A AND Y is B , THEN Z is C " into the conjunction (intersection) of the two simple FAM rules "IF X is A , THEN Z is C " and "IF Y is B , THEN Z is C ." A problem arises when we examine the truth table of the corresponding proposition

$$[(A \text{ AND } B) \longrightarrow C] \longrightarrow [(A \longrightarrow C) \text{ AND } (B \longrightarrow C)] \ . \tag{34}$$

The problem is that (34) is not always true, and hence not a tautology. The implication is false if A is true and B and C are false, or if A and C are false and B is true. But the implication (34) is valid if *both* antecedent terms A and B are true. So if $t(A) = t(B) = 1$, the compound conditional $(A \text{ AND } B) \rightarrow C$ implies both $A \rightarrow C$ and $B \rightarrow C$.

The simultaneous occurrence of the data values x_i and y_j satisfies this condition. Recall that logic is 1-dimensional set theory. The condition $t(A) = t(B) = 1$ is given by the 1 in I_X^i and the 1 in I_X^j. We can interpret the unit bit vectors I_X^i and I_Y^j as the (true) bivalent propositions "X is x_i" and "Y is y_j." Propositional logic applies coordinate-wise. A similar argument holds for the converse of (33).

For general fuzzy set inputs A' and B' the argument still holds in the sense of continuous-valued logic. But the truth values of the logical implications may be less than unity while greater than zero. If A' is a null vector and B' is not, or vice versa, the implication (34) is false coordinate-wise, at least if one coordinate of the non-null vector is unity. But in this case the decompositional inference scheme yields an output null vector C'. In effect the FAM system indicates the propositional falsehood.

Adaptive Decompositional Inference

The decompositional inference scheme allows the *splitting matrices* M_{AC} and M_{BC} to

be arbitrary. Indeed it allows them to be eliminated altogether.

Let $N_X : \ I^n \rightarrow I^q$ be an arbitrary *neural network* system that maps fuzzy subsets A' of X to fuzzy subsets C' of Z. $N_Y : \ I^p \rightarrow I^q$ can be a different neural network. In general N_X and N_Y are time-varying.

The adaptive decompositional inference (ADI) scheme allows compound FAM rules to be adaptively split, stored, and modified by arbitrary neural networks. The compound FAM rule "IF X is A AND Y is B, THEN Z is C," or $(A, B;\ C)$, can be split by N_X and N_Y. N_X can house the simple FAM association (A, C). N_Y can house (B, C). Then for arbitrary fuzzy set inputs A' and B', ADI proceeds as before for an adaptive FAM system $F : \ I^n \times I^p \rightarrow I^q$ that houses the FAM rule $(A, B;\ C)$ or a bank of such FAM rules:

$$\begin{aligned} F(A', B') &= N_X(A') \cap N_Y(B') \\ &= C_{A'} \cap C_{B'} \\ &= C' \quad . \end{aligned} \tag{35}$$

Any neural network technique can be used. A reasonable candidate for many unstructured problems is the backpropagation algorithm applied to several small feedforward multilayer networks. The primary concerns are space and training time. Several small neural networks can often be trained in parallel faster, and more accurately, than a single large neural network.

The ADI approach illustrates one way neural algorithms can be embedded in a FAM architecture. Below we discuss another way that uses unsupervised clustering algorithms.

ADAPTIVE FAMs: PRODUCT-SPACE CLUSTERING IN FAM CELLS

An **adaptive FAM (AFAM)** is a time-varying mapping between fuzzy cubes. In principle the adaptive decompositional inference technique generates AFAMs. But we

shall reserve the label AFAM for systems that generate FAM rules from training data but that do not require splitting and recombining FAM data.

We propose a geometric AFAM procedure. The procedure adaptively clusters training samples in the FAM system *input-output product space.* FAM mappings are balls or clusters in the input-output product space. These clusters are simply the fuzzy Hebb matrices discussed above. The procedure "blindly" generates weighted FAM rules from training data. Further training modifies the weighted set of FAM rules. We call this unsupervised procedure **product-space clustering.**

Consider first a discrete 1-dimensional FAM system $S: \ I^n \rightarrow I^p$. Then a FAM rule has the form "IF X is A_i, THEN Y is B_i" or (A_i, B_i). The input-output product space is $I^n \times I^p$.

What does the FAM rule (A_i, B_i) look like in the product space $I^n \times I^p$? It looks like a cluster of points centered at the numerical point (A_i, B_i). The FAM system maps points A near A_i to points B near B_i. The closer A is to A_i, the closer the point (A, B) is to the point (A_i, B_i) in the product space $I^n \times I^p$. In this sense FAMs map balls in I^n to balls in I^p. The notation is ambiguous since (A_i, B_i) stands for both the FAM rule mapping, or fuzzy subset of $I^n \times I^p$, and the numerical fit-vector point in $I^n \times I^p$.

Adaptive clustering algorithms can estimate the unknown FAM rule (A_i, B_i) from training samples of the form (A, B). In general there are m unknown FAM rules $(A_1, B_1), \ldots, (A_m, B_m)$. The number m of FAM rules is also unknown. The user may select m arbitrarily in many applications.

Competitive *adaptive vector quantization* (AVQ) algorithms can adaptively estimate both the unknown FAM rules (A_i, B_i) and the unknown number m of FAM rules from FAM system input-output data. The AVQ algorithms do not require fuzzy-set data. Scalar BIOFAM data suffices, as we illustrate below for adaptive estimation of inverted-pendulum control FAM rules.

Suppose the r fuzzy sets $A_1, \ldots, A_r$ quantize the input universe of discourse X. The s fuzzy sets $B_1, \ldots, B_s$ quantize the output universe of discourse Y. In general r and s are unrelated to each other and to the number m of FAM rules (A_i, B_i). The user must specify r and s and the shape of the fuzzy sets A_i and B_i. In practice this is not difficult.

Quantizing fuzzy sets are usually trapezoidal, and r and s are less than 10.

The quantizing collections $\{A_i\}$ and $\{B_j\}$ define rs FAM cells F_{ij} in the input-output product space $I^n \times I^p$. The FAM cells F_{ij} overlap since contiguous quantizing fuzzy sets A_i and A_{i+1}, and B_j and B_{j+1}, overlap. So the FAM cell collection $\{F_{ij}\}$ does not partition the product space $I^n \times I^p$. The union of all FAM cells also does not equal $I^n \times I^p$ since the patches F_{ij} are fuzzy subsets of $I^n \times I^p$. The union provides only a fuzzy "cover" for $I^n \times I^p$.

The *fuzzy Cartesian product* $A_i \times B_i$ defines the FAM cell F_{ij}. $A_i \times B_i$ is just the fuzzy outer product $A_i^T \circ B_i$ in (6) or the correlation product $A_i^T \; B_i$ in (12). So a FAM cell F_{ij} is simply the fuzzy correlation-minimum or correlation-product matrix M_{ij} : $F_{ij} = M_{ij}$.

Adaptive FAM Rule Generation

Let $\mathbf{m}_1, \ldots, \mathbf{m}_k$ be k quantization vectors in the input-output product space $I^n \times I^p$ or, equivalently, in I^{n+p}. $\mathbf{m}_j$ is the jth column of the synaptic connection matrix $\mathbf{M}$. $\mathbf{M}$ has $n + p$ rows and k columns.

Suppose, for instance, $\mathbf{m}_j$ changes in time according to the differential competitive learning (DCL) AVQ algorithm discussed in Chapters 6 and 9. The competitive system samples concatenated fuzzy set samples of the form $[A|B]$. The augmented fuzzy set $[A|B]$ is a point in the unit hypercube I^{n+p}.

The synaptic vectors $\mathbf{m}_j$ converge to FAM matrix centroids in $I^n \times I^p$. More generally they estimate the density or distribution of the FAM rules in $I^n \times I^p$. The quantizing synaptic vectors naturally weight the estimated FAM rule. The more synaptic vectors clustered about a centroidal FAM rule, the greater its weight w_i in (17).

Suppose there are 15 FAM-rule centroids in $I^n \times I^p$ and $k > 15$. Suppose k_i synaptic vectors $\mathbf{m}_j$ cluster around the ith centroid. So $k_1 + \ldots + k_{15} = k$. Suppose the *cluster counts* k_i are ordered as

$$k_1 \geq k_2 \geq \ldots k_{15} \quad . \tag{36}$$

The first centroidal FAM rule is as at least as frequent as the second centroidal FAM rule, and so on. This gives the adaptive FAM-rule weighting scheme

$$w_i = \frac{k_i}{k} \quad . \tag{37}$$

The FAM rule weights w_i evolve in time as new augmented fuzzy sets $[A|B]$ are sampled. In practice we may want only the 15 most-frequent FAM rules or only the FAM rules with at least some minimum frequency $w_{\min}$. Then (37) provides a quantitative solution.

Geometrically we count the number k_{ij} of quantizing vectors in each FAM cell F_{ij}. We can define FAM-cell boundaries in advance. High-count FAM cells outrank low-count FAM cells. Most FAM cells contain zero or few synaptic vectors.

Product-space clustering extends to compound FAM rules and product spaces. The FAM rule "IF X is A AND Y is B, THEN Z is C", or $(A,\ B;\ C)$, is a point in $I^n \times I^p \times I^q$. The t fuzzy sets $C_1, \ldots, C_t$ quantize the new output space Z. There are rst FAM cells F_{ijk}. (36) and (37) extend similarly. X, Y, and Z can be continuous. The adaptive clustering procedure extends to any number of FAM-rule antecedent terms.

Adaptive BIOFAM Clustering

BIOFAM data clusters more efficiently than fuzzy-set FAM data. Paired numbers are easier to process and obtain than paired fit vectors. This allows system input-output data to directly generate FAM systems.

In control applications, human or automatic controllers generate streams of "well-controlled" system input-output data. Adaptive BIOFAM clustering converts this data to weighted FAM rules. The adaptive system transduces behavioral data to behavioral rules. The fuzzy system learns causal patterns. It learns which control inputs cause which control outputs. The system approximates these causal patterns when it acts as the controller.

Adaptive BIOFAMs cluster in the input-output product space $X \times Y$. The product space $X \times Y$ is vastly smaller than the power-set product space $I^n \times I^p$ used above. The

adaptive synaptic vectors $\mathbf{m}_j$ are now 2-dimensional instead of $n + p$-dimensional. On the other hand, competitive BIOFAM clustering requires many more input-output data pairs $(x_i, y_i) \in R^2$ than augmented fuzzy-set samples $[A|B] \in I^{n+p}$.

Again our notation is ambiguous. We now use x_i as the numerical sample from X at sample time i. Earlier x_i denoted the ith ordered element in the finite nonfuzzy set $X = \{x_1, \ldots, x_n\}$. One advantage is X can be continuous, say R^n.

BIOFAM clustering counts synaptic quantization vectors in FAM cells. The system samples the nonfuzzy input-output stream $(x_1, y_1), (x_2, y_2), \ldots$ Unsupervised competitive learning distributes the k synaptic quantization vectors $\mathbf{m}_1, \ldots, \mathbf{m}_k$ in $X \times Y$. Learning distributes them to different FAM cells F_{ij}. The FAM cells F_{ij} overlap but are nonfuzzy subcubes of $X \times Y$. The BIOFAM FAM cells F_{ij} cover $X \times Y$.

F_{ij} contains k_{ij} quantization vectors at each sample time. The cell counts k_{ij} define a frequency *histogram* since all k_{ij} sum to k. So $w_{ij} = \frac{k_{ij}}{k}$ weights the FAM rule "IF X is A_i, THEN Y is B_j."

Suppose the pairwise-overlapping fuzzy sets NL, NM, NS, ZE, PS, PM, PL quantize the input space X. Suppose seven similar fuzzy sets quantize the output space Y. We can define the fuzzy sets arbitrarily. In practice they are normal and trapezoidal. (The boundary fuzzy sets NL and PL are ramp functions.) X and Y may each be the real line. A typical FAM rule is "IF X is NL, THEN Y is PS."

Input datum x_i is nonfuzzy. When $X = x_i$ holds, the relations $X = NL, \ldots, X = PL$ hold to different degrees. Most hold to degree zero. $X = NM$ holds to degree $m_{NM}(x_i)$. Input datum x_i partially activates the FAM rule "IF X is NM, THEN Y is ZE" or, equivalently, $(NM;\ ZE)$. Since the FAM rules have single antecedents, x_i activates the consequent fuzzy set ZE to degree $m_{NM}(x_i)$ as well. Multi-antecedent FAM rules activate output consequent sets according to a logic-based function of antecedent term membership values, as discussed above on BIOFAM inference.

Suppose Figure 17.5 represents the input-output data stream $(x_1, y_1), (x_2, y_2), \ldots$ in the planar product space $X \times Y$:

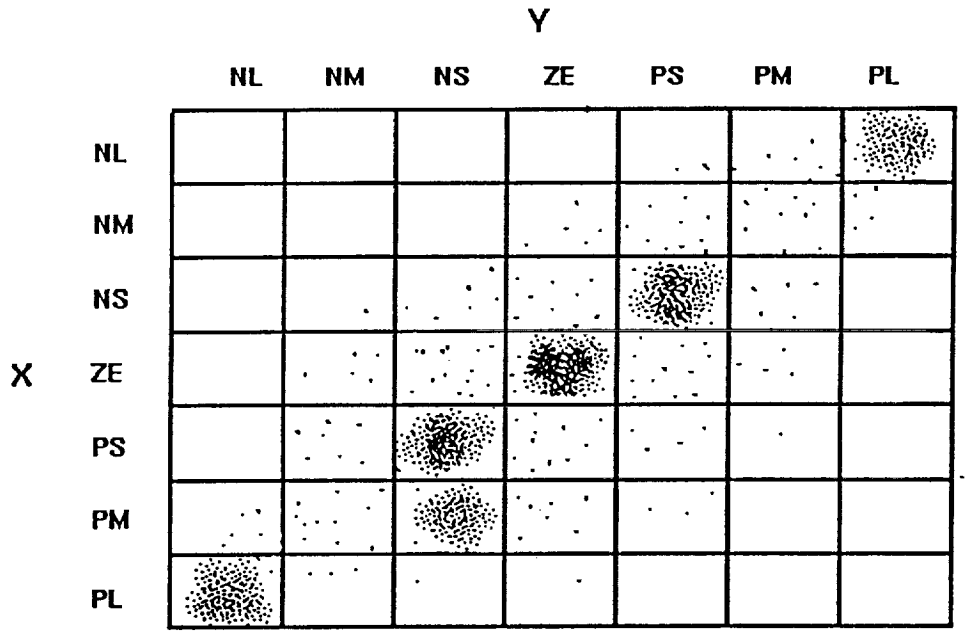


FIGURE 17.5 Distribution of input-output data (x_i, y_i) in the input-output product space $X \times Y$. Data clusters reflect FAM rules, such as the steady-state FAM rule "IF X is ZE, THEN Y is ZE".

Suppose the sample data in Figure 17.5 trains a DCL system. Suppose such competitive learning distributes ten 2-dimensional synaptic vectors $\mathbf{m}_1, \ldots, \mathbf{m}_{10}$ as in Figure 17.6:

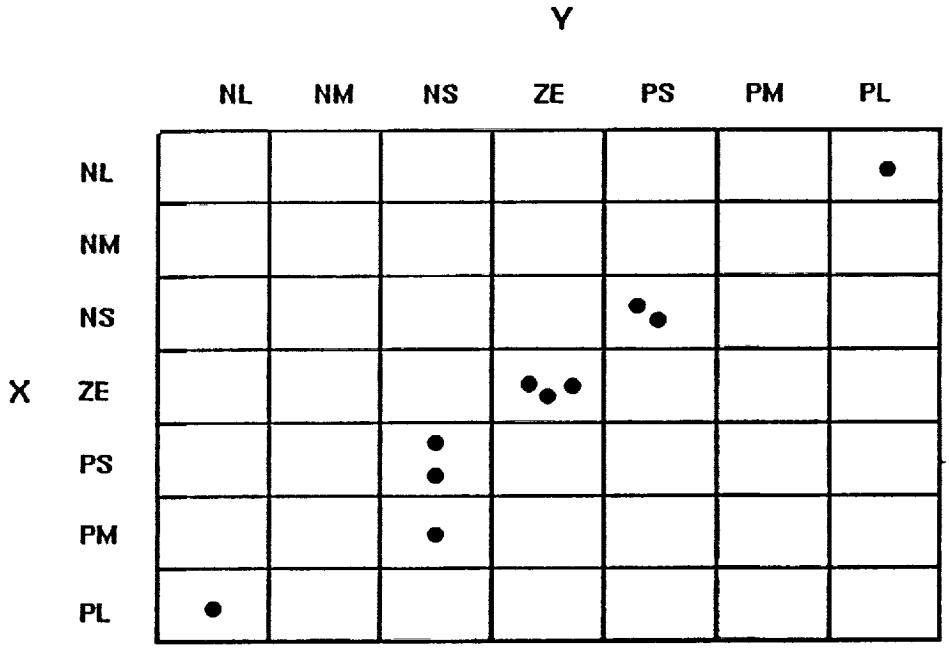


FIGURE 17.6 Distribution of ten 2-dimensional synaptic quantization vectors $\mathbf{m}_1, \ldots, \mathbf{m}_{10}$ in the input-output product space $X \times Y$. As the FAM system samples nonfuzzy data (x_i, y_i), competitive learning distributes the synaptic vectors in $X \times Y$. The synaptic vectors estimate the frequency distribution of the sampled input-output data, and thus estimate FAM rules.

FAM cells do not overlap in Figures 17.5 and 17.6 for convenience's sake. The corresponding quantizing fuzzy sets touch but do not overlap.

Figure 17.5 reveals six sample-data clusters. The six quantization-vector clusters in Figure 17.6 estimate the six sample-data clusters. The single synaptic vector in FAM cell $(PM;\ NS)$ indicates a smaller cluster. Since $k = 10$, the number of quantization vectors in each FAM cell measures the percentage or frequency weight w_{ij} of each possible FAM rule.

In general the additive combination rule (17) does not require normalizing the quantization-vector count k_{ij}. $w_{ij} = k_{ij}$ is acceptable. This holds for both maximum-membership defuzzification (18) and fuzzy centroid defuzzification (19). These defuzzification schemes prohibit only negative weight values.

The ten quantization vectors in Figure 17.6 estimate at most six FAM rules. From most to least frequent or "important", the FAM rules are $(ZE;\ ZE)$, $(PS;\ NS)$, $(NS;\ PS)$, $(PM;\ NS)$, $(PL;\ NL)$, and $(NL;\ PL)$. These FAM rules suggest that fuzzy variable X is an error variable or an error velocity variable since the steady-state FAM rule $(ZE;\ ZE)$ is most important. If we sample a system only in steady-state equilibrium, we will estimate only the steady-state FAM rule. We can accurately estimate the FAM system's global behavior only if we representatively sample the system's input-output behavior.

The "corner" FAM rules $(PL;\ NL)$ and $(NL;\ PL)$ may be more important than their frequencies suggest. The boundary sets Negative Large (NL) and Positive Large (PL) are usually defined as ramp functions, as negatively and positively sloped lines. NL and PL alone cover the important end-point regions of the universe of discourse X. They give $m_{NL}(x) = m_{PL}(x) = 1$ only if x is at or near the end-point of X, since NL and PL are ramp functions not trapezoids. NL and PL cover these end-point regions "briefly". Their corresponding FAM cells tend to be smaller than the other FAM cells. The end-point regions must be covered in most control problems, especially error nulling problems like stabilizing an inverted pendulum. The user can weight these FAM-cell counts more highly, for instance $w_{ij} = c\ k_{ij}$ for scaling constant $c > 0$. Or the user can simply include these end-point FAM rules in every operative FAM bank.

Most FAM cells do not generate FAM rules. More accurately, we estimate every possible FAM rule but usually with zero or near-zero frequency weight w_{ij}. For large numbers of multiple FAM-rule antecedents, system input-output data streams through comparatively few FAM cells. Structured trajectories in $X \times Y$ are few.

A FAM-rule's mapping structure also limits the number of estimated FAM rules. A FAM rule maps fuzzy sets in I^n or $F(2^X)$ to fuzzy sets in I^p or $F(2^Y)$. A fuzzy associative memory maps every domain fuzzy set A to a unique range fuzzy set B. Fuzzy set A cannot map to multiple fuzzy sets B, B', B'', and so on. We write the FAM rule as $(A;\ B)$ not

$(A;\ B$ or B' or B'' or $,..)$. So we estimate *at most* one rule per FAM-cell row in Figure 17.6.

If two FAM cells in a row are equally and highly frequent, we can pick arbitrarily either FAM rule to include in the FAM bank. This occurs infrequently but can occur. In principle we could estimate the FAM rule as a compound FAM rule with a disjunctive consequent. The simplest strategy picks only the highest frequency FAM cell per row.

The user can estimate FAM rules without counting the quantization vectors in each FAM cell. There may be too many FAM cells to search at each estimation iteration. The user never need examine FAM cells. Instead the user checks the synaptic vector components m_{ij}. The user defines in advance fuzzy-set intervals, such as $[l_{NL}, u_{NL}]$ for NL. If $l_{NL} \leq m_{ij} \leq u_{NL}$, then the FAM-antecedent reads "IF X is NL."

Suppose the input and output spaces X and Y are the same, the real interval $[-35,\ 35]$. Suppose we partition X and Y into the same seven disjoint fuzzy sets:

NL = [-35, -25]
NM = [-25, -15]
NS = [-15, -5]
ZE = [-5, 5]
PS = [5, 15]
PM = [15, 25]
PL = [25, 35] .

Then the observed synaptic vector $\mathbf{m}_j = [9,\ -10]$ increases the count of FAM cell $PS \times NS$ and increases the weight of FAM rule "IF X is PS, THEN Y is NS."

This amounts to nearest-neighbor classification of synaptic quantization vectors. We assign quantization vector $\mathbf{m}_k$ to FAM cell F_{ij} iff $\mathbf{m}_k$ is closer to the centroid of F_{ij} than to all other FAM-cell centroids. We break ties arbitrarily. Centroid classification allows the FAM cells to overlap.

Adaptive BIOFAM Example: Inverted Pendulum

We used DCL to train an AFAM to control the inverted pendulum discussed above. We used the accompanying C-software to generate 1,000 pendulum trajectory data. These product-space training vectors (θ, $\Delta\theta$, v) were points in R^3. Pendulum angle θ data ranged between -90 and 90. Pendulum angular veclocity $\Delta\theta$ data ranged from -150 to 150.

We defined FAM cells by uniformly partitioning the effective product space. Fuzzy variables could assume only the five fuzzy set values NM, NS, ZE, PS, and PM. So there were 125 possible FAM rules. For instance, the steady-state FAM rule took the form (ZE, ZE; ZE) or, more completely, "IF $\theta = ZE$ AND $\Delta\theta = ZE$, THEN $v = ZE$."

A BIOFAM controlled the inverted pendulum. The BIOFAM restored the pendulum to equilibrium as we knocked it over to the right and to the left. (Function keys F9 and F10 knock the pendulum over to the left and to the right. Input-output sample data reads automatically to a training data file.) Eleven FAM rules described the BIOFAM controller. Figure 17.1 displays this FAM bank. Observe that the zero (ZE) row and column are ordinal inverses of the respective row and column indices.

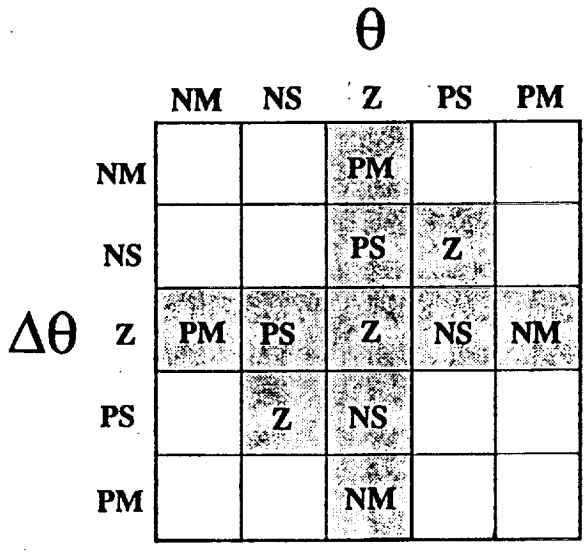


FIGURE 17.7 Inverted-pendulum FAM bank used in simulation. This

BIOFAM generated 1,000 sample vectors of the form $(\theta,\ \Delta\theta,\ v)$.

We trained 125 3-dimensional synaptic quantization vectors with differential competitive learning, as discussed in Chapters 4,6, and 9. In principle the 125 synaptic vectors could describe a uniform distribution of product-space trajectory data. Then the 125 FAM cells would each contain one synaptic vector. Alternatively, if we used a vertically stabilized pendulum to generate the 1,000 training vectors, all 125 synaptic vectors would concentrate in the $(ZE,\ ZE;\ ZE)$ FAM cell. This would still be true if we only mildly perturbed the pendulum from vertical equilibrium.

DCL distributed the 125 synaptic vectors to 13 FAM cells. So we estimated 13 FAM rules. Some FAM cells contained more synaptic vectors than others. Figure 17.8 displays the synaptic-vector histogram after the DCL samples the 1,000 samples. Actually Figure 17.8 displays a truncated histogram. The horizontal axis should list all 125 FAM cells, all 125 FAM-rule weights w_k in (17). The missing 112 entries have zero synaptic-vector frequency.

Figure 17.8 gives a snapshot of the adaptive process. In practice, and in principle, successive data gradually modify the histogram. "Good" training samples should include a significant number of equilibrium samples. In Figure 17.8 the steady-state FAM cell $(ZE,\ ZE;\ ZE)$ is clearly the most frequent.

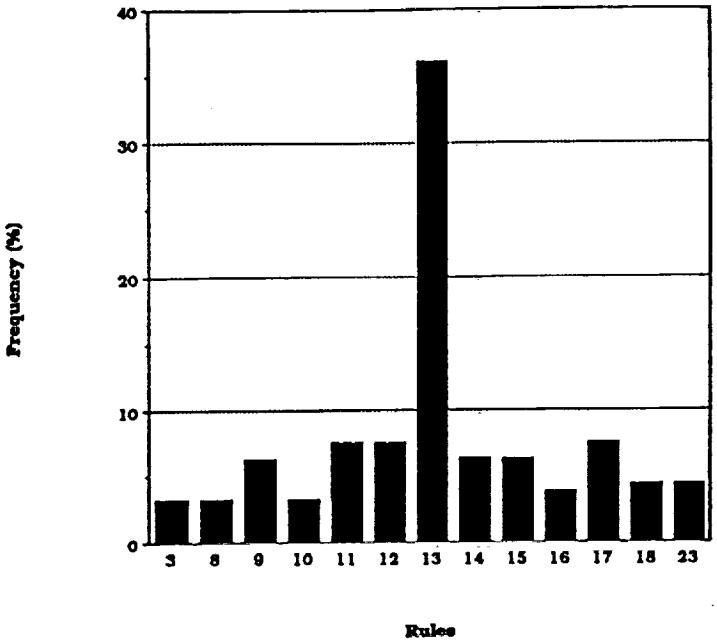


FIGURE 17.8 Synaptic-vector histogram. Differential competitive learning allocated 125 3-dimensional synaptic vectors to the 125 FAM cells. Here the adaptive system has sampled 1,000 representative pendulum-control data. DCL allocates the synaptic vectors to only 13 FAM cells. The steady-state FAM cell (ZE, ZE; ZE) is most frequent.

Figure 17.9 displays the DCL-estimated FAM bank. The product-space clustering method rapidly recovered the 11 original FAM rules. It also estimated the two additional FAM rules (PS, NM; ZE) and (NS, PM; ZE), which did not affect the BIOFAM system's performance. The estimated FAM bank defined a BIOFAM, with all 13 FAM-rule weights set w_k equal to unity, that controlled the pendulum as well as the original BIOFAM did.

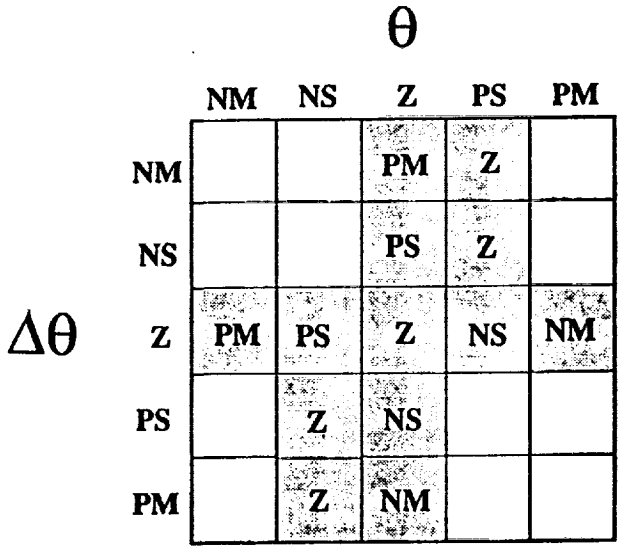


FIGURE 17.9 DCL-estimated FAM bank. Product-space clustering recovered the original 11 FAM rules and estimated two new FAM rules. The new and original BIOFAM systems controlled the inverted pendulum equally well.

In nonrealtime applications we can in principle omit the adaptive step altogether. We can directly compute the FAM-cell histogram if we exhaustively count all sampled data. Then the (growing) number of synaptic vectors equals the number of training samples. This procedure equally weights all samples, and so tends not to "track" an evolving process. Competitive learning weights more recent samples more heavily. Competitive learning's metrical-classification step also helps filter noise from the stream of sample data.

REFERENCES

Dubois, D., Prade, H., *Fuzzy Sets and Systems: Theory and Applications*, Academic Press, New York, 1980.

Hebb, D. *The Organization of Behavior*, Wiley, 1949.

Klir, G.J., Foger, T.A., *Fuzzy Sets, Uncertainty, and Information*, Prentice-Hall, 1988.

Kosko, B., "Fuzzy Knowledge Combination," *International Journal of Intelligent Systems*, vol. 1, 293 - 320, 1986.

Kosko, B., "Fuzzy Associative Memories," *Fuzzy Expert Systems*, A. Kandel (ed.), Addison-Wesley, in press, December 1986.

Kosko, B., "Fuzzy Entropy and Conditioning," *Information Sciences*, vol. 40, 165 - 174, 1986.

Kosko, B., *Foundations of Fuzzy Estimation Theory*, Ph.D. dissertation, Department of Electrical Engineering, University of California at Irvine, June 1987; Order Number 8801936, University of Microfilms International, 300 N. Zeeb Road, Ann Arbor, Michigan 48106.

Kosko, B., "Hidden Patterns in Combined and Adaptive Knowledge Networks," *International Journal of Approximate Reasoning*, vol. 2, no. 4, 377 - 393, October 1988.

Mamdani, E.H., "Application of Fuzzy Logic to Approximate Reasoning Using Linguistic Synthesis," *IEEE Transactions on Computers*, vol. C-26, no. 12, 1182 - 1191, December 1977.

Taber, W.R., Siegel, M.A., "Estimation of Expert Weights Using Fuzzy Cognitive Maps," *Proceedings of the IEEE 1st International Conference on Neural Networks (ICNN-87)*, vol. II, 319 - 325, June 1987.

Taber, W.R., "Knowledge Processing with Fuzzy Cognitive Maps," *Expert Systems with Applications*, in press, 1990.

Togai, M., Watanabe, H., "Expert System on a Chip: An Engine for Realtime Approximate Reasoning, *IEEE Expert*, vol. 1, no. 3, 1986.

Yamakawa, T., "A Simple Fuzzy Computer Hardware System Employing MIN & MAX Operations," *Proceedings of the Second International Fuzzy Systems Association (IFSA)*, Tokyo, 827 - 830, July 1987.

Yamakawa, T., "Fuzzy Microprocessors–Rule Chip and Defuzzification Chip," *Proceedings of the International Workshop on Fuzzy Systems Applications*, Iizuka-88, Kyushu Institute of Technology, 51 - 52, August 1988.

Zadeh, L.A., "A Computational Approach to Fuzzy Quantifiers in Natural Languages," *Computers and Mathematics*, vol. 9, no. 1, 149 - 184, 1983.

PROBLEMS

1. Use correlation-minimum encoding to construct the FAM matrix M from the fit-vector pair $(A,\ B)$ if $A = (.6\ 1\ .2\ .9)$ and $B = (.8\ .3\ 1)$. Is (A, B) a bidirectional fixed point? Pass $A' = (.2\ .9\ .3\ .2)$ through M and $B' = (.9\ .5\ 1)$ through M^T. Do the recalled fuzzy sets differ from B and A?

2. Repeat Problem 1 using correlation-product encoding.

3. Compute the fuzzy entropy $E(M)$ of M in Problems 1 and 2.

4. If $M = A^T \circ B$ in Problem 1, find a different FAM matrix M' with greater fuzzy entropy, $E(M') > E(M)$, but that still gives perfect recall: $A \circ M' = B$. Find the *maximum entropy fuzzy associative memory* (MEFAM) matrix $\mathbf{M}^*$ such that $A \circ \mathbf{M}^* = B$.

5. Prove: If $M = A^T \circ B$ or $M = A^T B, A \circ M = B$, and $A \subset A'$, then $A' \circ M = B$.

6. Prove: $\max\limits_{1 \le k \le m} \min(a_k, b_k) \le \min(\max\limits_{1 \le k \le m} a_k, \max\limits_{1 \le k \le m} b_k)$.

7. Use truth tables to prove the two-valued propositional tautologies:

   (a) [A $\longrightarrow$ (B OR C)] $\longrightarrow$ [(A $\longrightarrow$ B) OR (A $\longrightarrow$ C)] ,

   (b) [A $\longrightarrow$ (B AND C)] $\longrightarrow$ [(A $\longrightarrow$ B) AND (A $\longrightarrow$ C)] ,

   (c) [(A OR B)$\longrightarrow$ C] $\longrightarrow$ [(A $\longrightarrow$ C) OR (B $\longrightarrow$ C)] ,

   (d) [(A $\longrightarrow$ C) AND (B $\longrightarrow$ C)] $\longrightarrow$ [(A AND B) $\longrightarrow$ C] .

   Is the converse of (c) a tautology? Explain whether this affects BIOFAM inference.

8. BIOFAM inference. Suppose the input spaces X and Y are both $[-10, 10]$, and the output space Z is $[-100, 100]$. Define five trapezoidal fuzzy sets–NL, NS, ZE, PS, PL— on X, Y, and Z. Suppose the underlying (unknown) system transfer function is $z = x^2 - y^2$. State at least five FAM rules that accurately describe the system's

behavior. Use $z = x^2 - y^2$ to generate streams of sample data. Use BIOFAM inference and fuzzy-centroid defuzzification to map input pairs (x, y) to output data z. Plot the BIOFAM outputs and the desired outputs z. What is the arithmetic average of the squared errors $(F(x, y) - x^2 + y^2)^2$? Divide the product space $X \times Y \times Z$ into 125 overlapping FAM cells. Estimate FAM rules from clustered system data (x, y, z). Use these FAM rules to control the system. Evaluate the performance.

Software Problems

The following problems use the accompanying FAM software for controlling an inverted pendulum.

1. Explain why the pendulum stabilizes in the diagonal position if the pendulum bob mass increases to maximum and the motor current decreases slightly. The pendulum stabilizes in the vertical position if you remove which FAM rules?

2. Oscillation results if you remove which FAM rules? The pendulum sticks in a horizontal equilibrium if you remove which FAM rules?

NEURAL NETWORK REPRESENTATION AND LEARNING OF MAPPINGS AND THEIR DERIVATIVES

April 1990

Halbert White, Ph.D.
University of California, San Diego

Collaborative research with:

K. Hornik, M. Stinchcombe and A.R. Gallant

ABSTRACT

We discuss recent theorems proving that artificial neural networks are capable of approximating an arbitrary mapping and its derivatives as accurately as desired. This fact forms the basis for further results establishing the learnability of the desired approximations, using results from non-parametric statistics. These results have potential applications in robotics, chaotic dynamics, control, and sensitivity analysis (physics, chemistry, and engineering). We discuss an example involving learning the transfer function and its derivatives for a chaotic map.

Jordan (1989), "Generic Constraints on Underspecified Target Trajectories," *Proceedings IJCNN*, Washington D.C.:

The Jacobian matrix $\partial z/\partial x$ . . . is the matrix that relates small changes in the controller output to small changes in the task space results and cannot be assumed to be available a priori, or provided by the environment. However, all of the derivatives in the matrix are *forward* derivates. They are easily obtained by differentiation if a forward model is available. The forward model itself must be learned, but this can be achieved directly by system identification. Once the model is accurate over a particular domain, its derivatives provide a learning operator that allows the system to convert errors in task space into errors in articulartory space and thereby change the controller.

UNIVERSAL APPROXIMATION OF AN UNKNOWN MAPPING AND ITS DERIVATIVES USING MULTILAYER FEEDFORWARD NETWORKS *

by

Kurt Hornik, Maxwell Stinchcombe

and

Halbert White

January 1990

* We are indebted to Angelo Melino for pressing us on the issue addressed here and to the referees for numerous helpful suggestions. White's participation was supported by NSF Grant SES-8806990.

ABSTRACT

We give conditions ensuring that multilayer feedforward networks with as few as a single hidden layer and an appropriately smooth hidden layer activation function are capable of arbitrarily accurate approximation to an arbitrary function and its derivatives. In fact, these networks can approximate functions that are not differentiable in the classical sense, but possess only a generalized derivative, as is the case for certain piecewise differentiable functions. The conditions imposed on the hidden layer activation function are relatively mild; the conditions imposed on the domain of the function to be approximated have practical implications. Our approximation results provide a previously missing theoretical justification for the use of multilayer feedforward networks in applications requiring simultaneous approximation of a function and its derivatives.

Relevant Application Areas:

1. Robotics

2. Chaotic Dynamics

3. Control

4. Sensitivity Analysis (Physics, Chemistry, Engineering)

Intuition suggests that networks having smooth hidden layer activation functions ought to have output function derivatives that will approximate the derivatives of an unknown mapping. However, the justification for this intuition is not obvious. Consider the class of single hidden layer feedforward networks having network output functions belonging to the set

$$\Sigma(G) \equiv \{g : \mathbb{R}^r \to \mathbb{R} \mid g(x) = \sum_{j=1}^{q} \beta_j G(\tilde{x}^T \gamma_j);$$

$$x \in \mathbb{R}^r, \beta_j \in \mathbb{R}, \gamma_j \in \mathbb{R}^{r+1}, j = 1,..., q, q \in \mathbb{N}\},$$

where x represents an r vector of network inputs ($r \in \mathbb{N} \equiv \{1, 2, ...\}$), $\tilde{x} \equiv (1, x^T)^T$ (the superscript T denotes transposition), β_j represents hidden to output layer weights and γ_j represents input to hidden layer weights, $j = 1,..., q$, where q is the number of hidden units, and G is a given hidden unit activation function. The first partial derivatives of the network output function are given by

$$\partial g(x) / \partial x_i = \sum_{j=1}^{q} \beta_j \gamma_{ji} DG(\tilde{x}^T \gamma_j), \quad i = 1,..., r,$$

where x_i is the ith component of x, γ_{ji} is the ith component of γ_j, $i = 1,..., r$ (γ_{j0} is the input layer bias to hidden unit j), and DG denotes the first derivative of G.

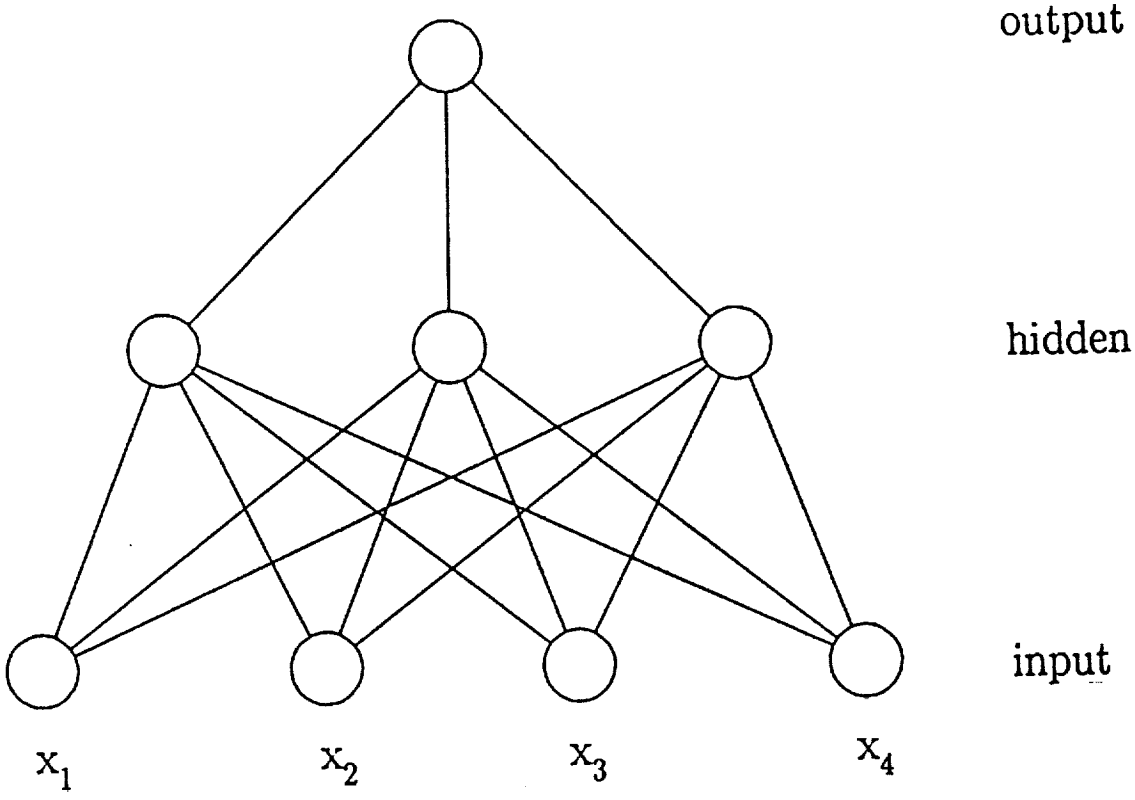


Figure 2

Single Hidden Layer Feedforward Network

Outline:

1. Mathematical Background

2. Approximation Results

3. Learning Results

4. Example: Learning Chaotic Map

1. MATHEMATICAL BACKGROUND

Let U be an open subset of $\mathbb{R}^r$, and let $C(U)$ be the set of all functions continuous on U. Let α be an r-tuple $\alpha = (\alpha_1, \ldots, \alpha_r)^T$ of non-negative integers (a "multi-index"). If x belongs to $\mathbb{R}^r$, let $x^\alpha \equiv x_1^{\alpha_1} \cdot \ldots \cdot x_r^{\alpha_r}$. Denote by D^α the partial derivative

$$\partial^{|\alpha|}/\partial x^\alpha \equiv \partial^{|\alpha|}/(\partial x_1^{\alpha_1}\, \partial x_2^{\alpha_2} \ldots \partial x_r^{\alpha_r})$$

of order $|\alpha| \equiv \alpha_1 + \alpha_2 + \ldots + \alpha_r$. For non-negative integers m, we define $C^m(U) \equiv \{f \in C(U)$: $D^\alpha f \in C(U)$ for all α, $|\alpha| \leq m\}$ and $C^\infty(U) = \cap_{m \geq 1} C^m(U)$. We let D^0 be the identity, so that $C^0(U) = C(U)$. Thus, the functions in $C^m(U)$ have continuous derivatives up to order m on U, while the functions in $C^\infty(U)$ have continuous derivatives on U of every order. We shall be interested in approximating elements of $C^m(U)$ using feedforward networks. When $U \neq \mathbb{R}^r$, the fact that network output functions (elements of $\Sigma(G)$) will belong to $C^m(\mathbb{R}^r)$ necessitates considering their restriction to U, written $g|_U$ for g in $\Sigma(G)$. Recall that $g|_U(x) = g(x)$ for x in U and is not defined for x not in U, thus $g|_U \in C^m(U)$, as desired.)

DEFINITION 2.1: Let U be a subset of $\mathbb{R}^r$, let S be a collection of functions f: $U \to \mathbb{R}$ and let ρ be a metric on S. For any g in $\Sigma(G)$ (recall $g: \mathbb{R}^r \to \mathbb{R}$) define the restriction of g to U, $g_{|U}$ as $g_{|U}(x) = g(x)$ for x in U, $g_{|U}(x)$ unspecified for x not in U.

Suppose that for any f in S and $\varepsilon > 0$ there exists g in $\Sigma(G)$ such that $\rho(f, g_{|U}) < \varepsilon$. Then we say that $\Sigma(G)$ *contains a subset* ρ*-dense in* S. If in addition $g_{|U}$ belongs to S for every g in $\Sigma(G)$, we say that $\Sigma(G)$ is ρ*-dense in* S. □

DEFINITION 2.2: Let $m, l \in \{0\} \cup \mathbb{N}$, $0 \le m \le l$, and $U \subset \mathbb{R}^r$ be given, and let $S \subset C^l(U)$. Suppose that for any f in S, compact $K \subset U$ and $\varepsilon > 0$ there exists g in $\Sigma(G)$ such that $\max_{|\alpha| \le m} \sup_{x \in K} | D^\alpha f(x) - D^\alpha g(x) | < \varepsilon$. Then we say that $\Sigma(G)$ is *m-uniformly dense on compacta* in S. □

When $\Sigma(G)$ is m-uniformly dense on compacta in S, then no matter how we choose an f in S, a compact subset K of U, or the accuracy of approximation $\varepsilon > 0$, we can always find a single hidden layer feedforward network having output function g (in $\Sigma(G)$) with all derivatives of $g_{|U}$ on K up to order m lying within ε of those of f on K. This is a strong and very desirable approximation property.

The space $L_p(U,\mu)$ is the collection of all measurable functions f such that $\|f\|_{p,U,\mu} \equiv [\int_U |f|^p d\mu]^{1/p} < \infty, 1 \le p < \infty$, where the integral is defined in the sense of Lebesgue. When $\mu = \lambda$ we may write either $\int_U f\, d\lambda$ or $\int_U f(x)dx$ to denote the same integral. We measure the distance between two functions f and g belonging to $L_p(U,\mu)$ in terms of the metric $\rho_{p,U,\mu}(f,g) \equiv \|f-g\|_{p,U,\mu}$. Two functions that differ only on sets of μ-measure zero have $\rho_{p,U,\mu}(f,g) = 0$. We shall not distinguish between such functions.

The first Sobolev space we consider is denoted $S_p^m(U,\mu)$, defined as the collection of all functions f in $C^m(U)$ such that $\|D^\alpha f\|_{p,U,\mu} < \infty$ for all $|\alpha| \le m$. We define the Sobolev norm $\|f\|_{m,p,U,\mu} \equiv (\sum_{|\alpha| \le m} \|D^\alpha f\|_{p,U,\mu}^p)^{1/p}$. The Sobolev metric is

$$\rho_{p,\mu}^m(f,g) \equiv \|f-g\|_{m,p,U,\mu} \quad f,g \in S_p^m(U,\mu).$$

Note that $\rho_{p,\mu}^m$ depends implicitly on U, but we suppress this dependence for notational convenience. The Sobolev metric explicitly takes into account distances between derivatives. Two functions in $S_p^m(U,\mu)$ are close in the Sobolev metric $\rho_{p,\mu}^m$ when all derivatives of order $0 \le |\alpha| \le m$ are close in L_p metric.

We also consider the Sobolev spaces

$$W_p^m(U) \equiv \{ f \in L_{1,\mathrm{loc}}(U) \mid \partial^\alpha f \in L_p(U,\lambda), 0 \le \mid \alpha \mid \le m \}.$$

This is the collection of all functions having generalized derivatives belonging to $L_p(U,\lambda)$ of order up to m. Consequently, $W_p^m(U)$ includes $S_p^m(U,\lambda)$, as well as functions that do not have derivatives in the classical sense, such as piecewise differentiable functions.

The norm on $W_p^m(U)$ generalizes that on $S_p^m(U,\lambda)$; we write it as

$$\| f \|_{m,p,U} \equiv \Big(\sum_{\mid \alpha \mid \le m} \| \partial^\alpha f \|_{p,U,\lambda}^p \Big)^{1/p} \quad f \in W_p^m(U).$$

For the metric on $W_p^m(U)$ we suppress the dependence on U and write

$$\rho_p^m(f,g) \equiv \| f - g \|_{m,p,U} \quad f, g \in W_p^m(U).$$

Two functions are close in the Sobolev space $W_p^m(U)$ if all generalized derivatives are close in $L_p(U,\lambda)$ distance.

Our results make fundamental use of one last function space, the space $C^{\infty}_{\downarrow}(\mathbb{R}^r)$ of rapidly decreasing functions in $C^{\infty}(\mathbb{R}^r)$. $C^{\infty}_{\downarrow}(\mathbb{R}^r)$ is defined as the set of all functions in $C^{\infty}(\mathbb{R}^r)$ such that for all multi-indices α and β, $x^{\beta}D^{\alpha}f(x) \to 0$ as $|x| \to \infty$, where $x^{\beta} \equiv x_1^{\beta_1}x_2^{\beta_2}...x_r^{\beta_r}$ and $|x| \equiv \max_{1 \le i \le r} |x_i|$. Note that $C_0^{\infty}(\mathbb{R}^r) \subset C^{\infty}_{\downarrow}(\mathbb{R}^r)$.

Desired results:

1.) $\Sigma(G)$ is m-uniformly dense on compacta in $C^{\infty}_{\downarrow}(\mathbb{R}^r)$, $S_p^m(U, \lambda)$

2.) $\Sigma(G)$ is $\rho^m_{p,\mu}$-dense in $S_p^m(\mathbb{R}^r, \mu)$

3.) $\Sigma(G)$ is ρ_p^m-dense in $W_p^m(U)$

2. APPROXIMATION RESULTS

THEOREM 3.1: Let $G \neq 0$ belong to $S_1^m(\ \mathbb{R},\lambda)$ for some integer $m \geq 0$. Then $\Sigma(G)$ is m-uniformly dense on compacta in $C^{\infty}_{\downarrow}(\ \mathbb{R}^r)$. □

DEFINITION 3.2: Let $l \in \{0\} \cup \mathbb{N}$ be given. G is *l-finite* if $G \in C^l(\ \mathbb{R})$ and $0 < \int | D^l G | \, d\lambda < \infty$. □

LEMMA 3.3: If G is l-finite then for all $0 \leq m \leq l$ there exists $H \in S_1^m(\ \mathbb{R},\lambda), H \neq 0$, such that $\Sigma(H) \subset \Sigma(G)$. □

l-finite activation functions G with $\int D^l G \, d\lambda \neq 0$ have $\int |D^m G| \, d\lambda = \infty$ for all $m < l$, and for $m > l$ all l-finite activation functions G have $\int D^m G \, d\lambda = 0$ (provided $D^m G$ exists).

It is informative to examine cases not satisfying the conditions of the theorems. For example, if $G = \sin$ then $G \in C^{\infty}(\ \mathbb{R})$, but for all l, $\int | D^l G | \, d\lambda = \infty$. If G is a polynomial of degree m then again $G \in C^{\infty}(\ \mathbb{R})$, but for $l \leq m$ we have $\int | D^l G | \, d\lambda = \infty$, although $\int | D^l G | \, d\lambda = 0$ for $l > m$. Consequently, neither trigonometric functions nor polynomials are l-finite.

COROLLARY 3.4: If G is l-finite, then for all $0 \leq m \leq l$, $\Sigma(G)$ is m-uniformly dense on compacta in $C^{\infty}_{\downarrow}(\ \mathbb{R}^r)$. □

COROLLARY 3.5: If G is l-finite, $0 \leq m \leq l$, and U is an open subset of $\mathbb{R}^r$ then $\Sigma(G)$ is m-uniformly dense on compacta in $S_p^m(U,\lambda)$ for $1 \leq p < \infty$. □

COROLLARY 3.6: If G is l-finite and μ is compactly supported, then for all $0 \leq m \leq l$ $\Sigma(G) \subset S_p^m(\ \mathbb{R}^r,\ \mu)$ and $\Sigma(G)$ is $\rho_{p,\mu}^m$-dense in $S_p^m(\ \mathbb{R}^r, \mu)$.

COROLLARY 3.8: If G is l-finite, $0 \leq m \leq l$, U is an open bounded subset of $\mathbb{R}^r$ and $C_0^{\infty}(\ \mathbb{R}^r)$ is ρ_p^m-dense in $W_p^m(U)$ then $\Sigma(G)$ is also ρ_p^m-dense in $W_p^m(U)$.

These results rigorously establish that sufficiently complex multilayer feedforward networks with as few as a single hidden layer are capable of arbitrarily accurate approximation to an unknown mapping and its (generalized) derivatives in a variety of precise senses. The conditions imposed on G are relatively mild; the conditions required of U have practical implications.

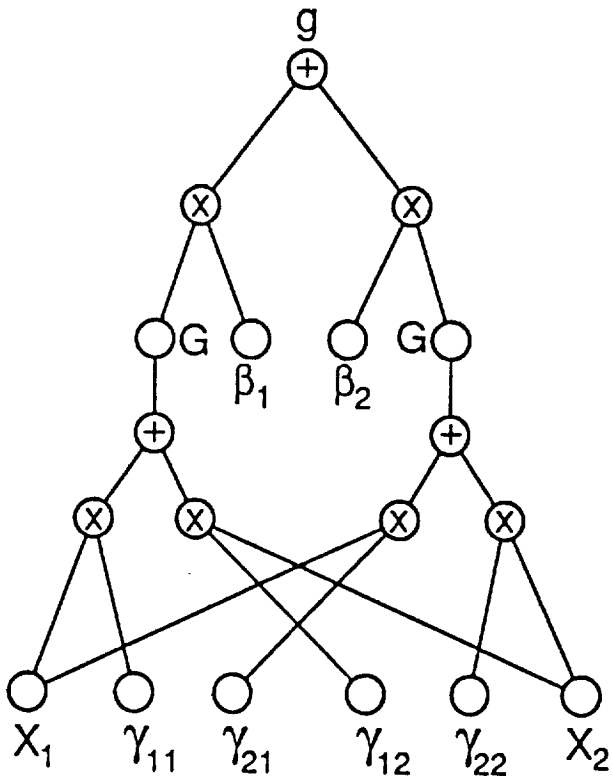


Figure 1. Feedforward Network

○ *input unit* ⊗ *multiplication unit*

G○ *activation unit* ⊕ *addition unit*

Note: biases not shown

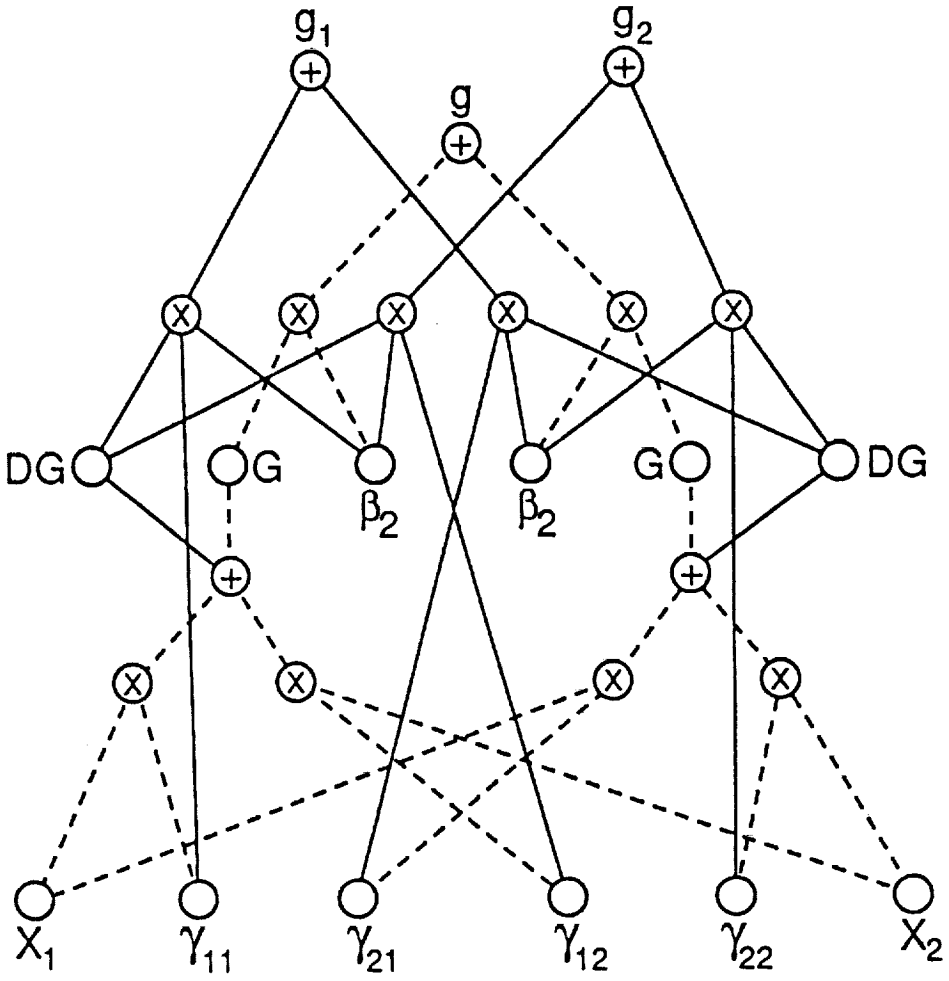


Figure 2. Derivative Network

○ *input unit* ⊗ *multiplication unit*

G○ *activation unit* ⊕ *addition unit*

DG○ *activation derivative unit*

Note: biases not shown

On Learning the Derivatives of an Unknown Mapping with Multilayer Feedforward Networks

by

A. Ronald Gallant
Department of Statistics
North Carolina State University
Raleigh, NC 27696-8203 USA

Halbert White
Department of Economics, D-008
University of California, San Diego
La Jolla, CA 92093

October 1989

ABSTRACT

Recently, multiple input, single output, single hidden layer, feedforward neural networks have been shown to be capable of approximating a nonlinear map and its partial derivatives. Specifically, neural nets have been shown to be dense in various Sobolev spaces (Hornik, Stinchcombe and White, 1989). Building upon this result, we show that a net can be trained so that the map and its derivatives are learned. Specifically, we use a result of Gallant (1987b) to show that least squares and similar estimates are strongly consistent in Sobolev norm provided the number of hidden units and the size of the training set increase together. We illustrate these results by an application to the inverse problem of chaotic dynamics: recovery of a nonlinear map from a time series of iterates. These results extend automatically to nets that embed the single hidden layer, feedforward network as a special case.

3. LEARNING RESULTS

SETUP. We consider a single hidden layer feedforward network having network output function

$$g_K(x, \theta) = \sum_{j=1}^{K} \beta_j G(x^T \gamma_j)$$

where x represents an $r \times 1$ vector of network inputs (including a "bias unit"), β_j represents hidden to output layer weights, γ_j represents input to hidden layer weights, K is the number of hidden units,

$$\theta' = (\beta_1, \gamma_1', \beta_2, \gamma_2', \ldots, \beta_K, \gamma_K'),$$

and G is the hidden unit activation function.

We assume that the network is trained using data $\{y_t, x_t\}$ generated according to

$$y_t = g^*(x_t) + e_t \qquad t = 1, 2, \ldots, n .$$

x_t denotes the observed input and e_t denotes random noise. The number K_n of hidden units employed depends on the size n of the training set. The network is trained by finding $g_{K_n}(x, \hat{\theta})$ that minimizes

$$s_n(\theta) = \frac{1}{n} \sum_{t=1}^{n} [y_t - \sum_{j=1}^{K_n} \beta_j G(x_t^T \gamma_j)]^2 ,$$

subject to the restriction that $g_{K_n}(x, \hat{\theta})$ is a member of the estimation space $\mathcal{G}$.

REGULARITY CONDITIONS:

Input space. The input space X is the closure of a bounded, open subset of $\mathbb{R}^r$.

Parameter space. For some integer m, $0 \leq m < \infty$, some integer p, $1 \leq p < \infty$, and some bound B, $0 < B < \infty$, g^* is a point in the Sobolev space $\mathcal{W}_{m+[r/p]+1, p, X}$ and $\|g^*\|_{m+[r/p]+1, p, X} < B$.

Activation function. The activation function G belongs to $C^m(\mathbb{R})$ and $\int_{-\infty}^{\infty} (d^m/du^m) G(u)\, du < \infty$. See Section 3 of Hornik, Stinchcombe and White (1989).

Estimation space. $g_{K_n}(x, \hat{\theta})$ is restricted to $\mathcal{G} = \{g: \|g\|_{m+[r/p]+1, p, X} \leq B\}$ in the optimization of $s_n(g)$.

Training set. The empirical distribution of $\{x_t\}_{t=1}^n$ converges to a distribution $\mu(x)$ and $\mu(O) > 0$ for every open subset O of X.

Error process. The errors $\{e_t\}$ are independently and identically distributed with common probability law P having $\int_{\mathbb{E}} eP(de) = 0$ and $0 \leq \int_{\mathbb{E}} e^2 P(de) < \infty$. ($\int_{\mathbb{E}} e^2 P(de) = 0$ implies $e_t = 0$ for all t.)

Independence. The probability law P of the errors does not depend on $\{x_t\}_{t=1}^{\infty}$; that is, $P(A)$ can be evaluated without knowledge of $\{x_t\}_{t=1}^{n}$, $\lim_{n\to\infty}(1/n)\sum_{t=1}^{n} x_t$, etc.

THEOREM 1. Under the Regularity Conditions

$$\lim_{n\to\infty} \| g^* - g_{K_n}(\cdot, \hat{\theta}) \|_{m,\infty,x} = 0 \qquad \text{almost surely}$$

provided $\lim_{n\to\infty} K_n = \infty$ almost surely. In particular,

$$\lim_{n\to\infty} \sigma[g_{K_n}(x, \hat{\theta})] = \sigma(g^*) \qquad \text{almost surely}$$

provided σ is continuous with respect to $\| \cdot \|_{m,\infty,x}$. □

4. EXAMPLE: LEARNING CHAOTIC MAP

Our investigation studies the ability of the single hidden layer network

$$g_K(x_{t-5}, \ldots, x_{t-1}) = \sum_{j=1}^{K} \beta_j G(\gamma_{5j} x_{t-5} + \cdots + \gamma_{1j} x_{t-1} + \gamma_{0j})$$

with logistic squasher

$$G(u) = 1/[1 + \exp(-u)]$$

to approximate the derivatives of a discretized variant of the Mackey-Glass equation (Schuster, 1988, p. 120)

$$g(x_{t-5}, x_{t-1}) = x_{t-1} + (10.5)\left[\frac{(0.2)x_{t-5}}{1 + (x_{t-5})^{10}} - (0.1)x_{t-1}\right].$$

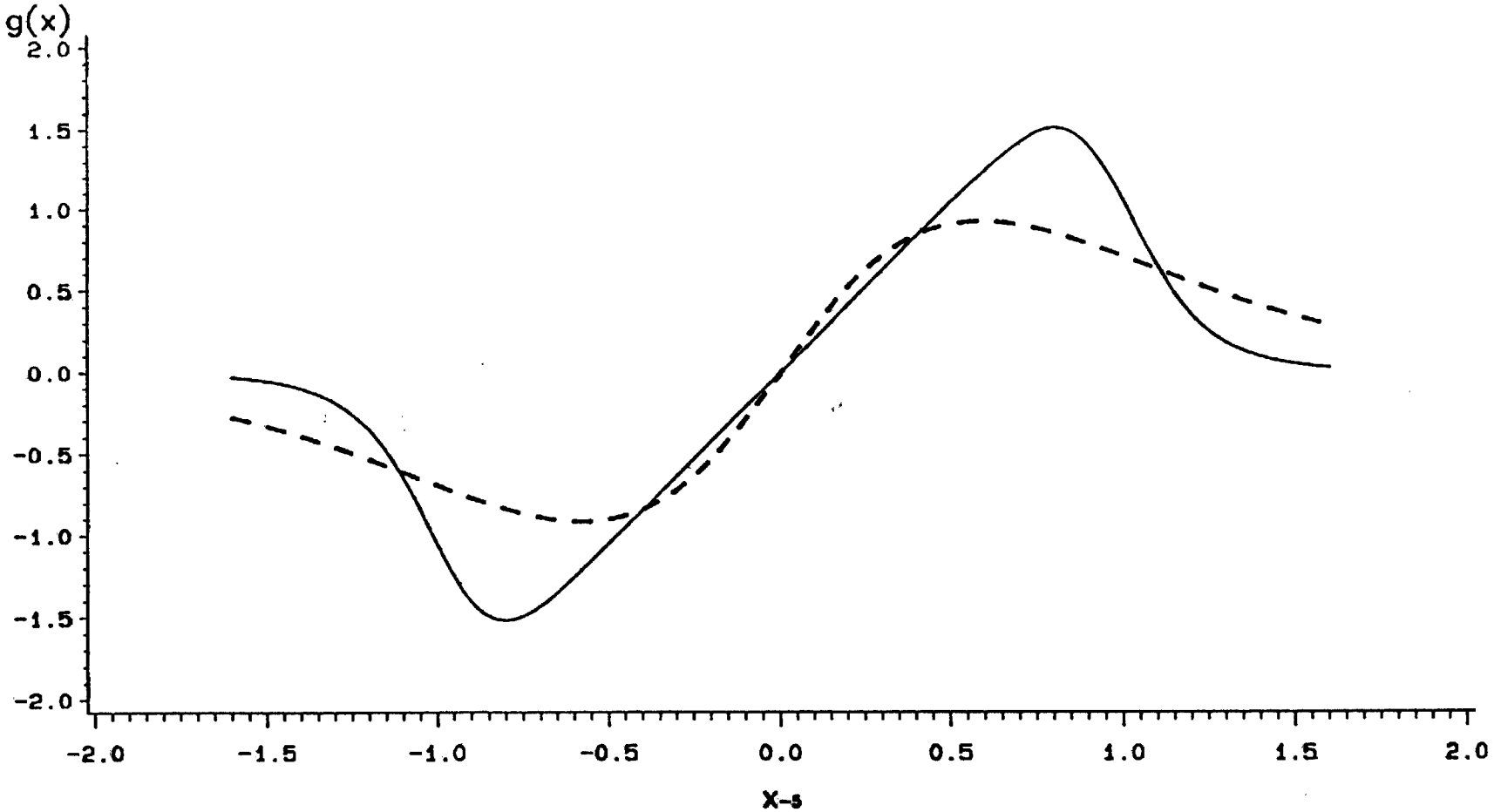
The values of the weights $\hat{\beta}_j$ and $\hat{\gamma}_{ij}$ that minimize

$$s_n(g_K) = \frac{1}{n} \sum_{t=1}^{n} [x_t - g_K(x_{t-5}, \ldots, x_{t-1})]^2$$

were determined using the Gauss-Newton nonlinear least squares algorithm. Our rule relating K to n was of the form $K \propto \log(n)$ because asymptotic theory in a related context (Gallant, 1989) suggests that this is likely to be the relationship that will give stable estimates.

Figure 1. Superimposed nonlinear map and neural net estimate

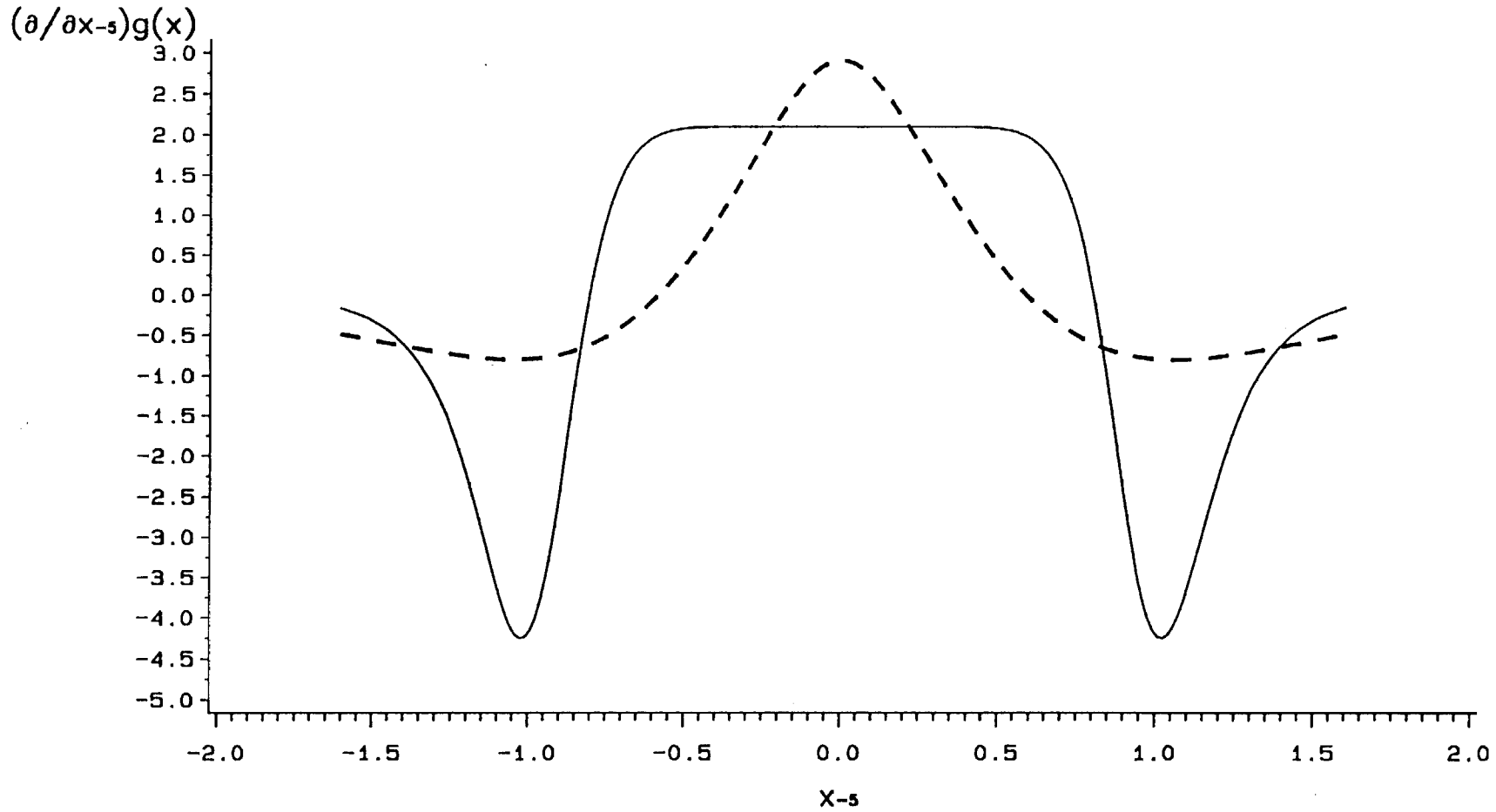
K = 3, n = 500



Note: Estimate is dashed line, x = (x_{-5}, 0, 0, 0, 0)

Figure 2. Superimposed derivative and neural net estimate

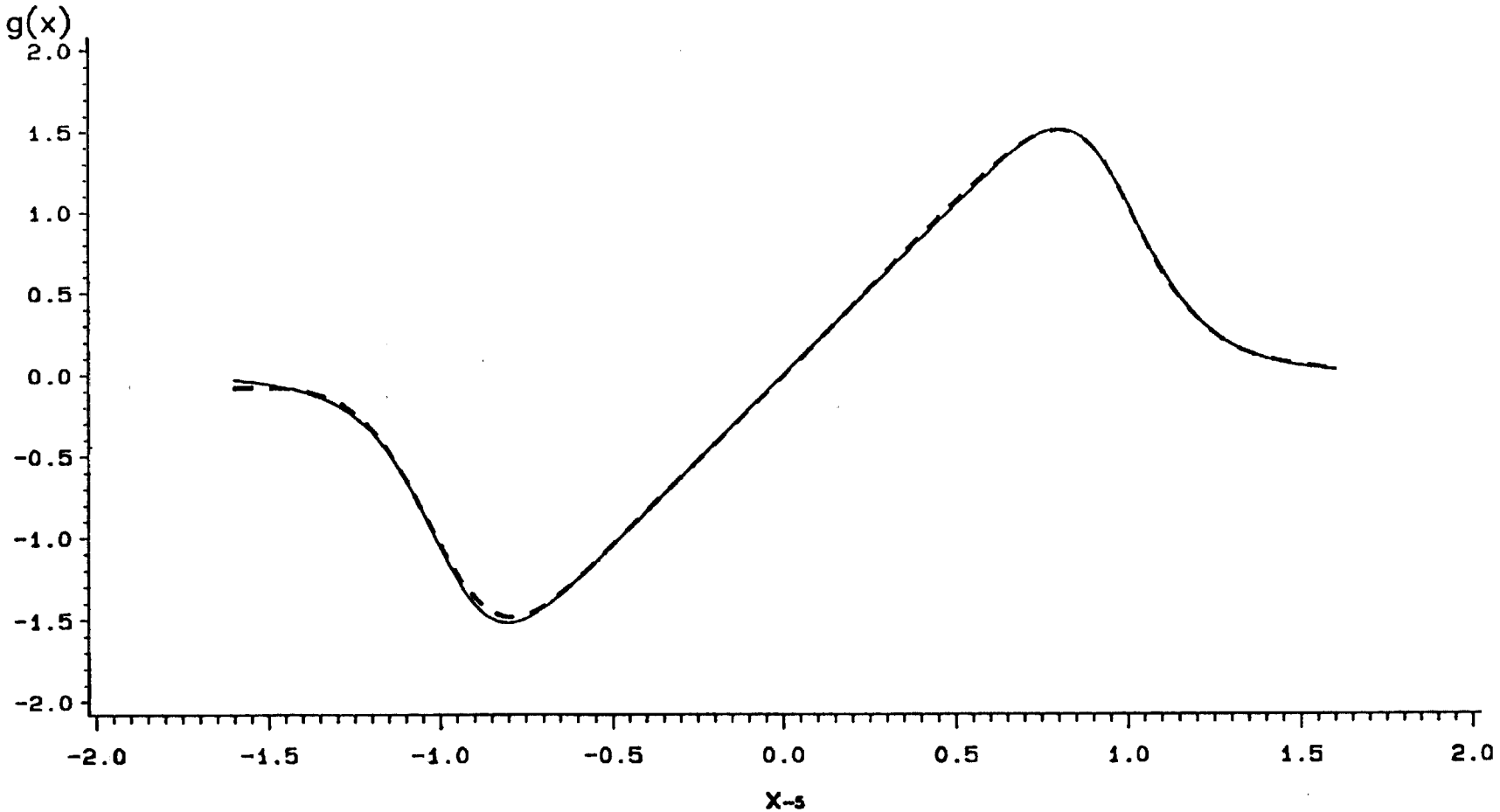
K = 3, n = 500



Note: Estimate is dashed line, x = (x-5, 0, 0, 0, 0)

Figure 3. Superimposed nonlinear map and neural net estimate

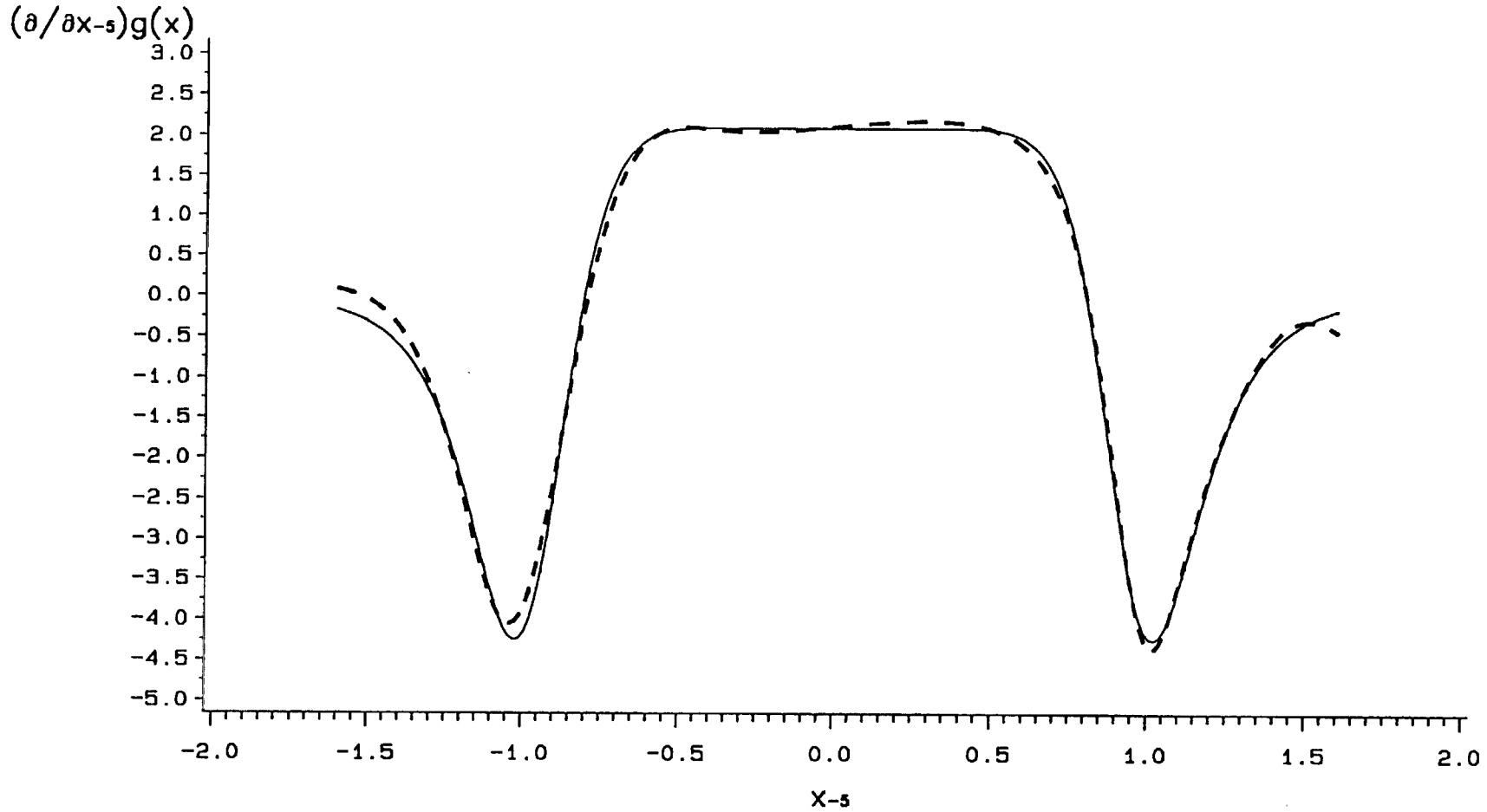
K = 7, n = 2000



Note: Estimate is dashed line, x = (x_{-5}, 0, 0, 0, 0)

Figure 4. Superimposed derivative and neural net estimate

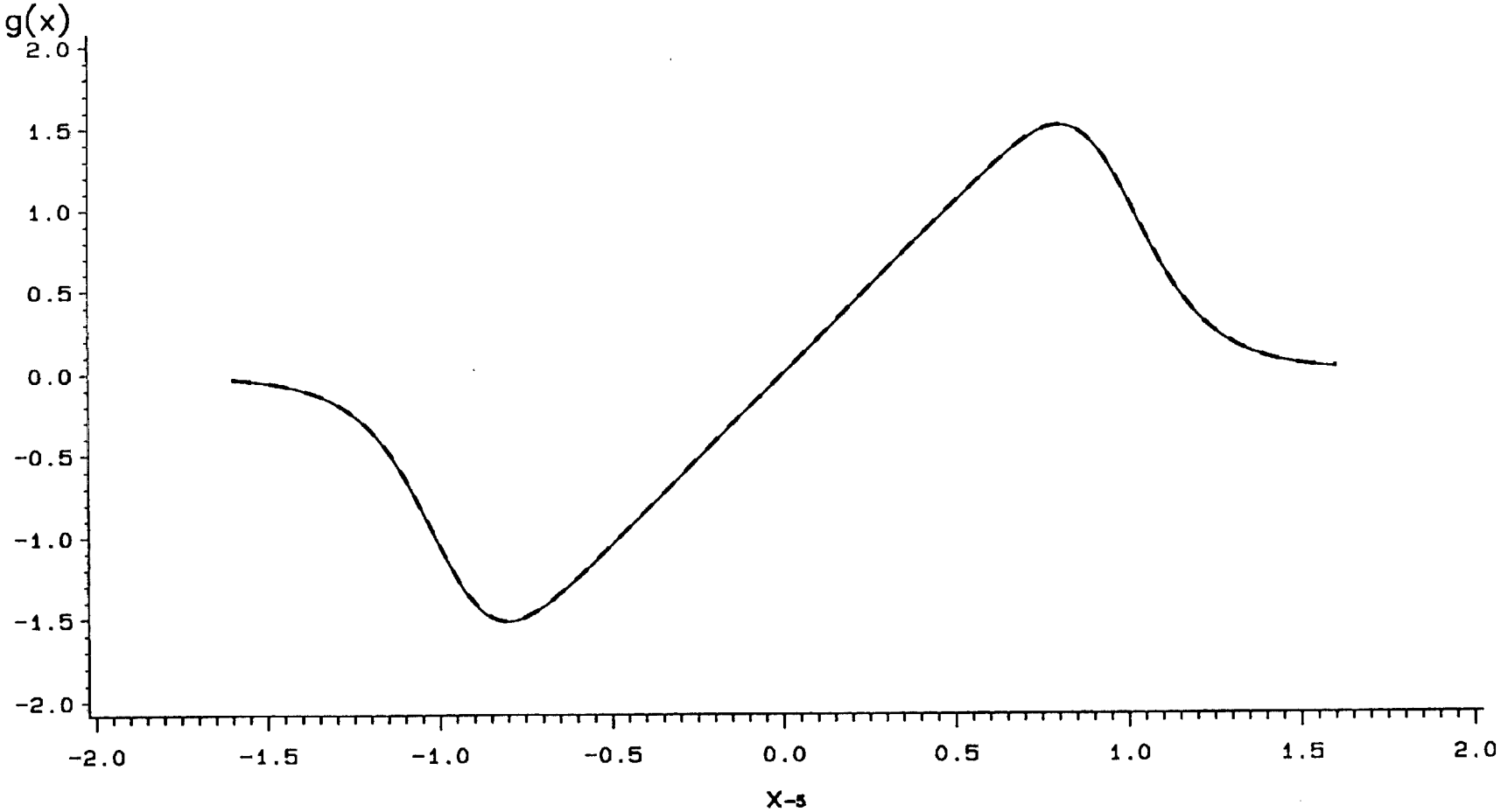
K = 7, n = 2000



Note: Estimate is dashed line, x = (x-5, 0, 0, 0, 0)

Figure 5. Superimposed nonlinear map and neural net estimate

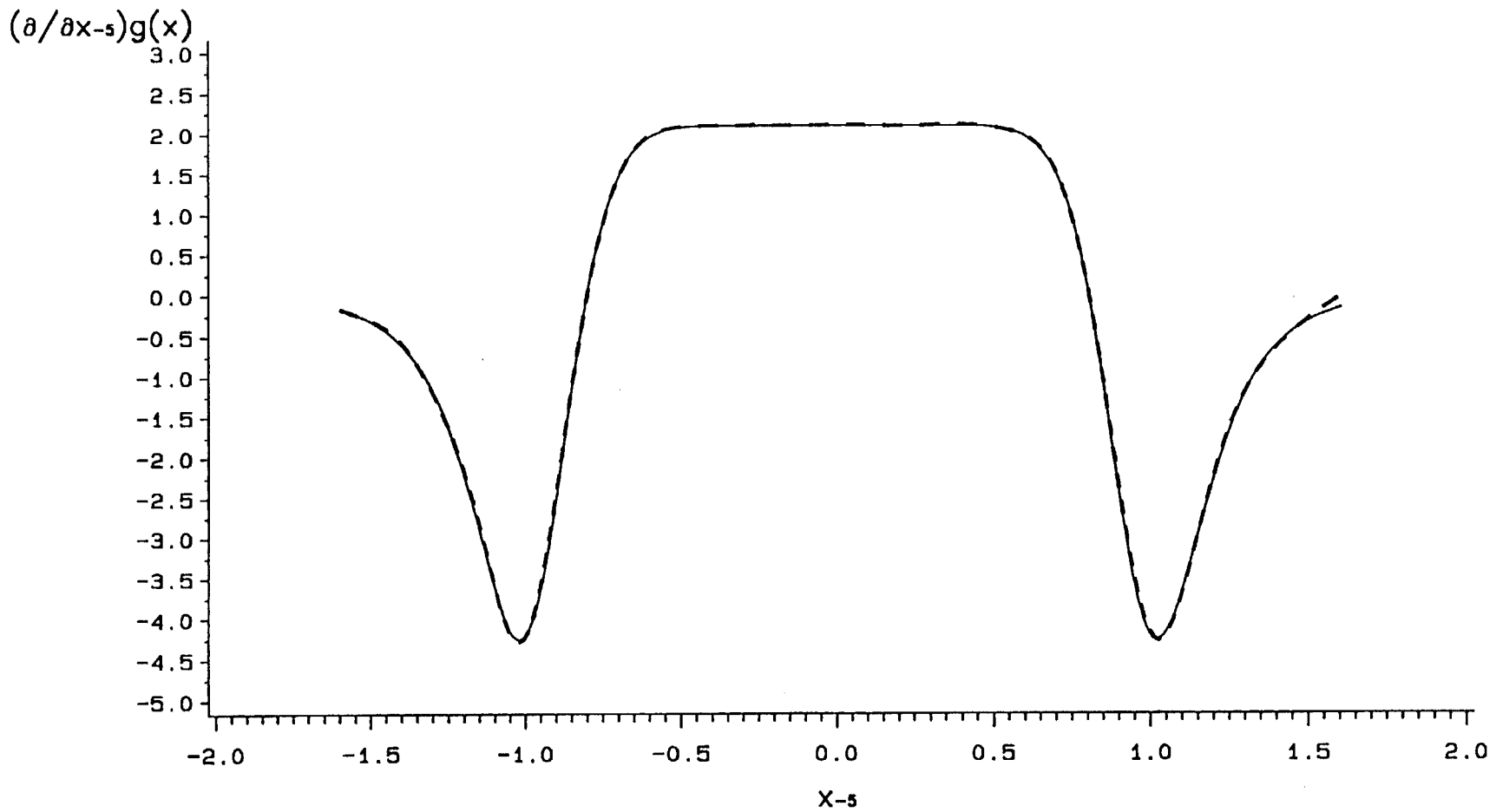
K = 11, n = 8000



Note: Estimate is dashed line, x = (x_{-5}, 0, 0, 0, 0)

Figure 6. Superimposed derivative and neural net estimate

K = 11, n = 8000



Note: Estimate is dashed line, x = (x-5, 0, 0, 0, 0)

Impact of Application of Fuzzy Theory to Industry

(Paper not provided by publication date.)

Time-sweeping Mode Fuzzy Computer -- Forward and Backward Fuzzy Inference Engine

(Paper not provided by publication date.)

N91-21781

The Simplification of Fuzzy Control Algorithm and Hardware Implementation

Z. Q. Wu, P. Z. Wang and H. H. Teh
Institute of Systems Science
National University of Singapore
Heng Mui Keng Terrace,Kent Ridge
Singapore 0511

Abstract

The conventional inference composition algorithm of fuzzy controller is very time and memory consuming. As a result, it is difficult to do real time fuzzy inference and most fuzzy controllers are realized by look-up tables. In this paper we derived a simplified algorithm using the defuzzification mean of maximum. This algorithm takes shorter computation time and needs less memory usage, thus making it possible to compute the fuzzy inference on real time and easy to tune the control rules on line. The responsibility of this algorithm is proved mathematically in this paper.

Fuzzy controller has been highly developed and come to a new stage of hardware implementation. Many fuzzy controllers(or so called fuzzy inference machines) in hardware have been available in the market. The conventional fuzzy inference algorithm on which most fuzzy controller based on is too complicated. Further, its hardware implementation is very expensive and of a large volume, and the inference speed is limited. Reducing its cost and volume and improving its inference speed are very important to this technology. In this paper we also describe a hardware implementation based on the above simplified fuzzy inference algorithm.

1. Fuzzy controller algorithm

Assume that the fuzzy controller has two inputs and a single output as shown in Figure 1,

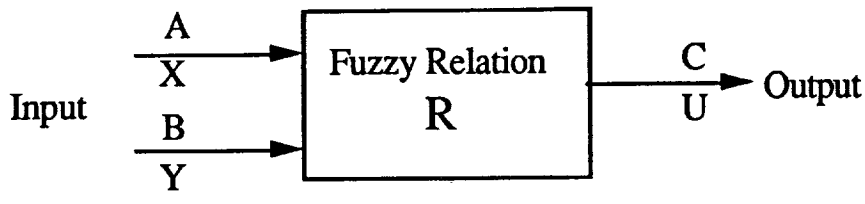


Fig.1 The block graph of fuzzy controller

where A and B are the linguistic variables of the inputs, with universe of discourse X and Y respectively, and C is the linguistic variable of the output, with universe of discourse U. We emphasize here that X and Y are not necessarily continuous on the real line R, but arbitrary subsets of R.

Let the sets of linguistic values concerning with A, B and C respectively be as follows

$$\{A_i\} \in \mathcal{F}(X), \ (i \in I) \tag{1}$$
$$\{B_j\} \in \mathcal{F}(Y), \ (j \in J) \tag{2}$$
$$\{C_k\} \in \mathcal{F}(U), \ (k \in K) \tag{3}$$

where $I=\{1, 2, \cdots, m\}$, $J=\{1,2, \cdots, n\}$, $K=\{1, 2, \cdots, h\}$, and $\mathcal{F}(X)$ represents the fuzzy power set of X.

The fuzzy control rules are described in terms of a group of multi-complexed fuzzy implications as follows:

$$\begin{aligned} &\text{If A is } A_i \text{ and B is } B_j \text{ then C is } C_k, \\ &(i \in I, j \in J, k = \varphi(i, j) \in K) \end{aligned} \tag{4}$$

The above fuzzy implications can be translated into a three-dimensional relation R according to the fuzzy Compositional Rule of Inference(CRI method).

Definition 1.

$$\begin{aligned} &R \equiv \bigcup_{i,j} (A_i \times B_j \times C_k) \\ &R \in \mathcal{F}(X \times Y \times U), \\ &R(x, y, u) = \bigvee_{i,j} (A_i(x) \wedge B_j(y) \wedge C_k(u)). \\ &\qquad (k = \varphi(i, j) \in K) \end{aligned} \tag{5}$$

Suppose that the inputs of the fuzzy controller at a certain instance are fuzzy sets $A^* \in \mathcal{F}(X)$ and $B^* \in \mathcal{F}(Y)$, according to the CRI method, the output of the controller will be the fuzzy set denoted by $C^* \in \mathcal{F}(U)$, i.e

$$C^* = (A^* \times B^*) \circ R$$

$$C^*(u) = \sup_{\substack{x \in X \\ y \in Y}} (A^*(x) \wedge B^*(y) \wedge R(x, y, u))$$

$$= \sup_{\substack{x \in X \\ y \in Y}} ((A^*(x) \wedge B^*(y)) \wedge (\vee_{i,j} (A_i(x) \wedge B_j(y) \wedge C_{\varphi(i,j)}(u))))$$

$$= \sup_{\substack{x \in X \\ y \in Y}} (\vee_{i,j} ((A^*(x) \wedge A_i(x)) \wedge (B^*(y) \wedge B_j(y)) \wedge C_{\varphi(i,j)}(u)))$$

$$= \vee_{i,j} \sup_{x \in X}((A^*(x) \wedge A_i(x)) \wedge \sup_{y \in Y}(B^*(y) \wedge B_j(y)) \wedge C_{\varphi(i,j)}(u) \quad (6)$$

In actual applications the inputs of the controller (i.e the observed values of the controlled process) are some definite real numbers. Suppose in a certain instance the observed value is a pair (x_o, y_o), then the fuzzy sets of inputs A* and B* are as follows,

$$A^*(x) = \begin{cases} 1, & x=x_o \\ 0, & x \neq x_o \end{cases}, \quad B^*(y) = \begin{cases} 1, & y=y_o \\ 0, & y \neq y_o \end{cases} \quad (7)$$

so that

$$\sup_{x \in X}(A^*(x) \wedge A_i(x)) = A_i(x_o) \quad (8)$$

$$\sup_{y \in Y}(B^*(y) \wedge B_j(y)) = B_j(y_o) \quad (9)$$

therefore

$$C^*(u) = \vee_{i,j} (A_i(x_o) \wedge B_j(y_o)) \wedge C_{\varphi(i,j)}(u) \quad (10)$$

$$(i \in I, j \in J, \varphi(i, j) \in K)$$

2. The responsibility of the fuzzy controller

The responsibility of a fuzzy controller has been defined and analyzed in depth by P. Z. Wang and S. P. Lou[3], here we discuss the responsibility of fuzzy controller under a weaker condition.

Definition 2 For a set of linguistic values concerning with A is $\{A_i\}(i \in I) \in \mathcal{F}(X)$, $I=\{1, 2, \cdots, m\}$, where A_i is a normally distributed fuzzy set, there exists m+1 real numbers

$$r_o < r_1 < r_2 < \ldots < r_m ,$$

such that for any given $x \in (r_{i-1}, r_i)$, if $j \neq i$, $A_j(x) < A_i(x)$ (see Figure 2). We called $J_i = (r_{i-1}, r_i) \subseteq X$, the interval of A_i, $i \in I$, and $N^1 = \{ r_i \}$ the net of A, where J_i and N^1 satisfy the following:

$$J_i \cap J_j = \Phi \ , (i \neq j) \tag{11}$$

$$\bigcup_{i=1}^{m} J_i = X - \{ r_i \} \tag{12}$$

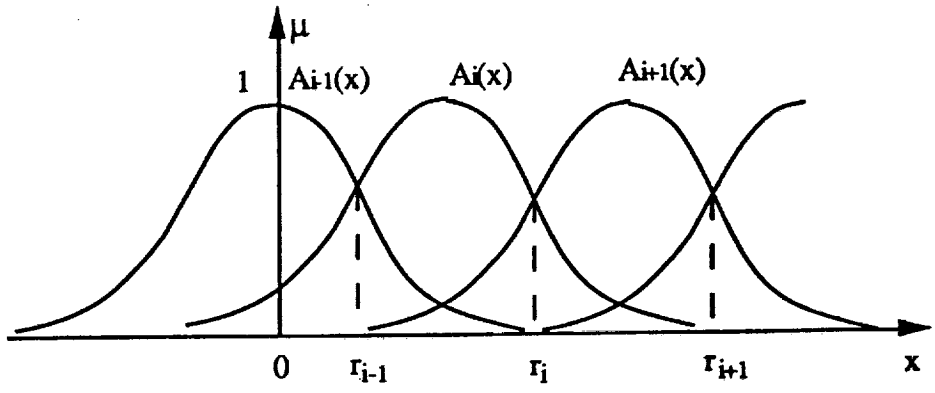


Figure 2 Membership functions of fuzzy sets A_i

For $\{A_i\} \in \mathcal{F}(X)$, $\{B_j\} \in \mathcal{F}(Y)$, we have (13)

$$(A_i \times B_j)(x, y) \equiv A_i(x) \wedge B_j(y)$$

$$\forall i \in I, \forall j \in J, I=\{1, 2, \cdots, m\}, J=\{1,2, \cdots, n\}. \tag{14}$$

If there exist nets $N^1= \{r'_i\}$, $N^2 = \{r''_j\}$ and intervals $\{J'_i\}$, $\{J''_j\}$ for A and B respectively, then

$$(x, y) \in J_{st} \longrightarrow A_s(x) \wedge B_t(y) > A_i(x) \wedge B_j(y) \quad (15)$$
$$((s, t) \neq (i, j))$$

where $J_{ij} \equiv J'_i \times J''_j$, is called the interval of $(A_i \times B_j)$.
In fact,

$$(x, y) \in J_{st} \longrightarrow x \in J'_s, y \in J''_t \longrightarrow A_s(x) > A_i(x), B_t(y) > B_j(y)$$
$$(s, t) \neq (i, j)$$
$$\longrightarrow \min(A_s(x), B_t(y)) > \min(A_i(x), B_j(y))$$
$$\longrightarrow A_s(x) \wedge B_t(y) > A_i(x) \wedge B_j(y) \quad (16)$$

we define the net of $A \times B$ as

$$N \equiv \{(x, y) \mid x = r'_i, y = r''_j\} \quad (17)$$

According to the definition of responsibility of fuzzy controller given in [3], we state the following definition with slight changes.

Definition 3 A Fuzzy controller is said to be responsive if there exists an interval $L \subseteq (-\infty, +\infty)$ such that

$$L = \{u \mid C^*(u) = \operatorname{hgt}_{u \in U} C^*(u)\}. \quad (18)$$

where hgt$C^*(u)$ is the height of fuzzy set $C^*(u)$ and L is the responsive interval.

If a fuzzy controller is responsive, the output of the controller, according to the defuzzification mean of maximum, is

$$u_o = M(L) \quad (u_o \in U), \quad (19)$$

where M(L) means the mid-point of L.

Theorem 1. A given fuzzy controller is responsive as long as there exists a Net N of $A \times B$ such that the intersection of N and the universe of discourse $(X \times Y)$ is empty, i.e

$$N \cap \underline{X} = \Phi, (\underline{X} = X \times Y) \tag{20}$$

Proof: Assume that N is the Net of A × B, which satisfies formula (20), i.e

$$\underline{X} = \underline{X} - N \tag{21}$$

from formula (21), we derive that

$$\bigvee_{i,j} J_{ij} = \underline{X} = (X \times Y) \tag{22}$$

so for any definite $(x_o, y_o) \in (X \times Y)$, there exist s, t, such that $(x_o, y_o) \in J_{st}$.

From formula (16), for any $s,i \in I, t,j \in J$, if $(s, t) \neq (i, j)$, then

$$A_s(x_o) \wedge B_t(y_o) > A_i(x_o) \wedge B_j(y_o) \tag{23}$$

According to formula (10), the response of the fuzzy controller is as follows

$$C^*(u) = \bigvee_{i,j} (A_i(x_o)) \wedge B_j(y_o) \wedge C_{\varphi(i, j)}(u) \tag{24}$$

By formula (23), it is obvious that

$$M(C^*) = M(f) \tag{25}$$

Where

$$M(C^*) = \{u \mid C^*(u) = hgtC^*(u)\} \tag{26}$$

$$M(f) = \{u \mid f(u) = hgt\ f(u)\} \tag{27}$$

$$f(u) = (A_s(x_o)) \wedge Bt(y_o) \wedge C_{\varphi(s, t)}(u) \tag{28}$$

As it is known that $C_{\varphi(i, j)} \in \{C_k\} \neq \Phi$ is a distributed fuzzy set whose kernel is

$$Ker(C_k) = \{u \mid C_k(u) = 1\} \neq \Phi \tag{29}$$

obviously, there exists an interval $L \subseteq (-\infty, +\infty)$ such that

$$L = \{u \mid C_{\varphi(s,\,t)}(u) \geq (A_S(x_o)) \wedge B_t(y_o))\}$$
$$= \{u \mid f(u) = (A_S(x_o)) \wedge Bt(y_o))\}$$
$$= \{u \mid C^*(u) = hgtC^*(u)\} \tag{30}$$

therefore the fuzzy controller is responsive and $u_o = M(L)$.

3. The simplification of fuzzy controller algorithm

In the right-hand side of formula (10), there are $I \times J$ terms of union operations. The ordinary algorithm does this calculations term by term and is very time consuming. We know from Theorem 1 that when a fuzzy controller is responsive and the defuzzification mean of maximum is used, we only need to calculate the interval L, then the mid-point of L will be the desired output of the fuzzy controller. Thus for all observed value (x_o, y_o), we only have to calculate f(u), only one of the terms in the formula (10). This will simplify the computation algorithm to a great extent.

Let

$$A_\Sigma = \bigvee_{i \in I} A_i \tag{31}$$

$$A_\Sigma(x) = \bigvee_{i \in I} A_i(x) \tag{32}$$

$$B_\Sigma = \bigvee_{j \in J} B_j \tag{33}$$

$$B_\Sigma(y) = \bigvee_{j \in J} B_j(y) \tag{34}$$

where $x \in X\text{-}\{r'_i\}$, $y \in Y\text{-}\{r''_j\}$. The membership functions of $A_\Sigma(x)$ and $B_\Sigma(y)$ are shown in Figure 3-a.

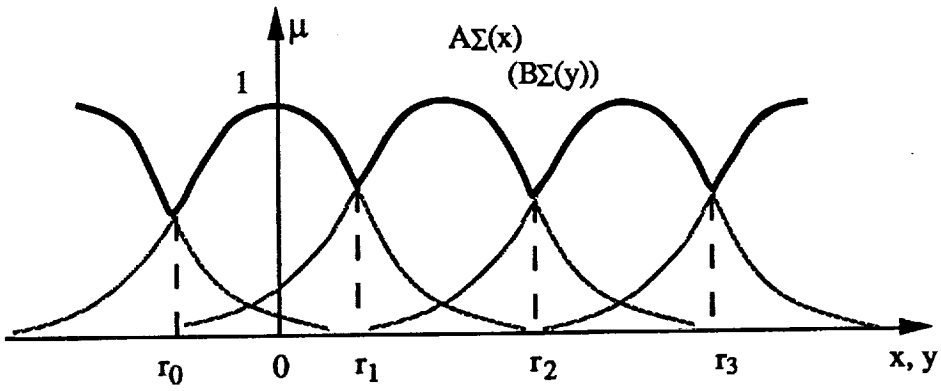


Figure 3-a The membership functions of $A_\Sigma(x)$ and $B_\Sigma(y)$

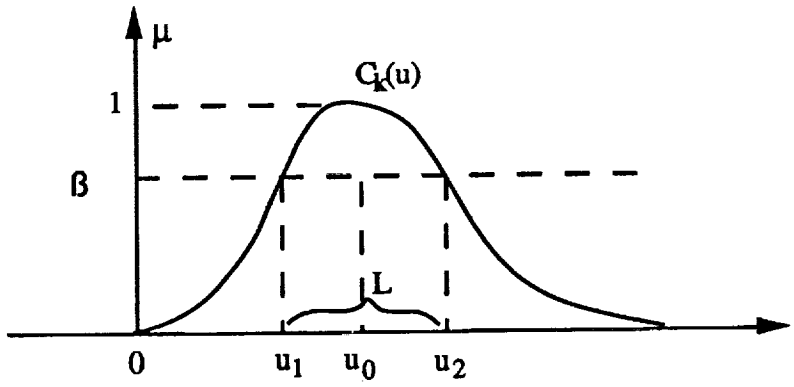


Figure 3-b The function of $\rho_k(\beta)$

Clearly, when $(x, y) \in J_{st}$, i.e $x \in \{r'_{s-1}, r'_s\}$, $y \in \{r''_{t-1}, r''_t\}$

$$A_\Sigma(x) = A_s(x) > A_i(x), \tag{35}$$
$$B_\Sigma(y) = B_t(y) > B_j(y), \tag{36}$$
$$A_\Sigma(x) \wedge B_\Sigma(y) = A_s(x) \wedge B_t(y) > A_i(x) \wedge B_j(y) \tag{37}$$
$$\forall\, s,i \in I,\ t,j \in J,\ (s, t) \neq (i, j)$$

Define the separating functions $\varphi_1(x)$, $\varphi_2(y)$ respectively as follows,

$$\varphi_1(x) = i, \quad x \in \{r'_{i-1}, r'_i\}, \tag{38}$$
$$\varphi_2(y) = j, \quad y \in \{r''_{j-1}, r''_j\}. \tag{39}$$

For $C \in \{C_k\}$, $k \in K$, we define the following function

$$\rho_k(\beta) = M(L) = M(C_{k\beta}),\ \beta \in [0, 1] \tag{40}$$

where $L \subseteq (-\infty, +\infty)$,

$$L \cap U = \{u \mid C_k \geq \beta\} \tag{41}$$

where $C_{k\beta}$ is the β- cuted set of C_k and $M(\cdot)$ represents the mid-point of $(\cdot)$ as shown in Figure 3-b. Since $C_k(k \in K)$ is normally distributed set, $\rho_k(\beta)$ is a continuous single-valued function of β, $\forall\ \beta \in [0, 1]$

So far as the functions $A_\Sigma(x)$, $B_\Sigma(y)$, $\varphi_1(x)$, $\varphi_2(y)$ and $\rho_k(\beta)$ are defined, we can derive the following simplified algorithm for the responsive fuzzy controllers:

1) Given the inputs (x_o, y_o) of the fuzzy controller, calculate

$$a = A_\Sigma(x_o),\ b = B_\Sigma(y_o), \tag{42}$$

$$s = \varphi_1(x_o),\ t = \varphi_2(y_o), \tag{43}$$

$$\beta = a \wedge b = \min(a,b). \tag{44}$$

2) Calculate

$$k = \varphi(s, t),\ k \in K \tag{45}$$

where the φ is determined by the given control rules.

3) Finally, the output of the fuzzy controller can be obtained from

$$u_o = \rho_k(\beta) \tag{46}$$

Obviously,
$$L \cap U = \{u \mid C_k \geq \beta\}$$
$$= \{u \mid C_{\varphi(s, t)} \geq (A_s(x_o)) \wedge B_t(y_o))\} \tag{47}$$

$$u_o = \rho_k(\beta) = M(L)$$

It can be seen that the result is exactly the same as that in formula (30).

The conventional fuzzy controller algorithm is very time consuming and needs

large memory space so that it is hardly possible to implement the fuzzy composition inference on line in a control system. In many applications, fuzzy controllers used look up tables instead of real time inference. Not only it is impossible to tune the fuzzy control rules on line, it takes a great amount of computation time to calculate the fuzzy controller look-up table. The simplified algorithm proposed above reduces the computation greatly and its calculating time is nearly the same as that taken by the conventional PID control algorithm. This makes it possible to do real time fuzzy inference in the controller, allowing the tuning of control rules on line. If the algorithm is used to calculate the fuzzy control look-up table, it takes less than one minute. Since we only need to store 5 functions, namely $A_{\Sigma}(x)$, $B_{\Sigma}(y)$, $\varphi_1(x)$, $\varphi_2(y)$ and $\rho_k(\beta)$ instead of all the Ai(x), Bj(y), and Ck(u), a total of I+J+K functions.

4. Hardware Implementation

Fuzzy controller has been highly developed and come to a new stage of hardware implementation. Many fuzzy controllers(or so called fuzzy inference machines) in hardware are available in the market[4][5]. The conventional fuzzy inference algorithm on which most fuzzy controllers are based on is too complicated. Further, its hardware implementation is very expensive and of a large volume, and the inference speed is limited. Reducing its cost and volume and improving its inference speed are very important to this technology.

As can be seen from the last section that with the proposed algorithm, the calculation is much simpler as there is no computation of fuzzy sets and most of the calculations involve only function operations and comparative operations. Therefore, this fuzzy control algorithm is very easy to implement in hardware. The main issue in a hardware design is to construct some function generators generating $A_{\Sigma}(x)$, $B_{\Sigma}(y)$, $\varphi_1(x)$, $\varphi_2(y)$ and $\rho_k(\beta)$, while the complicated fuzzy set operation which is difficult to turn into hardware counterparts is avoided.

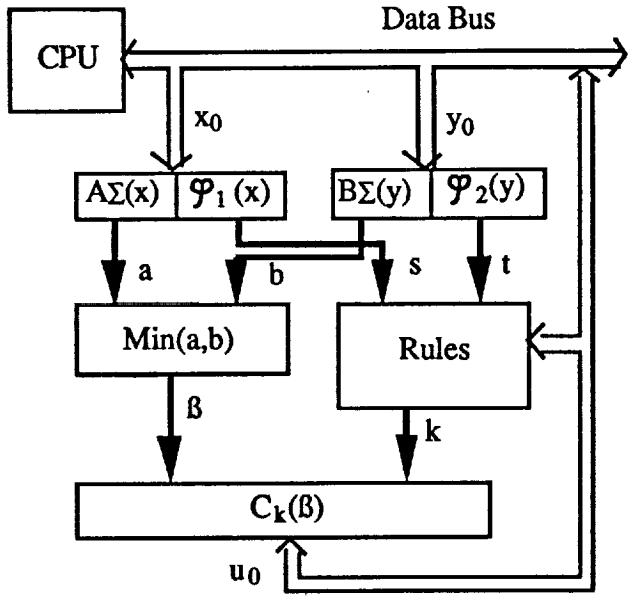


Figure 4 Block diagram of the fuzzy controller board

We have designed a fuzzy controller board for Personal Computers(PC) based on the above algorithm. The principle of the fuzzy controller board is illustrated in Figure 4. The board is composed of some function generators to generate $A_{\Sigma}(x)$, $B_{\Sigma}(y)$, $\varphi_1(x)$, $\varphi_2(y)$ and $\rho_k(\beta)$, a comparator to do the operation of Min(a, b) and a rule base to store the control rules. Each part is constructed with digital IC. The detailed design of the hardware will be presented in depth in our future papers.

The controller board is connected to the CPU through the data bus of the PC. The generators of $A_{\Sigma}(x)$, $B_{\Sigma}(y)$, $\varphi_1(x)$, $\varphi_2(y)$ and $\rho_k(\beta)$ and the control rules can be programmed conveniently. Using this board with its software environment on a personal computer, it is very flexible to construct a fuzzy control system for an industrial process in which large number of data needed to be processed. This is the reason why we design a fuzzy controller board instead of an independent fuzzy controller machine which is unable to process data and information.

Due to its fuzzy inference function and ability of data processing, the fuzzy control system can be applied not only to the control system but also to many other areas such as expert systems, pattern recognition and decision making where the fuzzy inference method may be employed.

Reference

[1] G. Q. Chen, Analysis into the fuzzy control algorithm, Information and Control, No.5(1980)

[2] M. Mizumoto, Fuzzy reasoning with "if...then else", Applied sys. and cybern., 1, 2927-2932(1980)

[3] P. Z. Wang and S. P. Lou, The responsibility of a fuzzy controller, IFAC 8th Triennial World Cong., Kyoto, Japan, August 1981.

[4] T. Yamakawa, Fuzzy Controller Hardware System, Preprints of 2nd IFSA Congress, Tokyo, July 1987.

[5] T. Yamakawa, A Simple Fuzzy Computer Hardware System Employing Min & Max operation--A Challenge to 6th Generation Computer, Preprints of 2nd IFSA Congress, Tokyo, July 1987.

N91-21782

Radar Signal Categorization using a Neural Network

James A. Anderson

Department of Cognitive and Linguistic Sciences

Box 1978

Brown University, Providence, RI 02912

and

Michael T. Gately, P. Andrew Penz, and Dean R. Collins

Central Research Laboratories, Texas Instruments

Dallas, Texas 75265

Accepted for Publication, IEEE Proceedings
To appear, August, 1990

This research was initially supported by Texas Instruments, the Office of Naval Research (Contract N00014-86-K-0600 to J.A.) and the National Science Foundation (Grant BNS-85-18675 to J.A.). Currently, this research is supported by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division [Contract F33615-87-C1454].

Radar Signal Categorization Using a Neural Network

Abstract

Neural networks were used to analyze a complex simulated radar environment which contains noisy radar pulses generated by many different emitters. The neural network used is an energy minimizing network (the BSB model) which forms energy minima -- attractors in the network dynamical system -- based on learned input data. The system first determines how many emitters are present (the deinterleaving problem). Pulses from individual simulated emitters give rise to separate stable attractors in the network. Once individual emitters are characterized, it is possible to make tentative identifications of them based on their observed parameters. As a test of this idea, a neural network was used to form a small data base that potentially could make emitter identifications.

We have used neural networks to cluster, characterize and identify radar signals from different emitters. The approach assumes the ability to monitor a region of the microwave spectrum and to detect and measure properties of received radar pulses. The microwave environment is assumed to be complex, so there are pulses from a number of different emitters present, and pulses from the same emitter are noisy or their properties are not measured with great accuracy.

For several practical applications, it is important to be able to tell quickly, first, how many emitters are present and, second, what their properties are. In other words time average prototypes must be derived from time dependent data without a tutor. Finally the system must tentatively identify the prototypes as members of previously seen classes of emitter.

Stages of Processing. We accomplish this task in several stages. Figure 1 shows a block diagram of the resulting system, which contains several neural networks. The system as a whole is referred to as the Adaptive Network Sensor Processor (ANSP).

Figure 1 About Here

In the block diagram given in Figure 1, the first block is a feature extractor. We start by assuming a microwave radar receiver of some sophistication at the input to the system. This receiver is capable of processing each pulse into feature values, i.e. azimuth, elevation, signal to noise ratio (normalized intensity), frequency, and pulse width. This data is then listed in a pulse buffer and tagged with time of arrival of the pulse. In a complex radar environment, hundreds or thousands of pulses can arrive in fractions of seconds, so there is no lack of data. The problem, as in many data rich environments, is making sense of it.

The second block in Figure 1 is the deinterleaver which clusters incoming radar pulses into groups, each group formed by pulses from a single emitter. A number of pulses are observed, and a neural network computes, off line, how many emitters are present, based on the sample, and estimates their properties. That is, it solves the so-called deinterleaving problem by identifying pulses as being produced by a particular emitter. This block also produces and passes forward measures of the each cluster's azimuth, elevation, SNR, frequency and pulse width.

The third block, the pulse pattern extractor, uses the deinterleaved information to compute the pulse repetition pattern of an emitter by using the times of arrival for the pulses that are contained in a given cluster. This information will be used for emitter classification.

The fourth block, the tracker, acts as a long term memory for the clusters found in the second block, storing the average azimuth, elevation, SNR, frequency, and pulse width. Since the diagram in Figure 1 is organized via initial computational functionality, the tracking module follows the deinterleaver so as to store its outputs. In an operationally organized diagram, the tracker is the first block to receive pulse data from the feature extractor. It must identify most of the pulses in real time as previously learned by the deinterleaver module and only pass a small number of unknown pulses back to the deinterleaver module for further learning. The tracker also updates the cluster averages. Their properties can change with time because of emitter or receiver motion, for example.

The fourth and fifth blocks, the tracker and the classifier operate as a unit to classify the observed emitters, based on information stored in a data base of emitter types. Intrinsic emitter properties stored in these blocks are frequency, pulse width and pulse repetition pattern.

The most important question for the ANSP to answer is what the emitters might be and what can they do. That is, "who is looking at me, should I be concerned, and should I (or can I) do something about it?"

Emitter Clustering. Most of the initial theoretical and simulation effort in this project has been focused on the deinterleaving problem. This is because the ANSP is being asked to form a conception of the emitter environment from the data itself. A teacher does not exist for most interesting situations.

In the simplest case, each emitter emits with constant properties, i.e. no noise is present. Then, determining how many emitters were present would be trivial: simply count the number of unique pulses via a look up table. Unfortunately, data is often moderately noisy because of receiver, environmental and emitter variability, and, sometimes, because of the frequent change of one or another emitter property at the emitter. Therefore, simple identity checks will not work. It is these later cases which this paper will address.

Many neural networks are supervised algorithms, that is, they are trained by seeing correctly classified examples of training data and, when new data is presented will identify it according to their past experience. Emitter identification does not fall into this category because the correct answers are not known ahead of time. That, after all, is the purpose of this system. The basic problem of a self-organizing clustering system has many historical precedents in cognitive science. For example, William James, in a quotation well known to developmental psychologists, wrote around 1890,

> ..the numerous inpouring currents of the baby bring to his consciousness ... one big blooming buzzing Confusion. That Confusion is the baby's universe; and the universe of all of us is still to a great extent such a Confusion, potentially resolvable, and demanding to be resolved, but not yet actually resolved into parts.
>
> William James (1890, p.29)

We now know that the new born baby is a very competent organism, and the outlines of adult perceptual preprocessing are already in place. The baby is designed to hear human speech in the appropriate way and to see a world like ours: that is, a baby is tuned to the environment in which he will live. The same is true of the ANSP, which must process pulses which will have feature values that fall within certain parameter ranges. That is, an effective feature analysis has been done for us by the receiver designer, and we do not have to organize a system from zero. This means that we can use a less general approach than we might have to in a less constrained problem. The result of both evolution and good engineering design is to build so much structure into the system that a problem, very difficult in its general form, becomes quite tractable.

At this point, neural networks are familiar to many. Introductions are available, for example, McClelland and Rumelhart, 1986; Rumelhart and McClelland, 1986; Hinton and Anderson, 1989; Anderson and Rosenfeld, 1988.

The Linear Associator. Let us begin our discussion of the network we shall use for the radar problem with the 'outer product' associator, also called the 'linear associator,' as a starting point. (Kohonen, 1972, 1977, 1984; Anderson, 1972). We assume a single computing unit, a simple model neuron, acts as a linear summer of its inputs. There are many such computing units. The set of activities of a group of units is the system state vector. Our notation has matrices represented by capital letters (A), vectors by lower case letters (f,g), and the elements of vectors as f(i) or g(j). A vector from a set of vectors is subscripted, for example, f_1, f_2 ... .

The ith unit in a set of units will display activity g(i) when a pattern f(j) is presented to its inputs, according to the rule,

$$g(i) = \sum_j A(i,j)\, f(j).$$

where A(i,j) are the connections between the ith unit in an output set of units and the jth unit in an input set. We can then can write the output pattern, g, as the matrix multiplication

$$g = A\, f.$$

During learning, the connection strengths are modified according to a generalized Hebb rule, that is, the change in an element of A, $\delta A(i,j)$, is given by

$$\delta A(i,j) \propto f_k(j)\, g_k(i),$$

where f_k and g_k are vectors associated with the kth learning example.
Then we can write the matrix A as a sum of outer products,

$$A = \eta \sum_{k=1}^{n} g_k f_k^T$$

where η is a learning constant.

Prototype Formation The linear model forms prototypes as part of the storage process, a property we will draw on. Suppose a category contains many similar items associated with the same response. Consider a set of correlated vectors, $\{f_k\}$, with mean p.

$$f_k = p + d_k .$$

The final connectivity matrix will be

$$A = \eta \sum_{k=1}^{n} g_k f_k^T$$

$$= \eta g (n p^T + \sum_{k=1}^{n} d_k^T)$$

If the sum of the d_k is small, the connectivity matrix is approximated by

$$A = \eta n g p^T .$$

The system behaves as if it had repeatedly learned only one pattern, p, and responds best to it, even though p, in fact, may never have been learned.

Concept forming systems. Knapp and Anderson (1984) applied this model directly to the formation of simple psychological 'concepts' formed of nine randomly placed dots. A 'concept' in cognitive science describes the common and important situation where a number of different objects are classed together by some rule or similarity relationship. Much of the power of language, for example, arises from the ability to see that physically different objects are really 'the same' and can be named and responded to in a similar fashion, for example, tables or lions. A great deal of experimentation and theory in cognitive science concerns itself with concept formation and use.

There are two related but distinct ways of explaining simple concepts in neural network models. First, there are prototype forming systems, which often involve taking a kind of average during the act of storage, and, second, there are models which explain concepts as related to attractors in a dynamical system. In the radar ANSP system to be described we use both ideas: we want to construct a system where

the average of a category becomes the attractor in a dynamical system, and an attractor and its surrounding basin represent an individual emitter. (For a further discussion of concept formation in simple neural networks, see Knapp and Anderson, 1984; Anderson, 1983, and Anderson and Murphy, 1986).

Error Correction. By using an error correcting technique, the Widrow-Hoff procedure, we can force the simple associative system to give us more accurate associations. Let us assume we are working with an autoassociative system. Suppose information is represented by associated vectors $f_1 \rightarrow f_1$, $f_2 \rightarrow f_2$ ... . A vector, f_k, is selected at random. Then the matrix, A, is incremented according to the rule

$$\Delta A = \eta (f_k - A f_k) f_k^T$$

where ΔA is the change in the matrix A. In the radar application, there is no 'correct answer' in the general sense of a supervised algorithm. However every input pattern can be its own 'teacher' in the error correction algorithm in that the network will try to better reconstruct that particular input pattern. The goal of learning a set of stimuli {f} is to have the system behave as

$$A f_k = f_k$$

The error correcting learning rule will approximate this result with a least mean squares approximation, hence the alternative name for the Widrow-Hoff rule: the LMS (least mean squares) algorithm. The autoassociative system combined with error correction, when working perfectly, is forcing the system to develop a particular set of eigenvectors with eigenvalue 1.

The eigenvectors of the connection matrix are also of interest when simple Hebbian learning is used in an autoassociative system. Then, the simple outer product associator has the form

$$\Delta A = \eta f_k f_k^T .$$

There is now an obvious connection between the eigenvectors of the resulting outer product connectivity matrix and the principal components of statistics, because the form of this matrix is the covariance matrix. In fact, there is growing evidence that many neural networks are doing something like principal component analyis. (See, for example, Baldi and Hornik, 1989 and Cottrell, Munro and Zipser, 1988).

BSB: A Dynamical System. We shall use for radar clustering a non-linear model that takes the basic linear associator, uses error correction to construct the connection matrix, and uses units containing a simple limiting non-linearity. Consider an autoassociative feedback system, where the vector output from the matrix is fed back into the input. Because feedback systems can become unstable, we incorporate a simple limiting non-linearity to prevent unit activity from getting too large or too small. Let f[i] be the current state vector describing the system. f[0] is the vector at step 0. At the i+1st step, f[i+1], the next state vector, is given by the iterative equation,

$$f[i+1] = \text{LIMIT}\ [\ \alpha A\ f[i] + \gamma\ f[i] + \delta\ f[0]\ ].$$

We stabilize the system by bounding the element activities within limits.

The first term, $\alpha Af[i]$, passes the current system state through the matrix and adds information reconstructed from the autoassociative cross connections. The second term, $\gamma f[i]$, causes the current state to decay slightly. This term has the qualitative effect of causing errors to eventually decay to zero as long as γ is less than 1. The third term, $\delta f[0]$, can keep the initial information constantly present and has the effect of limiting the flexibility of the possible states of the dynamical system since some vector elements are strongly biased by the initial input.

Once the element values for f[i+1] are calculated, the element values are 'limited', that is, not allowed to be greater than a positive limit or less than a negative limit. This is a particularly simple form of the sigmoidal nonlinearity assumed by most neural network model. The limiting process contains the state vector within a set of limits, and we have previously called this model the 'brain state in a box' or BSB model. (Anderson, Silverstein, Ritz, and Jones, 1977; Anderson and Mozer, 1981) The system is in a positive feedback loop but is amplitude limited. After many iterations, the system state becomes stable and will not change: these points are attractors in the dynamical system described by the BSB equation. This final state will be the output of the system. In the fully connected case with a symmetric connection matrix the dynamics of the BSB system can be shown to be minimizing an energy function. The location of the attractors is controlled by the learning algorithm. (Hopfield, 1982; Golden, 1986). Aspects of the dynamics of this system are related to the 'power' method of eigenvector extraction, since repeated iteration will leada to activity dominated by the eigenvectors with the largest postive eigenvalues. The signal processing abilities of such a network occur because eigenvectors arising from learning uncorrelated noise will tend to have small eigenvalues, while signal related eigenvectors will be large, will be enhanced by feedback, and will dominate the system state after a number of iterations.

We might conjecture that a category or a concept derived from many noisy examples would become identified with an attractor associated with a region in state space and that all examples of the concept would map into the point attractor. This is the behavior we want for radar pulse clustering.

Neural Network Clustering Algorithms. We know there will be many radar pulses, but we do not know the detailed descriptions of each emitter invoved. We want to develop the structure of the microwave environment, based on input information. A number of models have been proposed for this type of task, including various competitive learning algorithms (Rumelhart and Zipser, 1986; Carpenter and Grossberg, 1987).

Each pulse is different because of noise, but there are only a small number of emitters present relative to the number of pulses. We take the input data representing each pulse and form a state vector with it. A sample of several hundred pulses are stored in a 'pulse buffer.' We take a pulse at random and learn it, using the Widrow-Hoff error correcting algorithm with a small learning constant. Since there is no teacher, the desired output is assumed to be the input pulse data.

Learning rules for this class of dynamical system, Hebbian learning in general, (Hopfield, 1982) and the Widrow-Hoff rule in particular, are effective at 'digging holes in the energy landscape' so they fall where the vectors that are learned are. That is, the final low energy attractor states of the dynamical system when BSB

dynamics are applied will tend to lie near or on stored information. Suppose we learn each pulse as it comes in, using Widrow Hoff error correction, but with a small learning constant. Metaphorically, we 'dig a little hole' at the location of the pulse. But each pulse is different. So, after a while, we have dug a hole for each pulse, and if the state vectors coding the pulses from a single emitter are not too far apart in state space, we have formed an attractor that contains all the pulses from a single emitter, as well as new pulses from the same emitter. Figure 2 presents a (somewhat fanciful) picture of the behavior that we hope to obtain, where many nearby data points combine to give a single broad network energy minimum that contains them all.

Figure 2 about here

We can see why this behavior will occur from an informal argument. Call the average emitter state vector of a particular emitter p. Then, every observed pulse, f_k, will be

$$f_k = p + d_k ,$$

where d_k is a distortion, which will be assumed to be different for every individual pulse, that is, different d_k are uncorrelated, and are relatively small compared to p. With a small learning constant, and with the connection matrix A starting from zero, the magnitude of the output vector, Af, will also be small after only a few pulses are learned. This means that the error vector will point outward, toward f_k, that is, toward $p+d_k$, as shown in Figure 3.

Figure 3 about here

Early in the learning process with a small learning constant for a particular cluster, the error vectors (input minus output) all will point toward the cluster of input pulses. Widrow Hoff learning can be described as using a simple associator to learn the error vector. Since every d_k is different and uncorrelated, the error vectors from different pulses will have the average direction of p. The matrix will act as if it is repeatedly learning p, the average of the vectors. It is easy to show that if the centers of different emitter clusters are spaced far apart, in particular, if the cluster centers are orthogonal, then p will be close to an eigenvector of A. In more interesting and difficult cases, where clusters are close together or the data is very noisy, it is necessary to resort to numerical simulation to see how well the network works in practice. As we hope to show, this technique does work quite well.

After the matrix has learned so many pulses that the input and output vectors are of comparable magnitude, the output of the matrix when $p + d_k$ is presented will be near p. (See Figure 4) Then,

$$p \simeq Ap.$$

Over a number of learned examples,

$$\text{total error} \simeq \Sigma_k (p+d_k - A(p+d_k))$$

$$\simeq \Sigma_k (d_k - Ad_k)$$

The maximum values of the eigenvalues of A are 1 or below, the d's are uncorrelated, and this error term will average to zero.

Figure 4 about here

However, as the system learns more and more random noise, the average magnitude of the error vector will tend to get longer and longer, as the eigenvalues of A related to the noise become larger. Note that system learning never stops because there is always an error vector to be learned, which is a function of the intrinsic noise in the system. Therefore, there is a 'senility' mechanism found in this class of neural networks. For example, the covariance matrix of independent, identically distributed Gaussian noise added to each element is proportional to the identity matrix, then every vector becomes an eigenvector with the same eigenvalue, and this matrix is the matrix toward which A will evolve, if it continues to learn random noise indefinitely. When the BSB dynamics are applied to matrices resulting from learning very large numbers of noisy pulses, the attractor basins become fragmented, so that the clusters break up. However, the period of stable cluster formation is very long and it is easy to avoid cluster breakup in practice. (Anderson, 1987)

In BSB clustering the desired output is a particular stable state. Ideally, all pulses from one emitter will be attracted to that final state. Therefore a simple identity check is now sufficient to check for clusters. This check is performed by resubmitting the original noisy pulses to the network that has learned them and forming a list of the stable states that result. The list is then compared with itself to find which pulses came from the same emitter. For example, a symbol could be associated with the pulses from the same final state, i.e. the pulses have been deinterleaved or identified.

Once the emitters have been identified, the average characteristics of the features describing the pulse (frequency, pulse width and pulse repetition pattern) can be computed. These features are used to classify the emitters with respect to known emitter types in order to 'understand' the microwave environment. A two stage system, which first clusters and then counts clusters is easy to implement, and, practically, allows convenient 'hooks' to use traditional digital techniques in conjunction with the neural networks.

Stimulus Coding and Representation. The fundamental represention assumption of almost all neural networks is that information is carried by the pattern or set of activities of many neurons in a group of neurons. This set of activities carries the

meaning of whatever the nervous system is doing and these sets of activities are represented as state vectors. The conversion of input data into a state vector, that is, the representation of the data in the network, is the single most important engineering problem faced in network design. In our opinion, choice of a good input and output representation is usually more important for the ultimate success of the system than the choice of a particular network algorithm or learning rule.

We now suggest an explicit representation of the radar data. From the radar receiver, we have a number of continuous valued features to represent: frequency, elevation, azimuth, pulse width, and signal strength. Our approach is to code continuous information as locations on a topographic map, i.e. a bar graph or a moving meter pointer. We represent each continuous parameter value by location of block of activation on a linear set of elements. Increase in a parameter value moves the block of activity to the right, say, and a decrease, moves the activity to the left. We have used a more complex topographic representation in several other contexts, with success. (Sereno, 1989; Rossen, 1989; Viscuso, Anderson, and Spoehr, 1989).

We represent the block/bar of activity value with a block (three or four) "=", equal, symbols placed in a region of ".", period, symbols. Single characters are coded by eight bit ASCII bytes. The ASCII 1's and 0's are further transformed to +1's and -1's, so that the magnitude of any feature vector is the same regardless of the feature value. Input vectors are therefore purely binary. On recall, if the vector elements coding a character do not rise above a threshold size, the system is not 'sure' of the output. Then that character is represented as the underline, "_", character. Being 'not sure' can be valuable information relative to the confidence of a particular output state relative to an input. Related work has developed a more numeric, topographic representation for this task, called a 'closeness code' (Penz, 1987) which has also been successfully used for clustering of simulated radar data.

Neural networks can incorporate new information about the signal and make good use of it. This is one version of what is called the data fusion or sensor fusion problem. To code the various radar features, we simply concatenate the topographic vectors of individual feature into a single long state vector. Bars in different fields code the different quantities. Figure 5 shows these fields.

Figure 5 about here

Below we will gradually add information to the same network to show the utility of this fusion methodology. The conjecture is is that adding more information about the pulse will produce more accurate clustering. Note that we can insert 'symbolic' information (say word identifications or other appropriate information) in the state vector as character strings, forming a hybrid code. For instance the state vector can contain almost unprocessed spectral data together with the symbolic bar graph data combined with character strings representing symbols at the same time.

A Demonstration. For the simulations of the radar problem that we describe next, we used a BSB system with the following properties. The system used 480 units, representing 60 characters. Connectivity was 25%, that is, each element was connected at random to 120 others. There were a total of 10 simulated emitters with considerable added intrinisic noise. A pulse buffer of 510 different pulses was used for learning and, after learning, 100 new pulses, 10 from each emitter were used to

test the system. There were about 2000 total learning trials, about that is, about four presentations per example. Parameter values were $\alpha = 0.5$, $\gamma = 0.9$ and $\delta = 0$. The limits for thresholding were +2 and -2. None of these parameters were critical, in that moderate variations of the parameters had little effect on the resulting classifications of the network.

Suppose we simply learn frequency information. Figure 6 shows the total number of attractors formed when ten new examples of each of ten emitters were passed through the BSB dynamics, using the matrix formed from learning the pulses in the pulse buffer. In a system that clustered perfectly, exactly 10 final states would exist, one different final state for each of the ten emitters. However, with only frequency information learned, all the 100 different inputs mapped into only two attractors.

Figure 6 about here

Figure 6 and others like it below are graphical indications of the similarity between recalled clusters or states with computational energy minima. The states shown in the figures are ordered via a priori knowledge of the emitters, although this information was obviously not given to the network. One can visually interpret the outputs for equality of two emitters (lumping of different emitters) or separation of outputs for a single emitter (splitting of the same emitter) in the outputs. This display method is for the reader's benefit. The ANSP system determines the number and state vector of separate minima by a dot product search of the entire output list, as discussed above. Position of the bar of '='s codes the frequency in the frequency field which is the only field learned in this example.

Let us now give the system additional information about pulse azimuth and elevation. Clustering performance improves markedly, as shown in Figure 7. We get nine different attractors. There is still uncertainty in the system, however, since few corners are fully saturated, as indicated by the underline symbols on the corners of some bar's. States 1 and 3 are in the same attractor, an example of incorrect 'lumping' as a result of insufficient information. Two other final states (8 and 9) are very close to each other in Hamming distance.

Figure 7 about here

Let us assume that future advances in receivers will allow a quick estimation of the microstructure of each radar pulse. We have used, as shown in Figure 8, a coding which is a crude graphical version of a Fourier anlysis of an individual pulse, with the center frequency located at the middle of the field. Emitter pulse spectra were assigned arbitrarily.

Figure 8 about here

Note that the spectral information can be included in the state vector in only slightly processed form: we have included almost a caricature of the actual spectrum.

Addition of spectral information improved performance somewhat. There were nine distinct attractors, though still many unsaturated states. Two emitters were still 'lumped', 8 and 9. Figure 9 shows the results.

Figure 9 about here

Suppose we add information about pulse width to azimuth, elevation, and frequency. The simulated pulse width information is very poor. It actually degrades performance, though it does allow separation of a couple of nearby emitters. The results are given in Figure 10.

Figure 10 about here

The reason pulse width data is of poor quality and hurts discrimination is because of a common artifact due to the way that pulse width is measured. When two pulses occur close together in time a very long pulse width is measured by the receiver circuitry. This can give rise in unfavorable cases to a spurious bimodal distribution of pulsewidths for a single emitter. Therefore, a single emitter seems to have some short pulse widths and some very long pulse widths and this can split the category. Bimodal distributions of an emitter parameter, when the peaks are widely separated, is a hard problem for any clustering algorithm. A couple of difficult discriminations in this simulation, however, are aided by the additional data.

We now combine all this information about pulse properties together. None of the subsets of information could perfectly cluster the emitters. Pulse width, in particular, actually hurt performance. Figure 11 shows that, after learning, using all the information, we now get ten well separated attractors, i.e. the correct number of emitters relative to the data set. The conclusion is that the additional information, even if it was noisy, could be used effectively. Poor information could be combined with other poor information to give good results.

Figure 11 about here

Processing After Deinterleaving. Having used the ANSP system to deinterleave and cluster data, we also have a way of producing an accurate picture of each emitter. We now have an estimate of the frequency and pulse width and can derive other emitter properties (Penz et. al., 1989), for example, the emitter pulse repetition pattern. One method to learn this pattern is to learn pulse repetition interval (PRI) pairs autoassociatively. Another is to autocorrelate the PRI's of a string. This technique probably provides more information than any other for characterizing emitters, because the resulting correlation functions are very useful for characterizing a particular emitter type.

Classification Problem and Neural Network Data Bases. The next task is to classify the observed emitters based on our previous experience with emitters of various types. We continue with the neural network approach because of the ability of networks to incorporate a great deal of information from different sensors, their ability to generalize (i.e. 'guess') based on noisy or incomplete information, and their ability to handle ambiguity. Known disadvantages of neural networks used as data bases are their slow computation using traditional computer architectures, erroneous generalizations (i.e. 'bad guesses'), their unpredictability, and the difficulty of adding new information to them, which may require time consuming relearning.

Information, in traditional expert systems, is often represented as collections of atomic facts, relating pairs or small sets of items together. Expert systems often assume 'IF (x) THEN (y)' kinds of information representation. For example, such a rule in radar might look like:

IF (Frequency is 10 gHz)
AND (Pulse Width is 1 microsecond)
AND (PRI is constant at 1 kHz)

THEN (Emitter is a Klingon air traffic control radar).

Problems with this approach are that rules usually have many exceptions, data may be erroneous or noisy, and emitter parameters may be changed because of local conditions. Expert systems may be exceptionally prone to confusion when emitter properties change because of the rigidity of their data representation. Neural networks allow a different strategy: Always try to use as much information as you have, because, in most cases, the more information you have, the better performance will be.

As William James commented in the nineteenth century,

> ... the more other facts a fact is associated with in the mind, the better posession of it our memory retains. Each of its associates becomes a hook to which it hangs, a means to fish it up by when sunk beneath the surface. Together, they form a network of attachments by which it is woven into the entire

tissue of our thought.

William James (1890). p. 301

Perhaps, as William James suggests, information is best represented as large sets of correlated information. We could represent this in a neural network by a large, multimodal state vector. Each state vector contains a large number of 'atomic facts' together with their cross correlations. Our clustering demonstration showed that more information could be added and used efficiently and that identification depends on a cluster of information co-occuring. (See Anderson, 1986 for further discussion of neural network data bases of this type.)

Ultimately, we would like a system that would tentatively identify emitters based on measured properties and previously known information. Since we know, in operation, that parameters can and often do change, we can never be sure of the answers.

As a specific important example, radar systems can shift parameters in ways consistent with their physical design, that is, waveguide sizes, power supply size, and so on, for a number of reasons, for example, weather conditions. If an emitter is characterized by only one parameter, and that parameter is changed, then identification becomes very unlikely. Therefore, accuracy of measurement of a particular parameter may not be as useful for classification as one might expect. However, using a whole set of co-occuring properties, each at low precision, may prove a much more efficient strategy for identification. For further discussion of how humans often seem to use such a strategy in perception, consult George Miller's classic 1956 paper, "The magic number seven, plus or minus two."

Classification Problem for Shifted Emitters. Our first neural net classification simulation is specifically designed to study sensitivity to shifts in parameters. Two data sets were generated. One set has 'normal' emitter properties and the other set had all the emitter properties changed about 10 percent. The two sets each contained about 500 data points. The names used are totally arbitrary. The state vector was constructed of a name string (the first 10 characters) and bar codes for frequency, pulse width, and pulse repetition interval. For the classification function, the position of "+" symbols indicates the feature magnitude while the blank symbol fills the rest of the feature field. Again the "_" symbol indicates an undecided node.

Figures 12 and 13 show the resulting attractor interpretations. Figure 12 shows the vectors to be learned autoassociatively by the BSB model. The first field is the emitter name. The last three fields represent the numerical information produced by the deinterleaver and pulse repetition interval modules. An input consists of leaving the identification blank and filling in the analog information for the emitter which one wants an identification. The autoassocative connections fill in the missing identification information.

Figure 12 shows the identifications produced when the normal set is provided to the matrix: all the names are produced correctly and in a small number of iterations through the BSB algorithm. Figure 13 uses the same matrix, but the input data is now derived from sources whose mean values are shifted about 10 percent, to emulate this parameter shift.

Figure 12 about here

Figure 13 about here

There were three errors of classification. Emitter 3 was classified as 'Airborn In' instead of 'AA FC'. Emittter 4 was classified as 'SAM target' instead of 'Airborn In'. Emitter 7 was classified as 'Airborn In' rather than the correct 'SAM Target' name. Note that the recalled analog information is also not exactly the correct analog information even for the correctly identified emitters. At a finer scale, the number of iterations required to reach an attractor state was very long. This is a direct measure of the uncertainty of the neural network about the shifted data. Some of the final states were not fully limited, another indication of uncertainty.

Large Classification Data Bases. It would be of interest to see how the system worked with a larger data base. Some information about radar systems is published in Jane's Weapon Systems (Blake, 1988). We can use this data as a starting point to see if a neural network might scale to larger systems. Figure 14 shows the kind of data available from Jane's. Some radars have constant pulse repetition frequency (PRF) and others have highly variable PRF's. (Jane's lists Pulse Repetition Frequency (PRF) in its tables instead of Pulse Repetition Interval (PRI). We have used their term for their data in this simulation.) We represented PRF variability in the state vector coding by increasing the last bar width (Field 7, Figure 15) for highly variable PRF's (see the Swedish radar, for an example.) Also, when a parameter is out of range (the average PRF of the Swedish radar) it is not represented.

Figure 14 about here

Figure 15 about here

We perform the usual partitioning of the state vector into fields, as shown in Figure 15. For this simulation, the frequency scale is so coarse that even enormous changes in frequency would not change the bar coding significantly. We are more interested here in whether the system can handle large amounts of Jane's data. We taught the network 47 different kinds of radar transmitters. Some transmitter names were represented by more than one state vector because they can have several, quite different modes of operation, that is, the parameter part of the code can differ significantly from mode to mode. (The clustering algorithms would almost surely pick

up different modes as different clusters.) After learning, we provided the measured properties to the the transmitter to see if it could regenerate the name of the country that the radar belonged to. There were only three errors of retrieval from 47 sets of input data, corresponding to 94 percent accurate country identification. This experiment was basically coding a lookup table, using low precision representations of the parameters. Figure 16 shows a sample of the output, with reconstructions of the country, designations, and functions.

Figure 16 about here

Conclusions. We have presented a system using neural networks which is capable of clustering and identifying radar emitters, given as input data large numbers of received radar pulses and with some knowledge of previously characterized emitter types.

Good features of this system are its robustness, its ability to integrate information from co-occurance of many features, and its ability to integrate information from individual data samples.

We might point out that the radar problem is similar to data analysis problems in other areas. For example, it is very similar to a problem in experimental neurophysiology, where action potentials from multiple neurons are recorded with a single electrode. Applications of the neural network techniques described here may not be limited to radar signal processing.

References

Anderson, J.A. (1972), A simple neural network generating an interactive memory, Mathematical Biosciences, 14, 197-220.

Anderson, J.A. (1983), Neural models for cognitive computation. IEEE Transactions: Systems, Man, and Cybernetics, SMC-13, 799-815.

Anderson, J.A. (1986), Cognitive Capabilities of a Parallel System. In E. Bienenstock, F. Foglemann, and G. Weisbuch. (Eds.) Disordered Systems and Biological Organization, Berlin: Springer.

Anderson, J.A. (1987), Concept formation in neural networks: Implications for evolution of cognitive functions. Human Evolution, 2, 1987.

Anderson, J.A. and Mozer, M.C. (1989), Categorization and selective neurons, In: G.E. Hinton and J.A. Anderson (Editors), Parallel Models of Associative Memory (Rev. Ed.), Hillsdale, NJ: Erlbaum.

Anderson, J.A. and Murphy, G.L. (1986), Psychological Concepts in a Parallel System. In J.D. Farmer, A. Lapedes, N. Packard, and B. Wendroff. (Eds.) Evolution, Games, and Learning.. New York: North Holland.

Anderson, J.A. and Rosenfeld, E., Eds. (1988) Neurocomputing: Foundations of Research, Cambridge, MA: MIT Press.

Anderson, J.A., Silverstein, J.W, Ritz, S.A., and Jones, R.S., Distinctive features, categorical perception, and probability learning: Some applications of a neural model, Psychological Review, 84, 413-451.

Baldi, P. and Hornik, K. (1989), Neural networks and principal component analysis: Learning from examples without local minima. Neural Networks, 2, 53.

Blake, B. (Editor), (1988), Jane's Weapon Systems, (19th Edition), Surrey, U.K.: Jane's Information Group.

Carpenter, G. and Grossberg, S. (1987), ART 2: self organization of stable category recognition codes for analog input patterns, Applied Optics, 26, 4919-4942.

Cottrell, G.W., Munro, P.W. and Zipser, D. (1988). Image compression by back propagation: A demonstration of extensional programming. In N.E. Sharkey (Ed.), Advances in Cognitive Science, Vol. 3. Norwood, NJ: Ablex.

Golden, R.M. (1986), The "Brain state in a box" neural model is a gradient descent algorithm, Journal of Mathematical Psychology, 30, 73-80.

Hinton, G.E. and Anderson, J.A. (Eds., 1989), Parallel Models of Associative Memory (Rev. Ed.), Hillsdale, NJ: Erlbaum.

James, W. (1961/1894), Briefer Psychology, New York: Collier.

Knapp A. and Anderson, J.A. (1984), A theory of categorization based on distributed memory storage. Journal of Experimental Psychology: Learning, Memory and Cognition. 9, 610-622.

Kohonen, T. (1972). Correlation matrix memories, IEEE Transactions on Computers, C-21, 353-359.

Kohonen, T. (1977). Associative Memory. Berlin: Springer.

Kohonen, T. (1984). Self Organization and Associative Memory. Berlin: Springer.

McClelland, J.L. and Rumelhart, D.E., Eds. (1986), Parallel, Distributed Processing, Volume 2, Cambridge, MA: MIT Press.

Miller, G.A. (1956), The magic number seven, plus or minus two: Some limits on our capacity for processing information, Psychological Review,63, 81-97.

Penz, P.A. (1987), The closeness code, Proceedings of IEEE International Conference on Neural Networks, III-515, IEEE Catalog 87th0191-7.

Penz, P.A., Katz, A.J., Gately, M.T., Collins, D.R. and Anderson, J.A. (1989), Analog capabilities of the BSB model as applied to the anti-radiation homing missile problem, Proceedings of the International Joint Conference on Neural Nets, II-7.

Rossen, M.L. (1989). Speech Syllable Recognition with a Neural Network, Ph.D. Thesis, Department of Psychology, Brown University, Providence, RI 02912, May, 1989.

Rumelhart, D.E. and McClelland, J.L., Eds. (1986), Parallel, Distributed Processing, Volume 1, Cambridge, MA: MIT Press.

Rumelhart, D.E. and Zipser, D. (1986), Feature discovery by competitive learning, In D.E. Rumelhart, and J.L. McClelland, Eds., Parallel, Distributed Processing, Volume 1, Cambridge, MA: MIT Press.

Sereno, M.E. (1989)A Neural Network Model of Visual Motion Processing, Ph.D. Thesis, Department of Psychology, Brown University, Providence, RI, May, 1989.

Viscuso, S.R., Anderson, J.A. and Spoehr, K.T., (1989) Representing simple arithmetic in neural networks, In G. Tiberghien, Ed., Advanced Cognitive Science: Theory and Applications , London: Horwoods.

Figures for

Radar Signal Categorization using a Neural Network

James A. Anderson

Department of Cognitive and Linguistic Sciences

Box 1978

Brown University, Providence, RI 02912

and

Michael T. Gately, P. Andrew Penz, and Dean R. Collins

Central Research Laboratories, Texas Instruments

Dallas, Texas 75265

Figures, Anderson, Gately, Penz and Collins

Caption, Figure 1

Block diagram of the radar clustering and categorizing system.

Caption, Figure 2

Landscape surface of system energy. Several learned examples may contribute to the formation of a single energy minimum which will correspond to a single emitter. This drawing is only for illustrative purposes and is not meant to represent the very high dimensional simulations actually used.

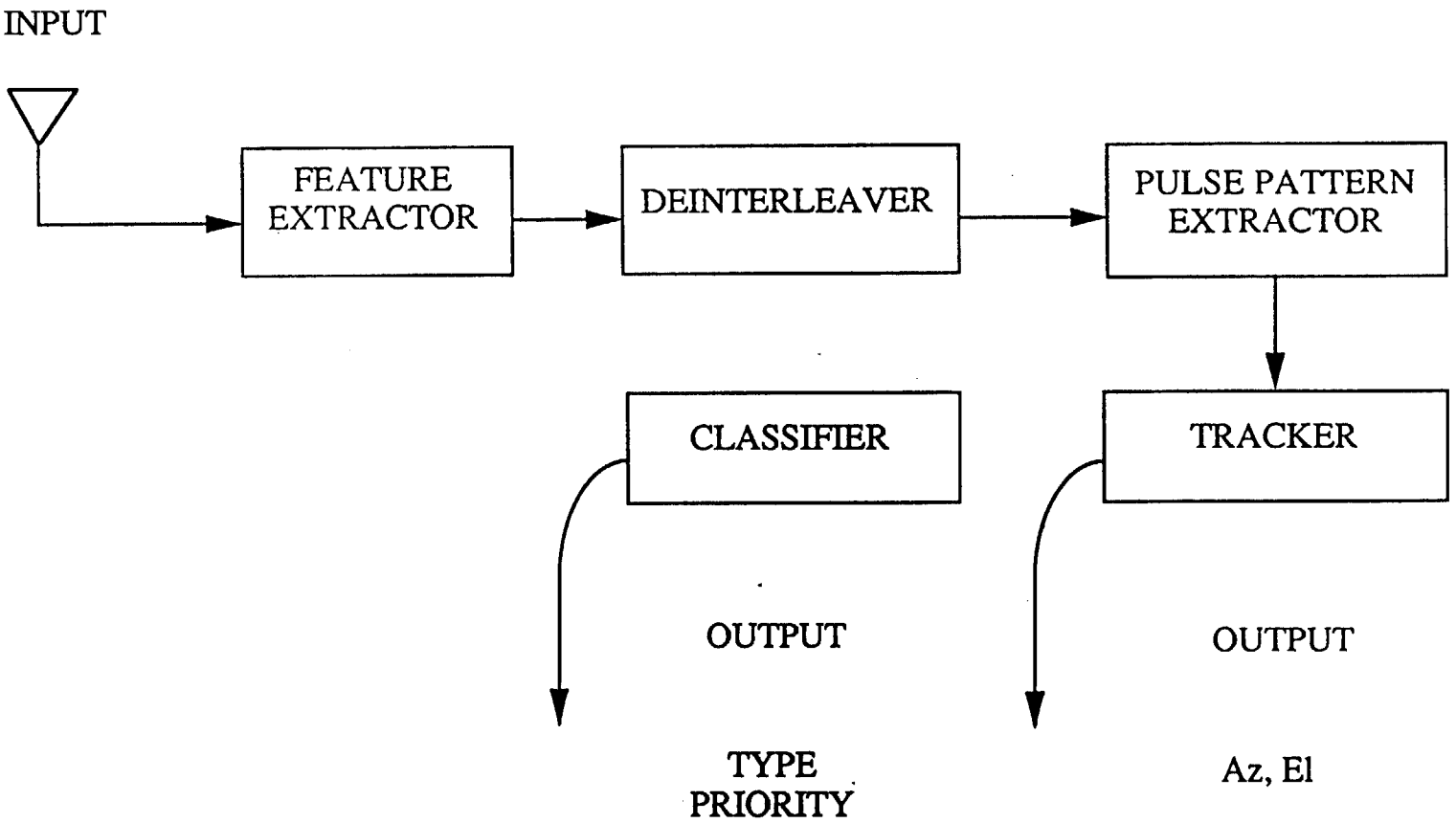
Caption, Figure 3

The Widrow-Hoff procedure learns the error vector. The error vectors early in learning with a small learning constant point toward examples, and the average of the error vectors will point toward the category mean, i.e. all the examples of a single emitter.

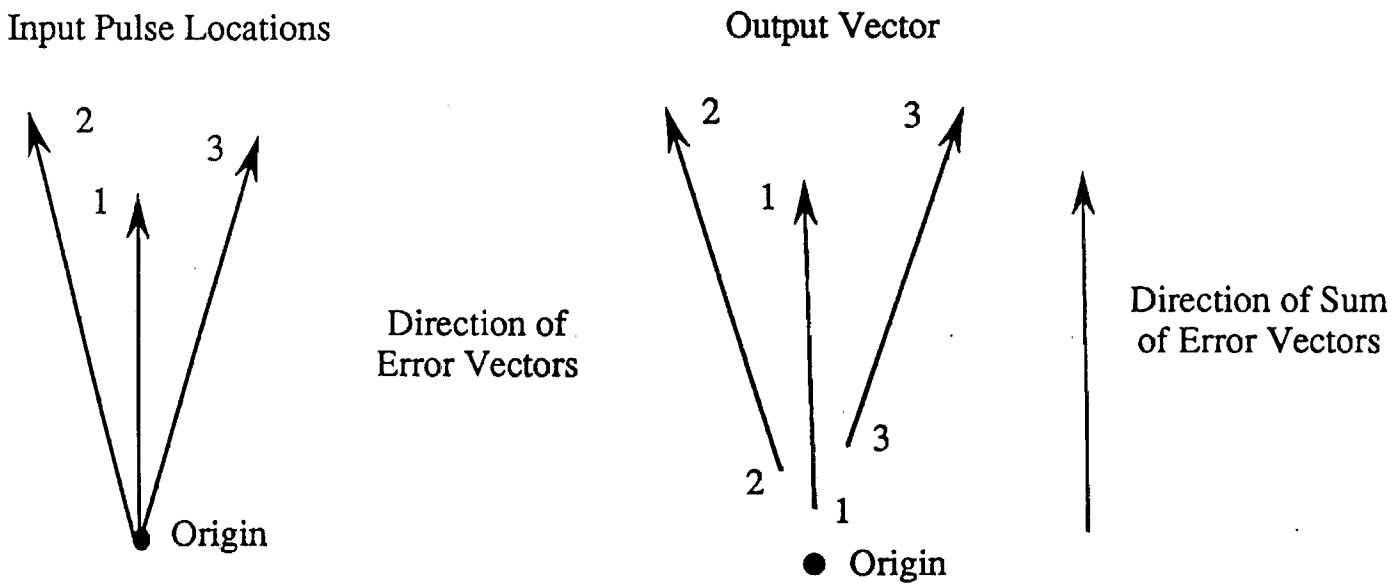
Caption, Figure 4

Assume an eigenvector is close to a category mean, as will be the result after extensive error correcting, autoassociative learning. The error terms from many learned examples, with a small learning constant, will average to zero and the system attractor structure will not change markedly. (There are very long term 'senility' mechanisms with continued learning, but they are not of practical importance for this application.)

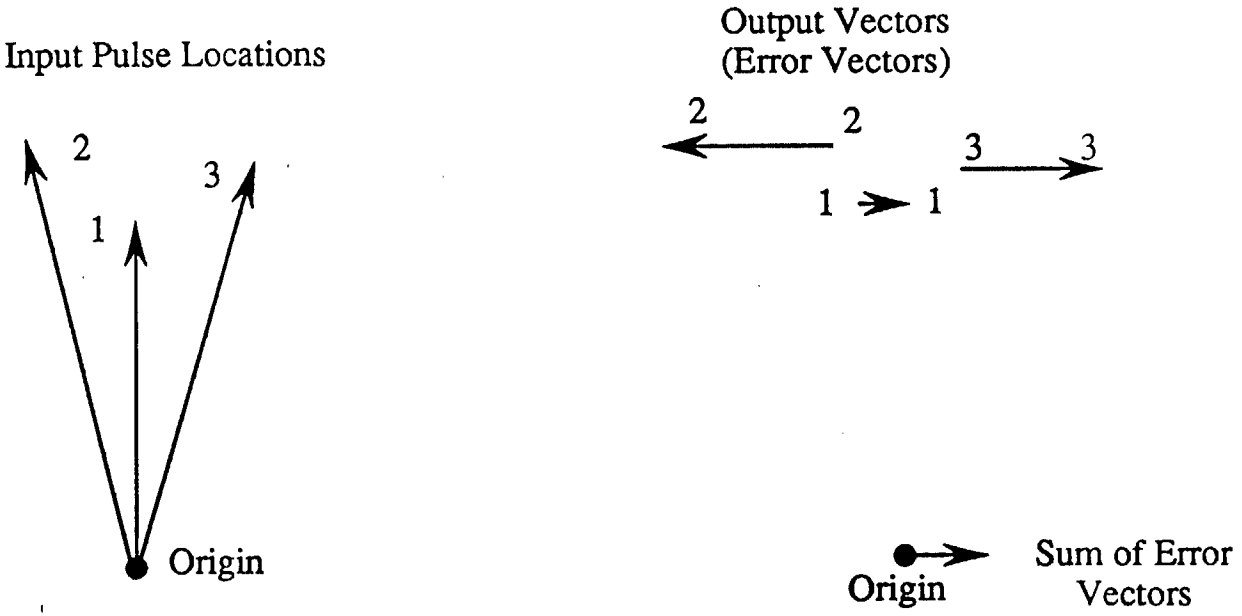
FUNCTIONAL FLOW



Early in Learning Process
Small Learning Constant, Many Examples



Late in Learning Process
Small Learning Constant, Many Examples



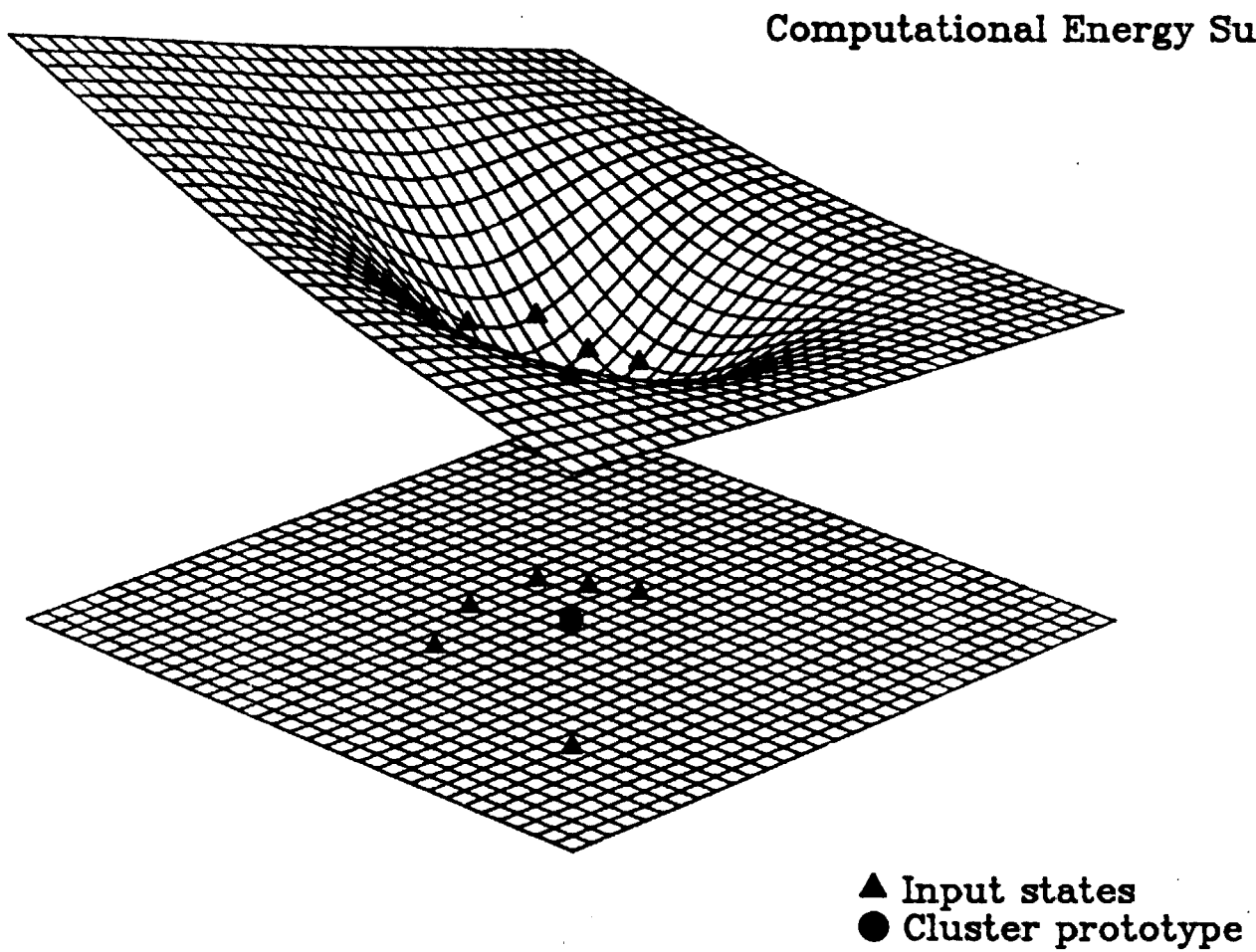
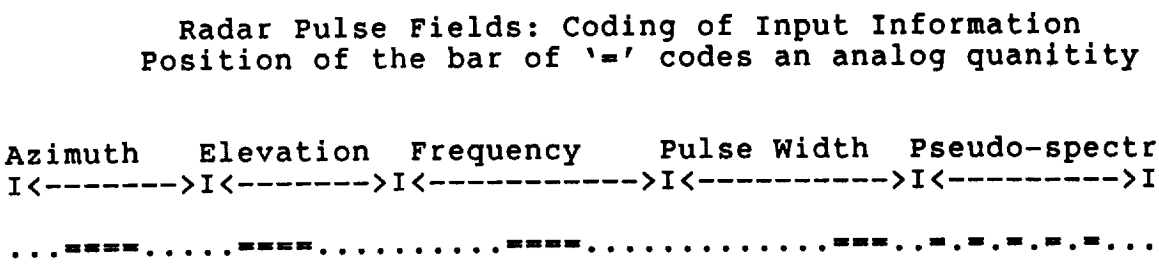

Computational Energy Surface
▲ Input states
● Cluster prototype

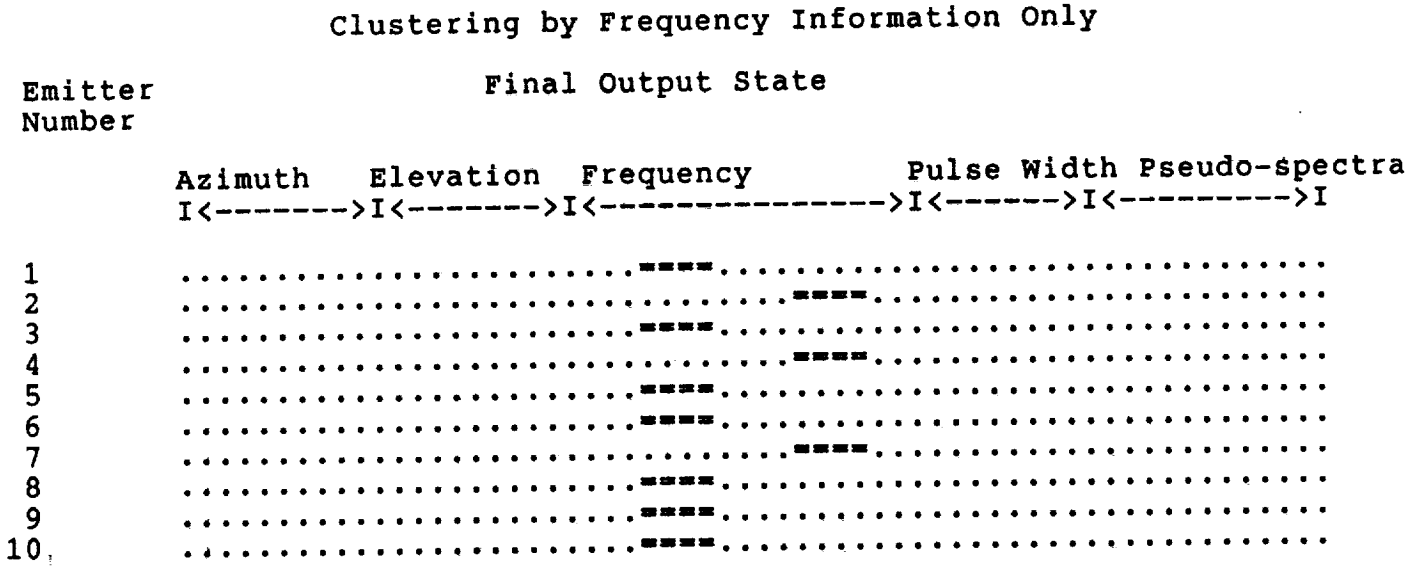
Figure 5



Caption, Figure 5

Input representation of analog input data uses bar codes. The state vector is partitioned into fields, corresponding to azimuth, elevation, frequency, pulse width, and a field corresponding to additional information that might become available with advances in receiver technology.

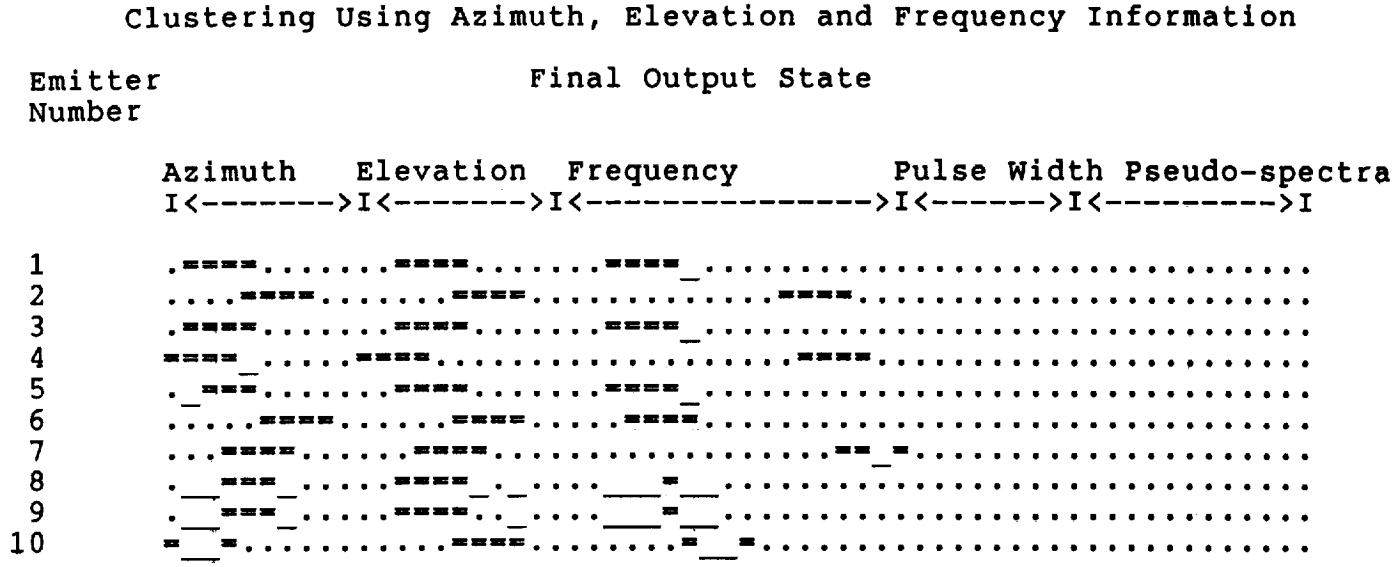
Figure 6



Caption, Figure 6

Final attractor states when only frequency information is learned. Ten different emitter are present, but only two different output states are found.

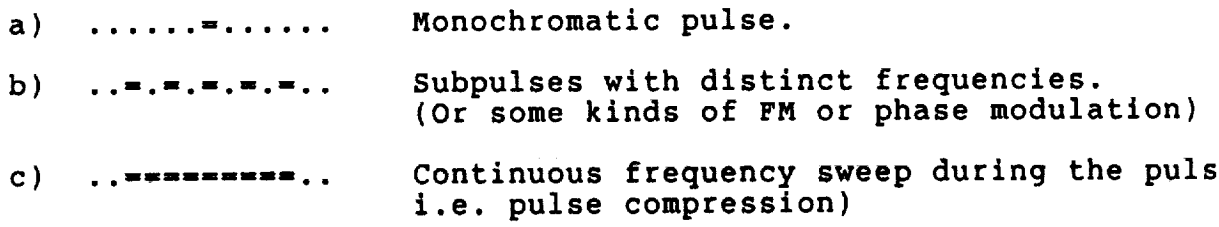
Figure 7



Caption, Figure 7

When azimuth, elevation and frequency are provided for each data point, performance is better. However, two emitters are lumped together, and three others have very close final states.

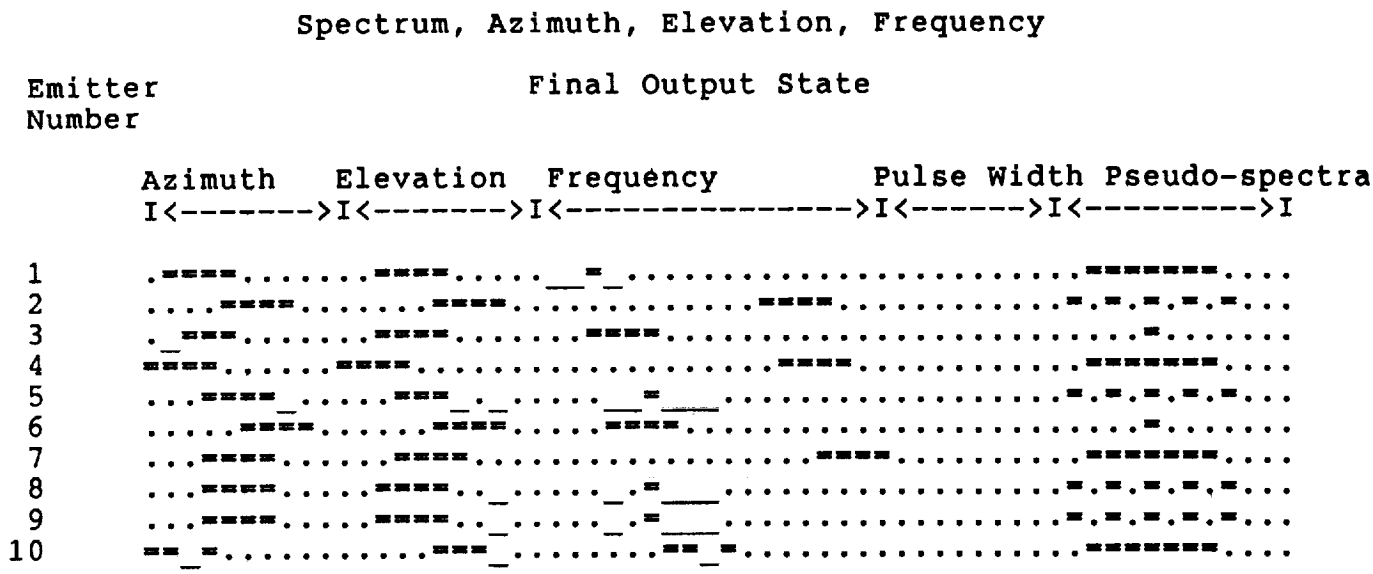
Figure 8



Caption, Figure 8

Suppose we can assume that advances in receiver technology will allow us to incorporate a crude 'cartoon' of the spectrum of an individual pulse into the coding of the state vector representing an example. The spectral information can be included in the state vector in only slightly processed form.

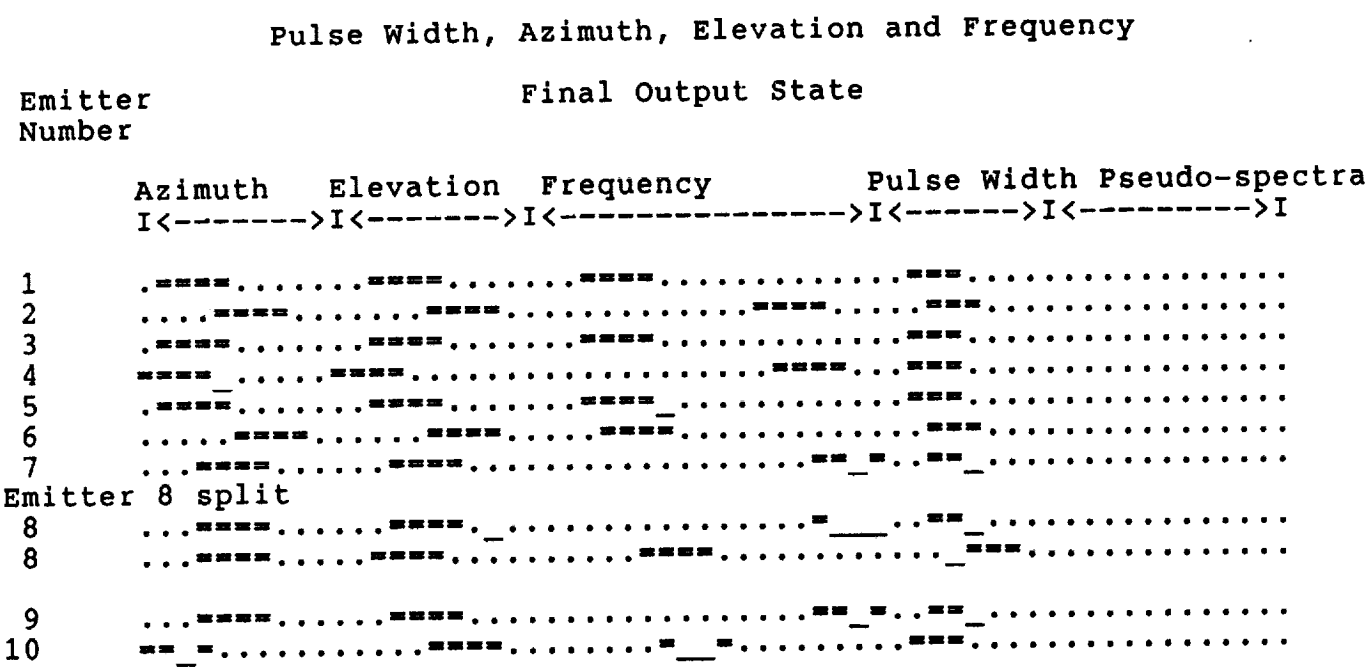
Figure 9



Caption, Figure 9

Including pseudo-spectral information helped performance considerably. Only two emitters are lumped and the other emitters are well separated.

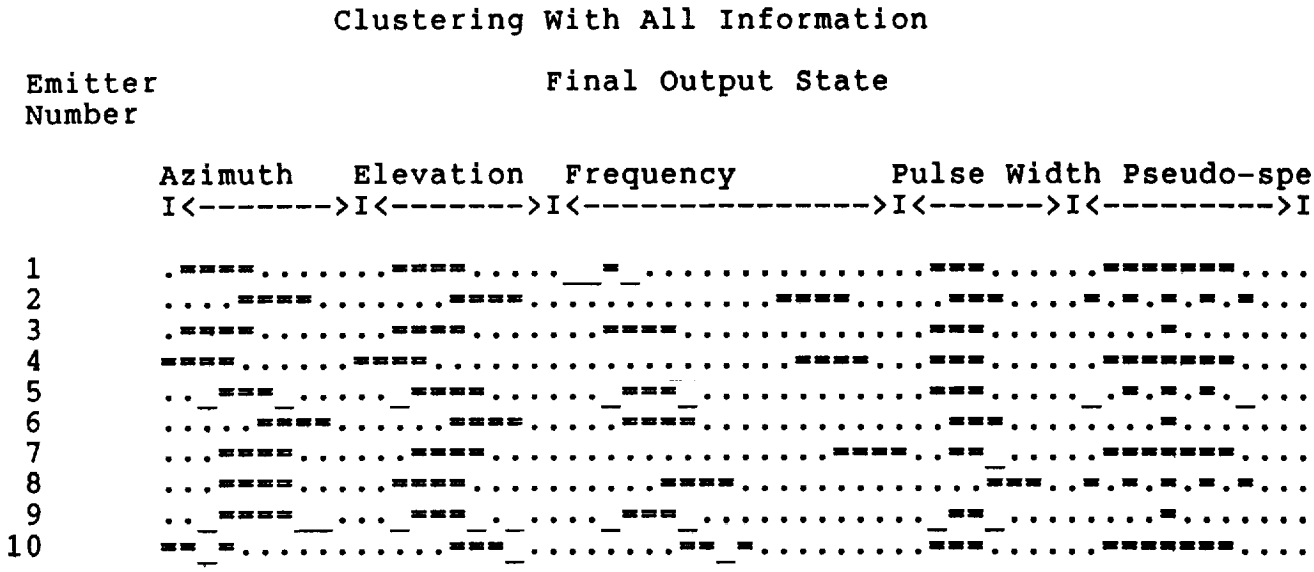
Figure 10



Caption, Figure 10

Suppose we add pulse width information to our other information. Pulse width data is of poor quality because when two pulses occur close together, a very long pulse width is measured by the receiver circuitry. This gives rise to a bimodal distribution of pulsewidths, and the system splits one category.

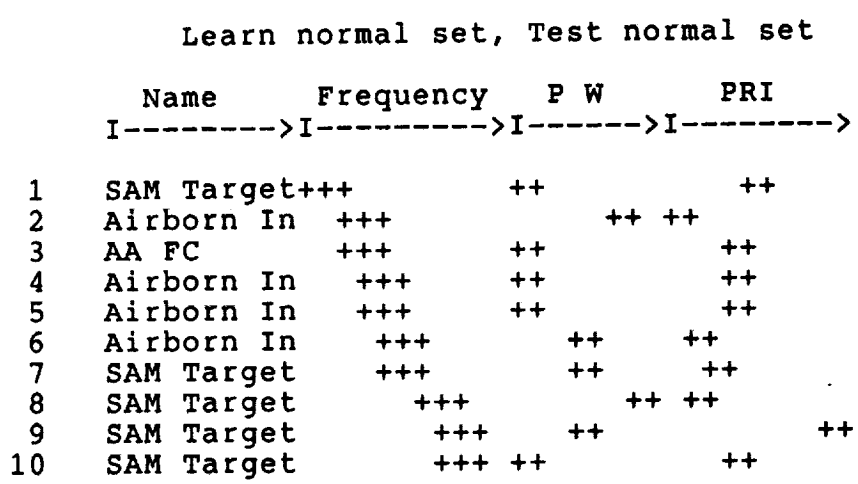
Figure 11



Caption, Figure 11

When all available information is used, ten stable, well separated attractors are formed. This shows that such a network computation can make good use of additional information.

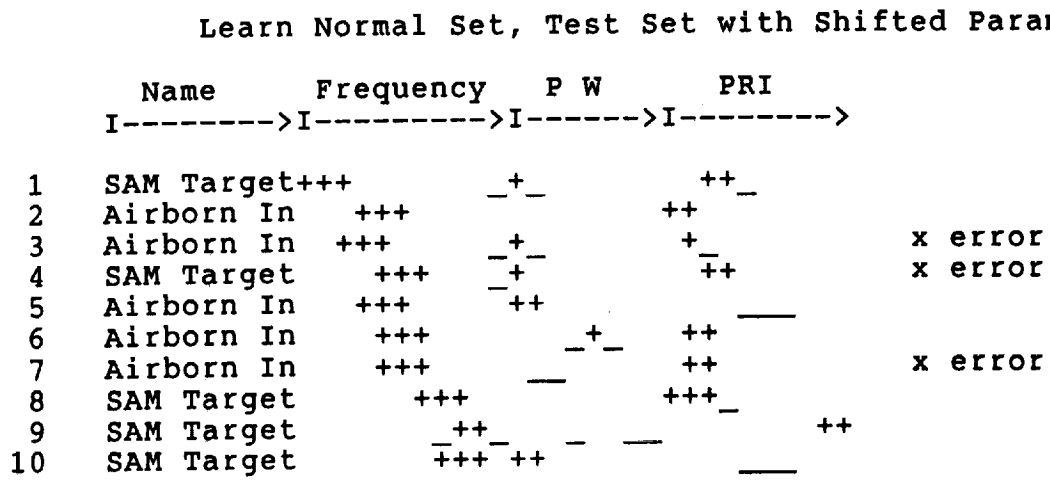
Figure 12



Caption, Figure 12

We can attach identification labels to emitters along with representations of their analog parameters. The names and values used here are random and were chosen arbitrarily.

Figure 13



Caption, Figure 13

Even if the emitter parameters shift slightly, it is still possible to make some tentative emitter identifications. Three errors of identification were made. Neural networks are able to generalize to some degree, if the representations are chosen properly. The names and values used here are random and were chosen arbitrarily.

Figure 14

Sample Data Obtained from Jane's Weapon Systems

Three Radars from Jane's:

China, JY-9, Search
Frequency : 2.0 - 3.0 gHz
Pulse Width : 20 microseconds
PRF : 0.850 kHz
PRF Variance: Constant frequency

Sweden, UAR1021, Surveillance
Frequency : 8.6 - 9.5 gHz
Pulse Width : 1.5 microseconds
PRF : 4.8 - 8.1 kHz
PRF Variance: 3 frequency staggered

USA, APQ113, FireControl
Frequency : 16 - 16.4 gHz
Pulse Width : 1.1 microseconds
PRF : 0.674 kHz
PRF Variance: None (Constant frequency)

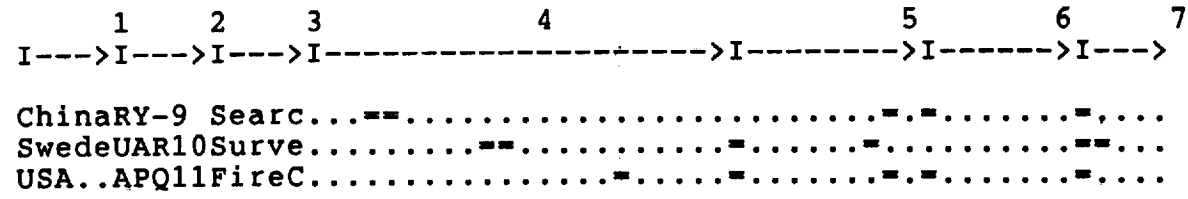
Caption, Figure 14

Sample data on radar transmitters taken from Jane's Weapon Systems. (Blake, 1988).

Figure 15

Coding into Partitioned State Vector:

Symbolic Fields:		Continuous Fields:	
Field 1	Country	Field 4	Frequency
Field 2	Designation	Field 5	Pulse Width
Field 3	Purpose	Field 6	PRF
		Field 7	PRF Variation



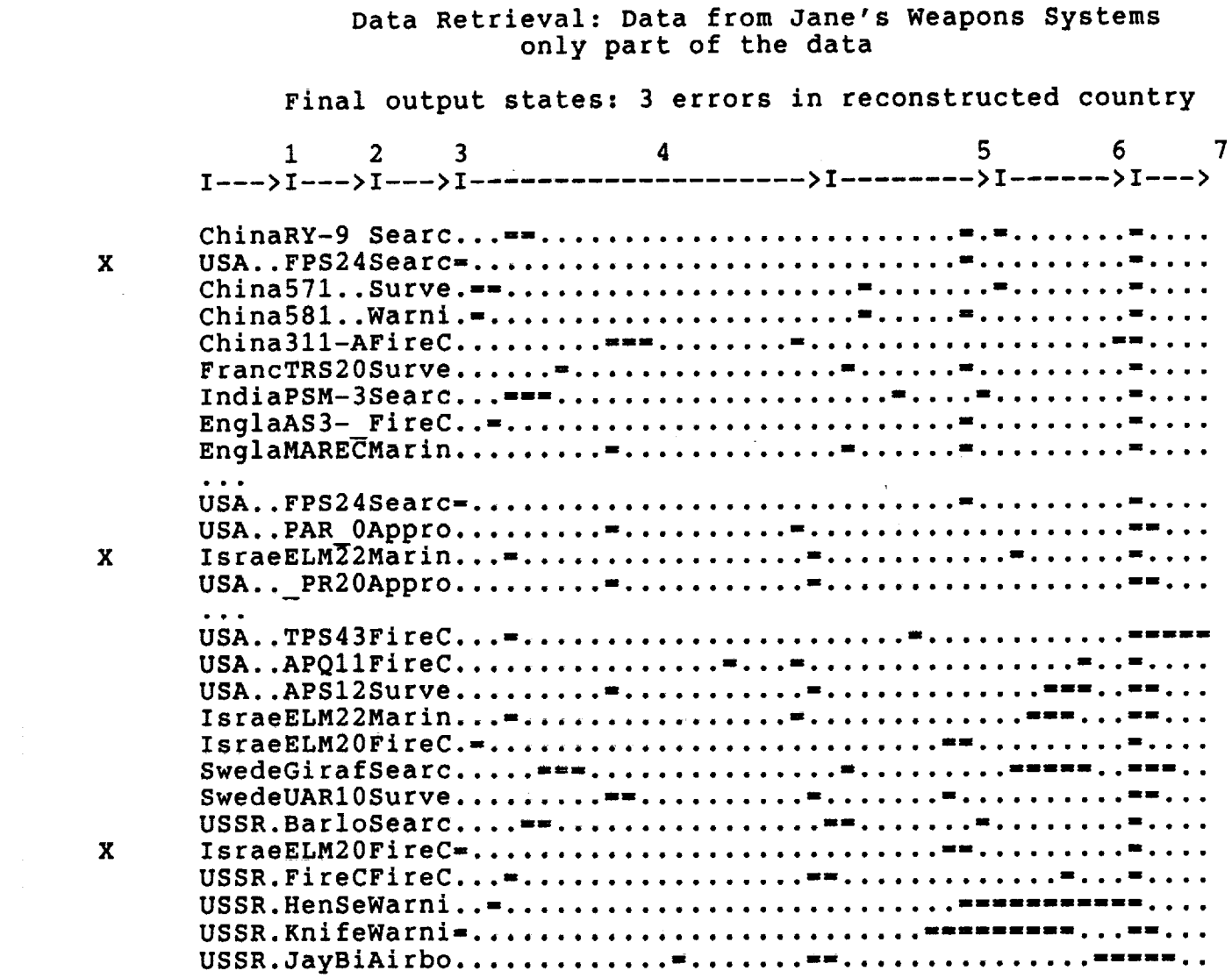
Analog Bar Code Ranges:

Frequency:	0 - 14 gHz
Pulse Width:	0 - 10 microseconds
PRF:	0 - 4 kHz
PRF Variance:	0 - 200% of average PRF

Caption, Figure 15

Bar code representation of Jane's data. Note the presence of both symbolic information such as country name and transmitter designation, and analog, bar coded information such as frequency, pulse width, etc.

Figure 16



Caption, Figure 16

When only analog data is provided at the input, the network will fill in the most appropriate country name. In this trial simulation, a network learned 47 different transmitters and was able to correctly retrieve the associated country in 43 of them.

SELF-ORGANIZATION AND CLUSTERING ALGORITHMS

James C. Bezdek
Div. of Computer Science
University of West Florida
Pensacola, Florida 32514

This research was partially supported by NSF Grant # IRI-9003252

ABSTRACT

Kohonen's "feature maps" approach to clustering is often likened to the k or c-means clustering algorithms. In this note we identify some similarities and differences between the hard and fuzzy c-Means (HCM/FCM) or ISODATA algorithms and Kohonen's "self-organizing" (KSO) approach. We conclude that some differences are significant, but at the same time there may be some important unknown relationship(s) between the two methodologies. We propose several avenues of research which, if successfully resolved, would strengthen both the HCM/FCM and Kohonen clustering models. We do not, in this note, address aspects of the KSO method related to associative memory and to the feature map display technique.

1. INTRODUCTION

Treatments of many classical approaches to clustering appear in Kohonen [1], Bezdek [2], and Duda and Hart [3]. Kohonen's work has become particularly timely in recent years because of the widespread resurgence of interest in Artificial Neural Network (ANN) structures. ANNs and pattern recognition are discussed by Pao [4] and Lippman [5]. Our interest lies with the KSO algorithm as it relates to the solution of clustering and classification problems and the HCM/FCM models.

2. CLUSTERING ALGORITHMS AND CLASSIFIER DESIGN

Let (c) be an integer, $1< c < n$ and let $X = \{x_1, x_2, ..., x_n\}$ denote a set of (n) feature vectors in $\Re^s$. X is *numerical object data*; the j-th object (some physical entity such as a medical patient, seismic record etc.) has vector x_j as it's numerical representation; x_{jk} is the k-th characteristic (or *feature*) associated with object j. Given X, we say that (c) fuzzy subsets $\{u_i: X \rightarrow [0,1]\}$ are a fuzzy c-partition of X in case the (cn) values $\{u_{ik} = u_i(x_k),\ 1 \le k \le n,\ 1 \le i \le c\}$ satisfy three conditions:

$$0 \le u_{ik} \le 1 \text{ for all } i,k \qquad (1a)$$

$$\Sigma u_{ik} = 1 \text{ for all } k \quad ; \tag{1b}$$

$$0 < \Sigma u_{ik} < n \text{ for all } i \quad . \tag{1c}$$

Each set of (cn) values satisfying conditions (1) can be arrayed as a (cxn) matrix $U = [u_{ik}]$. The set of all such matrices are the *non-degenerate fuzzy c-partitions* of X:

$$M_{fcn} = \{U \text{ in } \mathfrak{R}^{cn} \mid u_{ik} \text{ satisfies (1) for all i and k}\}. \tag{2}$$

And in case all the u_{ik}'s are either 0 or 1, we have the subset of *hard (or crisp) c-partitions* of X:

$$M_{cn} = \{U \text{ in } M_{fcn} \mid u_{ik} = 0 \text{ or } 1 \text{ for all i and k}\}. \tag{3}$$

The reason these matrices are called partitions follows from the interpretation of u_{ik} as the *membership* of x_k in the i-th partitioning subset (cluster) of X. M_{fcn} is more realistic as a physical model than M_{cn}, for it is common experience that the boundaries between many classes of real objects are in fact very badly delineated (i.e., really fuzzy). The important point is that *all* clustering algorithms generate solutions to the clustering problem for X which are matrices in M_{fcn}. The *clustering problem for X*, is, quite simply, the identification of an "optimal" partition U of X in M_{fcn}; that is, one that groups together object data vectors (and hence the objects they represent) which share some well defined (mathematical) similarity. It is our hope and implicit belief, of course, that an optimal mathematical grouping is in some sense an accurate portrayal of natural groupings in the physical process from whence the object data are derived. The number of clusters (c) must be known, or becomes an integral part of the problem.

3. THE ISODATA AND KSO ALGORITHMS

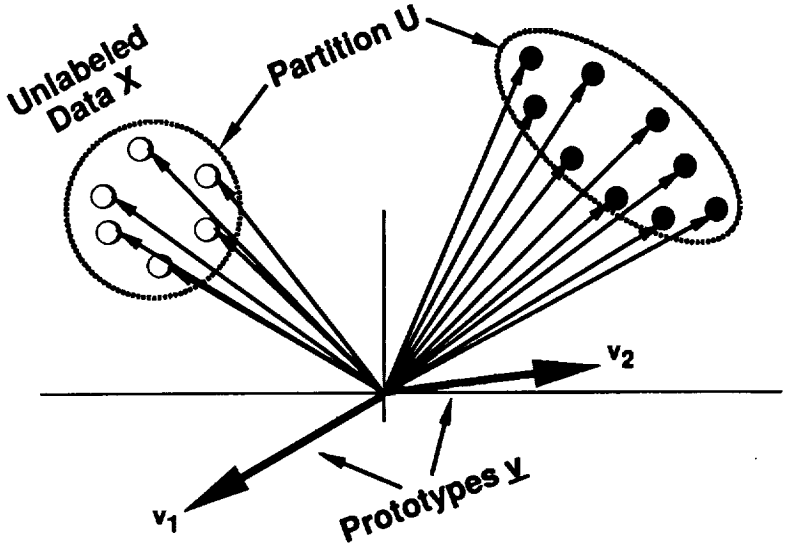
The most well known objective function for clustering is the least total squared error function:

$$J_1(U,\mathbf{v}\,;X) = \Sigma\Sigma\, u_{ik}\,(\|x_k - v_i\|_I)^2 \quad , \tag{4}$$

where $\mathbf{v} = (v_1, v_2, \ldots, v_c)$ is a vector of (unknown) cluster centers (weights or prototypes), $v_i \in \mathfrak{R}^s$ for $1 \le i \le c$, $U \in M_{cn}$ is an unknown hard c-partition of X, and $\|*\|_I$ is the Euclidean norm on $\mathfrak{R}^s$. Optimal partitions U^* of X are taken from pairs (U^*, v^*) that are "local minimizers" of J_1. It is important to recognize the geometric impact that the use of a norm function in J_1 as the criterion of (dis)similarity has on "good clusters" (here $\|*\|_I$, but more generally, any norm on $\mathfrak{R}^s$ induced by a positive definite weight matrix A, as described below). Figure 1 illustrates this graphically; partitions that optimize J_1 will, generally speaking, contain clusters that conform to

the topology that is induced on $\mathfrak{R}^S$ by the eigenstructure of the norm-inducing matrix A. When A = I, good clusters will be hyperspherical, as the one in the left portion of Figure 1; otherwise, they will be hyperelliptical, as the one on the right side of Figure 1.

Figure 1. Geometry of Cluster Formation in Norm-Driven Clustering Algorithms



As is evident in Figure 1, clusters that optimize J_1 are formed on the basis of two properties: *location* and shape. Location information is contained in the lengths of the data vectors and "cluster centers" or prototypes $\{\mathbf{v}_i\}$ from the origin, whilst shape information is embedded in the topology induced by the norm in use. Roughly speaking, these correspond to the mean and variance of probability distributions, so (4) is in some sense analogous to regarding the data as being drawn from a mixture of probability density functions (indeed, there are special cases when (4) yields identical results to the maximum likelihood estimators of the parameters of a mixture of normal distributions). Although the norm shown in (4) is the Euclidean norm, generalizations of J_1 have used all five of the usual norms encountered in numerical analysis and pattern recognition - viz, the Euclidean, Diagonal and Mahalonobis inner product A-norms; and the $p = 1$ and $p = \infty$ (city block and sup) Minkowski norms. The defining equations and unit ball shapes for these two families of norms are shown in Figure 2.

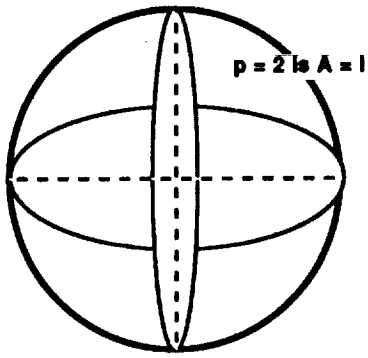
As an explicit means for finding optimal partitions of object data, J_1 was popularized as part of the ISODATA ("Iterative Self-Organizing Data Analysis") algorithm (c-Means + Heuristics) by Ball and Hall [6] in 1967. It is interesting to note that Kohonen apparently first used the term "self-organizing" to describe his approach

about 15 years later [1]. Apparently, the feature of both algorithms that suggests this phrase is their ability to iteratively adjust the weight vectors or prototypes that subsequently represent the data in an orderly and improving manner as the algorithms proceed with iteration. We contend that this use of the term "self-organizing" in the current context of neural network research is somewhat misleading (in both cases). Indeed, if the aspect of FCM/HCM and KSO that entitles us to call them self-organizing is their ability to adjust their parameters during "training", then every iterative method that produces approximations from data is self-organizing (e.g., Newton's method!). On the other hand, if this term serves to indicate that the algorithms in question can find meaningful labels for objects, without external interference (labelled) training examples), then all clustering algorithms are "self-organizing". Since the terminology in both cases is well established, the only expectation this writer has about the efficacy of these remarks is that they caution readers take the semantics associated with much of the current Neural Network literature with a large grain of salt.

Figure 2. Geometry of Level Sets for Inner product A-norms and Minkowski p-norms

Unit Ball Shapes in the A - norms

$$L_A = \{x : \langle x,x\rangle_A = x^T A x = (\|x\|_A)^2 = 1\}$$



$$\| x_k - v_i \|_A = ((x_k - v_i)^T A (x_k - v_i))^{(1/2)}$$

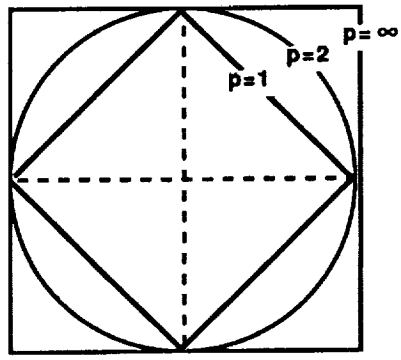
EV's of pos-definite (A) Induce shapes

Inner product : Hilbert Space Structure

Differentiable in All Variables

Unit Ball Shapes in the p - norms

$$L_p = \{x : \|x\|_p = 1\}$$



$$\| x_k - v_i \|_p = (\Sigma\ |x_{kj} - v_{ij}|^p)^{(1/p)}$$

$$\| x_k - v_i \|_1 = (\Sigma\ |x_{kj} - v_{ij}|)$$

$$\| x_k - v_i \|_\infty = (\max_j \{|x_{kj} - v_{ij}|\})$$

p = 2 : Hilbert ; p ≠ 2: Banach Spaces

Dunn [7] first generalized J_1 by allowing U to be fuzzy (m=2 below) and the norm to be an arbitrary inner product A-norm. Bezdek [8] generalized Dunn's functional to the fuzzy ISODATA family written as:

$$J_m(U,v;X) = \Sigma\Sigma u_{ik}^m (||x_k - v_i||_A)^2, \tag{5}$$

where $m \in [1, \infty)$ is a weighting exponent on each fuzzy membership; $U \in M_{fcn}$ is a fuzzy c-partition of X; $v = (v_1, v_2, ..., v_c)$ are cluster centers in $\Re^s$; A = is any positive definite (s x s) matrix; and $(||x_k - v_i||_A)^2 = (x_k - v_i)^T A (x_k - v_i)$ is the OG distance (in the A norm) from x_k to v_i .

In 1979, Gustafson and Kessel [8] derived necessary conditions to minimize an extension of (5) with (c) different norm inducing matrices. In 1981 Bezdek et. al. [9] generalized (5) by allowing the prototypes to be (convex combinations of) linear manifolds of arbitrary and different dimensions. In 1985 Pedrycz [10] introduced a way to use partially labeled data with (5) that amounts to a mixed supervised-unsupervised clustering scheme. In 1989 Dave [11] introduced a generalization of (5) that uses *hyperspherical* prototypes for v. In 1990 Bobrowski and Bezdek [12] used the city block and sup norms with (5), thus extending the c-Means algorithms to the most important Minkowski norms (p = 1 and p = ∞).

Necessary conditions that define iterative algorithms for (approximately) minimizing J_m and its generalizations are known. Our interest lies with the cases represented by (4) and (5). The conditions that are necessary for minima of J_1 and J_m follow :

Hard c-Means (HCM) Theorem [2]. (U, v) may minimize $\Sigma\Sigma\, u_{ik}(||x_k - v_i||_A)^2$ only if

$$u_{ik} = \quad 1; \ (||x_k - v_i||_A)^2 = \min_j\{(||x_k - v_j||_A)^2\}; \quad \text{and} = 0; \ \text{otherwise} \tag{6a}$$

$$v_i = \Sigma u_{ik} x_k / \Sigma u_{ik} \tag{6b}$$

Note that HCM produces hard clusters $U \in M_{cn}$. The HCM conditions are necessary for "minima"of (4) (i.e., with A=I, the Euclidean norm on $\Re^s$), and, as we shall note, are also used to derive hard clusters in the KSO algorithm. The well known generalization of the HCM conditions is contained in the:

Fuzzy c-Means (FCM) Theorem [2]. (U,v) may minimize $\Sigma\Sigma\, u_{ik}^m (||x_k - v_i||_A)^2$ for m > 1 only if :

$$u_{ik} = (\Sigma(||x_k - v_i||_A / ||x_k - v_j||_A)^{2/(m-1)})^{-1} \tag{7a}$$

$$v_i = \Sigma (u_{ik})^m x_k / \Sigma (u_{ik})^m \qquad (7b)$$

The FCM conditions are necessary for minima of (5). There is an alternative equation for (7a) if one or more of the denominators in (7a) is zero. These equations converge to the HCM equations as $m \to 1$ from above, and for ($m > 1$), the U in FCM is truly fuzzy, i.e., $U \in (M_{fcn} - M_{cn})$. The FCM algorithms are simple Picard iteration through the paired variables U and v. Because we want to compare this method to the KSO algorithm, we give a brief description of the FCM/HCM algorithms.

(Parallel) c-Means (FCM/HCM) Algorithms

<FCM/HCM 1> : Given unlabeled data set $X = \{x_1, x_2, ..., x_n\}$. Fix : $1 < c < n$; $1 \le m < \infty$ (m=1 for HCM); positive definite weight matrix A to induce an inner product norm on $\Re^s$; and ϵ, a small positive constant.

<FCM/HCM 2>: Guess $v_0 = (v_{1,0}, v_{2,0}, ..., v_{c,0}) \in \Re^{cs}$ (or, initialize $U_0 \in M_{fcn}$).

<FCM/HCM 3>: For j = 1 to J:

<3a> : Calculate U_j with $\{v_{i,j-1}\}$;

<3b>: Update $\{v_{i,j-1}\}$ to $\{v_{i,j}\}$ with U_j ;

<3c>: If $\max_i \{ \|v_{i,j-1} \text{ to } v_{i,j}\| \} \le \epsilon$, then stop and put $(U^*,v^*) = (U_j,v_j)$; Else : Next j

This procedure is known to converge q-linearly from any initialization to a local minimum or saddle point (U^*,v^*) of J_m. Note again that the update rule for the weights $\{v_i\}$ at step <3b> is a necessary condition for minimizing J_m. Moreover, all (c) weight vectors are updated using all (n) data points simultaneously at each pass; i.e., the weights $\{v_i\}$ are not sequentially updated as each x_k is processed. This is why we call the above description a "parallel" version of c-means, as opposed to the well known sequential version.

There is a *sequential* version of hard c-means (SHCM) that can be used to minimize J_1, and readers should be aware that it may produce quite different results than HCM on the same data set. One iteration of the SHCM algorithm is as follows: beginning with some hard U, the centers $\{v_i\}$ are calculated with (6b). Once the prototypes are known, one returns to update U. Beginning with x_1, each point is examined, and moved from,

say, cluster i to cluster j, so as to maximize the decrease in J_1 (if possible) . Then the two affected centers (v_i , v_j} and rows i and j of U are updated using equations (6) . One complete pass of SHCM consists of testing each of the n data points in X, and effecting a transfer at each point where a decrease in J_1 can be realized. SHCM terminates when a complete pass can be made without transfers. We mention this version of HCM because it is SHCM that most closely resembles the KSO algorithm. Figure 3 is a rough depiction of how the HCM method might begin; Figure 4 indicates a desirable situation at termination. In Figure 3 the initial hard clusters subdivide the data badly, and the overall mean squared error (the sum of squares of the solid line distances between data points and prototypes) is large; at termination, the prototypes lie "centered" in their clusters, the overall sum of squared errors is low, and the hard 2-partition subdivides the data "correctly" (this is what happens if we are lucky !).

Figure 3. An Initial 2-Partition and Prototypes for HCM

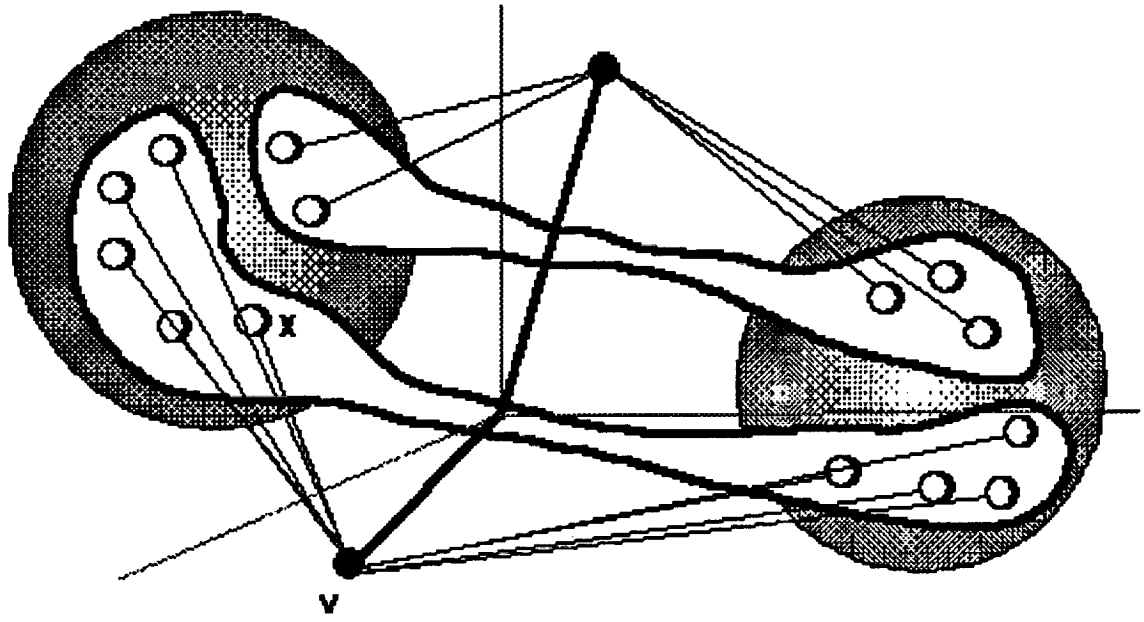
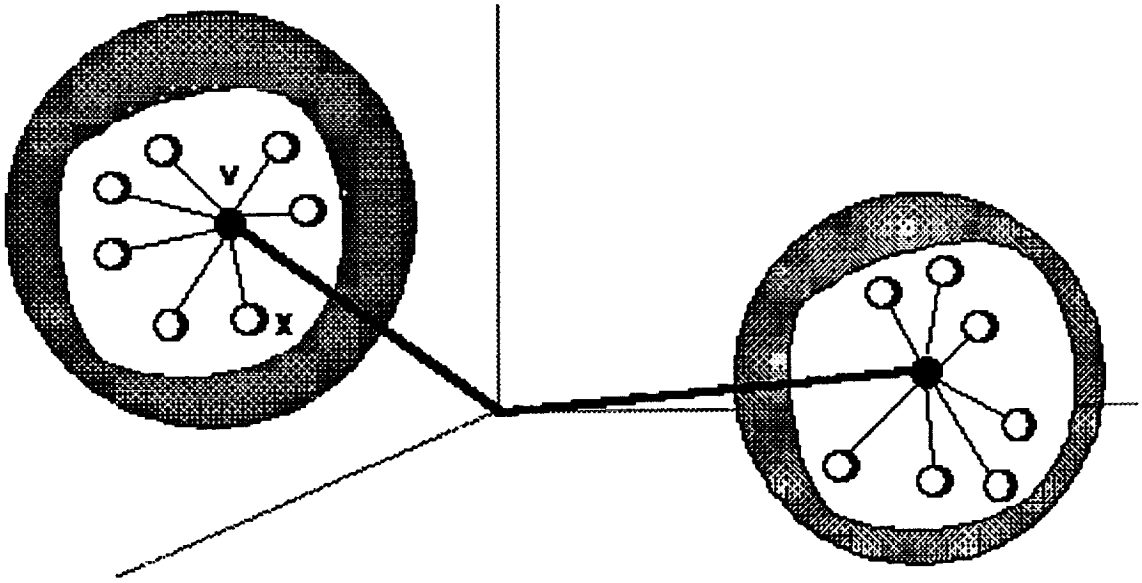


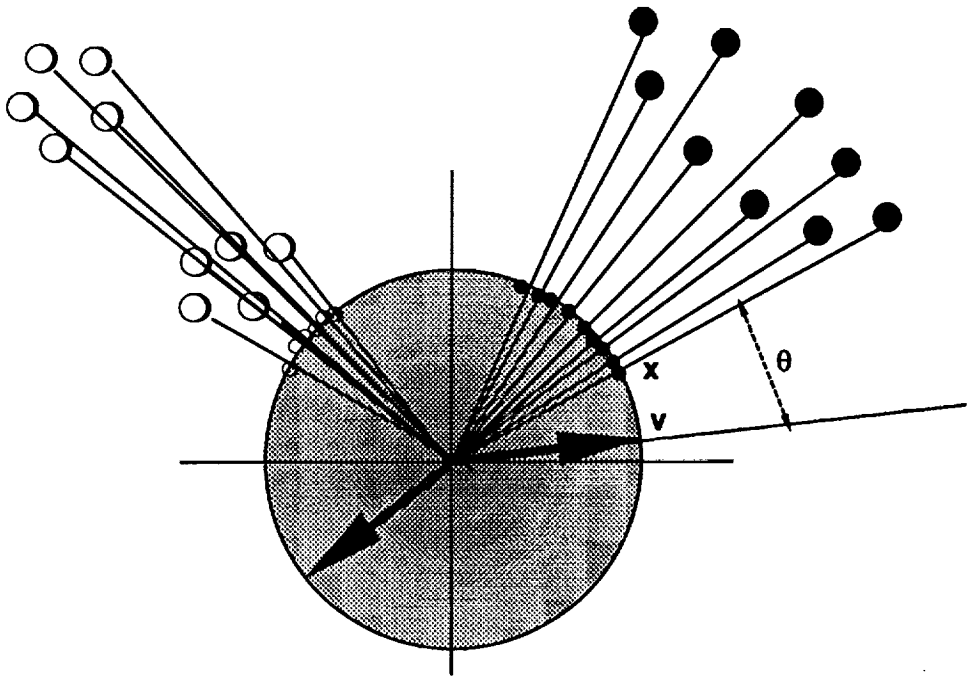
Figure 4. A (Benevolent) Final Configuration of 2-Partition and Prototypes for HCM



Kohonen's method differs from the c-means approach in several important ways. First, it is not a norm-driven scheme. Instead, the KSO method uses the geometric notion of *orientation matching*, depicted in Figure 5, as the basic measure of similarity between data points and cluster centers. Second, there is no partition U involved in the KSO algorithm. Instead, an initial set of cluster centers are iteratively updated without reference to partitions of the unlabeled data. The underlying geometry of the criterion of similarity is shown in Figure 5.

The measure of similarity, as shown in Figure 5, is the angle between a data point **x** and prototype **v** (in the neural network community, the vectors $\{\mathbf{v}_i\}$ are often called "weight" vectors; each one being attached or identified with a "node" in the network). Information that the data set may contain about cluster shapes in feature space is lost (i.e., not used by $\cos(\theta)$); and if the data are normalized at each step to be vectors of length 1, as they usually are in the KSO approach, location information is lost as well. Consequently, the geometry favored by the KSO criterion of similarity is data substructures that lie in *angular cones* emanating from the origin. We emphasize that in real data, either type of criterion - the c-means type norm driven measure, or the KSO angular measure - may or may not be appropriate for matching the data. As with all clustering problems, the question is not - which is better? the question is, which is better for this data set? In order to effect comparison with the c-means model, a brief description of Kohonen's algorithm follows.

Figure 5. Geometry of Cluster Formation in Orientation Matching Clustering Algorithms



$$\cos(\theta) = \langle x,v \rangle = 1 - \|x-v\|^2 / 2$$

Kohonen's (KSO) Clustering Algorithm

<KSO1> : Given unlabeled , "ordered" data set $X = \{x_1, x_2, ..., x_n\}$. Fix : $1 < c^*$; Choose update scale factors $\{\alpha_j\}$ so that $\{\alpha_j\} \to 0$; $\Sigma\alpha_j = \infty$, $\Sigma(\alpha_j)^2 < \infty$; Choose update neighborhood "radii" $\{\rho_j\} \in \{0,1,2,..., c^*\}$:

<KSO2> Guess (unit vectors) $v_0 = (v_{1,0}, v_{2,0}, ..., v_{c^*,0}) \in \Re^{c^*s}$

<KSO3> For j = 1 to J: : For k = 1 to n:

<3a> : Find i*(k) st $(\|x_k - v_{i^*(k)}\|_I)^2 = \min\{(\|x_k - v_{j(k)}\|_I)^2$

<3b>: For indices $N^*(k) = i^*(k), i^*(k) \pm 1, \ldots i^*(k) \pm \rho_j$ Update $v_{t,j-1}$:

$$v_{t,j} = v_{t,j-1} + \alpha_j (x_k - v_t) / \| v_{t,j-1} + \alpha_j (x_k - v_t) \|_I \text{; otherwise, } v_{t,j} = v_{t,j-1} . \quad (8)$$

Next k; Next j

We have used c^* instead of c in this procedure to emphasize the fact that Kohonen's method often uses "multiple" prototypes, in the sense that even though (unbeknownst to us !) X contains only c clusters, it may be advantageous to look for $c^* > c$ cluster centers; this is a further difference between the c-means and KSO strategies. This is one form of Kohonen's approach; other update rules have been used. The geometry of the update rule for the weight vectors in (8) is depicted in Figure 6. Thus, if we are at point x_k, as shown in Figure 6, <3a> of the KSO algorithm simply finds the current prototype (v_{old}) closest to x_k in angle (minimizing the angle is equivalent to the formula in <3a>). If the current center is called $v_{old} = v_{i^*(k)}$ as in Figure 6, then update equation (8) connects $v_{old} = v_{i^*(k)}$ to the vector x_k, rotates v_{old} to the new position v_{new}, and finally normalizes v_{new}.

The KSO procedure is exactly like SHCM in that it updates (some subset of the) prototypes sequentially after the examination of each data point. Figure 7 indicates the geometry of the scheme specified in <3b>; the basic idea is that once the prototype v_{old} closest to the current data point is found, all prototypes in a neighborhood of the "winner" are also updated.

Figure 6. The Geometry of Kohonen's Updating Rule

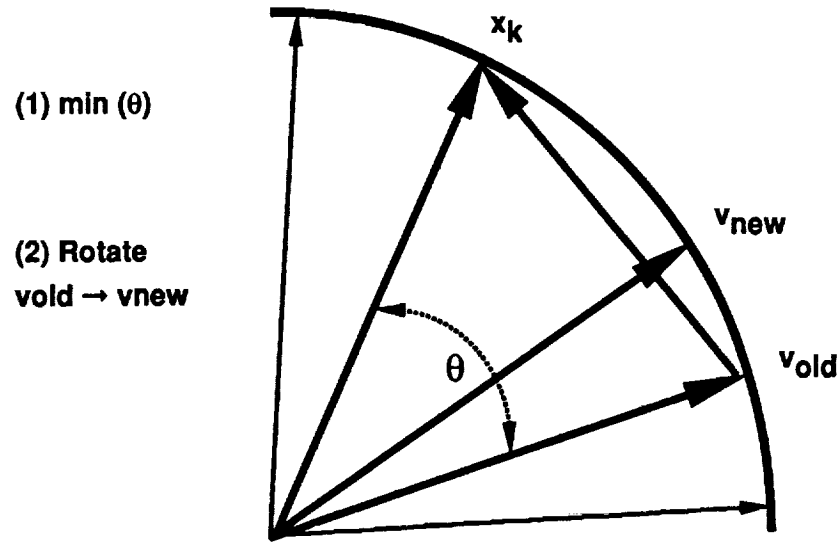
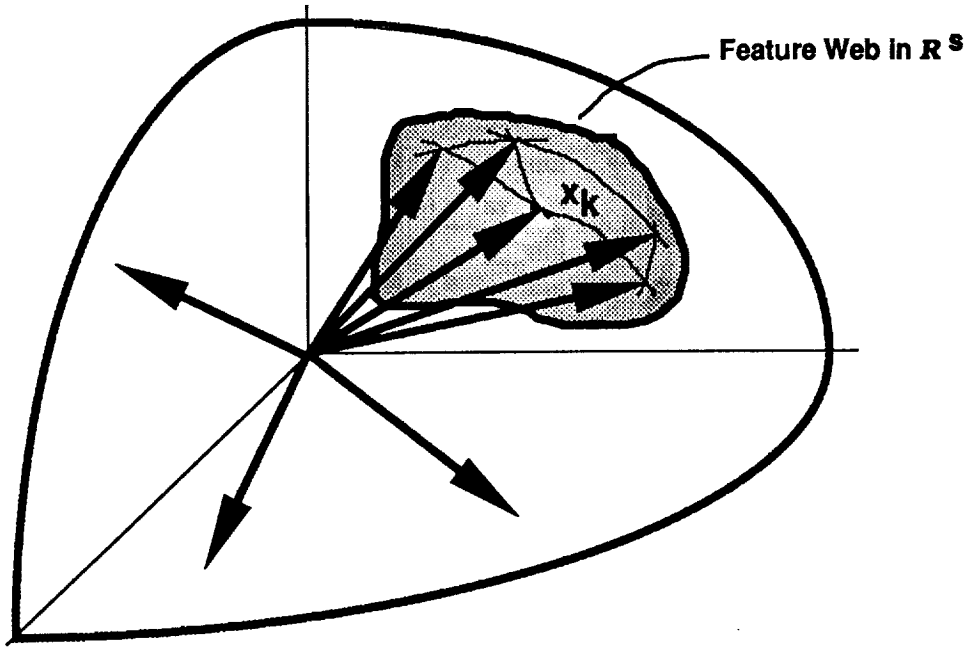


Figure 7. KSO Updating of Prototypes in the Neighborhood N*(k) of "Winner" $v_{l^*(k)}$



Although the "feature web" shown in Figure 7 is conceptualized here as being in $\mathcal{R}^S$, it has actually been displayed only the case s = 2. Kohonen has shown that this process converges, in the sense that the $\{v_{t,j}\} \rightarrow \{v_t^*\}$ as $\{\alpha_j\} \rightarrow 0$, in the special case s=2. Moreover, the limiting $\{v_t^*\}$ preserve a "topological ordering" property of the data set X on an array of output nodes associated to the weight vectors. Iteration in the KSO method thus trains the weight vectors $\{v_t^*\}$ so that they preserve "order" in the output nodes. As previously noted, the KSO method does not use or generate a partition U of the data during training. However, once the weight vectors stabilize, the KSO model produces a hard U by following the nearest prototype rule below.

More specifically, once a set of prototypes $\{v_i\}$ are found by "training" on some data set X (this includes all four methods described above, HCM, FCM SHCM and KSO), they can be used to label *any* unlabeled data set. For *any* vector $\mathbf{x} \in \mathcal{R}^S$, the HCM equation for u_{ik} defines a (piecewise linear) *nearest prototype classifier:*

The Nearest Prototype Classifier Decision Rule : Given $\{v_i\}$, Compute, *non-iteratively*, the hard c-partition of (*any*) data X with HCM equation (6a):

$$u_{ik} = \begin{cases} 1; & ((\|x_k - v_i\|_I))^2 = \min_j\{(\|x_k - v_j\|_I)^2\} \\ 0; & \text{otherwise} \end{cases} \qquad (9)$$

Note that we have written (9) with the Euclidean norm. Theorem 2 suggests that any scalar product induced A-norm might be used in the formula; however, interpretation of the subsequent decision rule as discussed above becomes very difficult. Thus, while it makes sense geometrically to consider variations in the norm as in (7) while *searching* for the cluster centers, it is much less clear that norms other than the Euclidean norm should be used during *classification*. Figure 8 is a rough depiction of how the KSO method might begin; Figure 9 shows the situation after termination of KSO, followed by *a posteriori* application of (9) to find an "optimal" hard c-partition U corresponding to the final weights . A question about how rule (9) is used with the KSO prototypes remains: how do we, without labeled data, assign one of $c < c^*$ "real" labels to subsets of the c^* weight vectors found by the KSO scheme? The same question applies to FCM - we still need to decide which of the c "real" labels belongs to each prototype - the problem is just more pronounced when there are multiple prototypes for each class.

Figure 8. Initial Configuration of Weight Vectors in the KSO Scheme

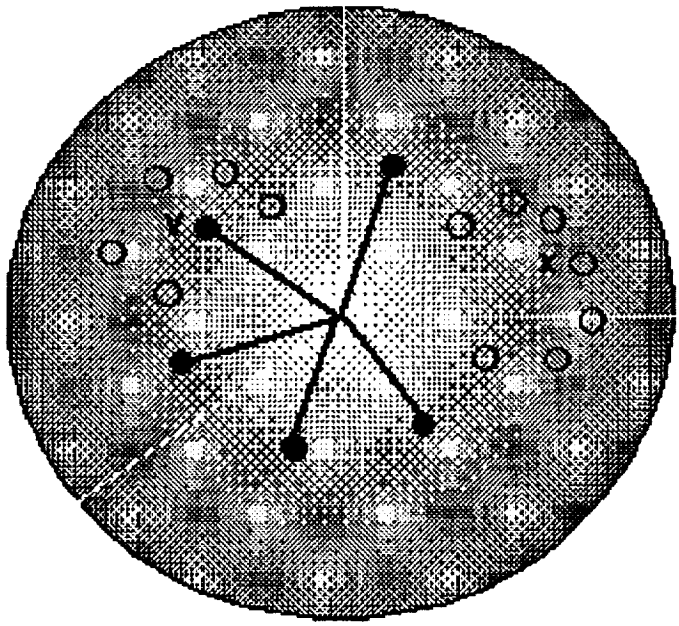
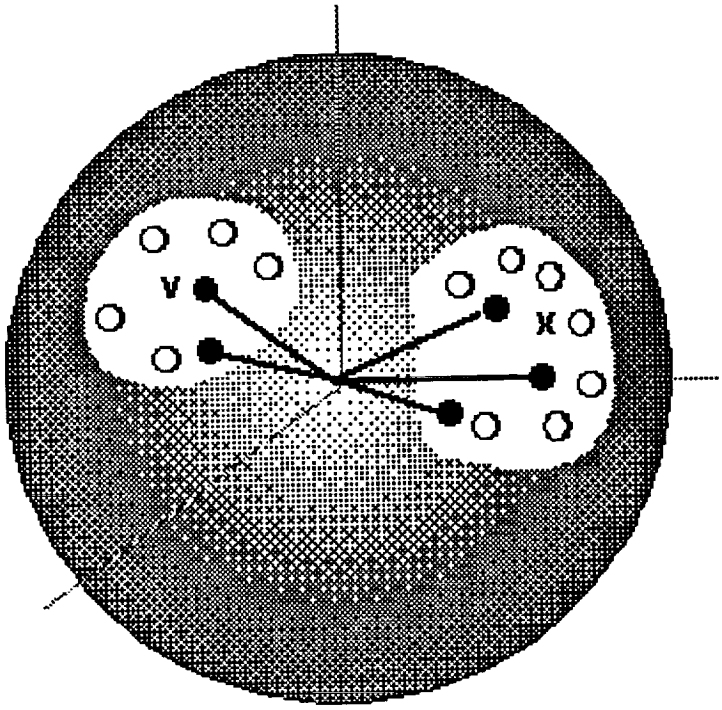


Figure 9. Terminal Weight Vectors and an HCM Partition in the KSO Scheme



4. DISCUSSION AND CONCLUSIONS

First, we itemize the major *differences* between ISODATA and KSO :

(D1) FCM , HCM and SHCM are *intrinsic* clustering methods - i.e., one of their inputs is an unknown partition, and one of their outputs is a partition of unlabeled data set X which is optimal in the sense of minimizing a norm driven objective function. The KSO method, on the other hand, needs an *a posteriori* rule such as the nearest prototype rule at (9) to generate a partition of the data non-iteratively. We might call this an *extrinsic* clustering scheme. Moreover, without labeled data that can be used to discover which subsets of the c^* multiple prototypes found by the KSO scheme should be identified with each of the c classes assumed in (9), there is no general way to even implement (9) with the KSO rule. Thus, much must be added to KSO to make it a true clustering method.

(D2) The data set X is used differently. KSO uses the data sequentially (locally) and hence, its outputs are dependent on local geometry and order of labels, whereas ISODATA utilizes the data globally, and

updates both the weights and partition values in parallel at each pass. In this sense KSO is most akin to Sequential Hard c-Means, which is also sensitive to ordering of labels - this is often regarded as a fatal flaw in clustering.

(D3) KSO can have multiple prototypes for each class; ISODATA has but one. In clustering, the usual assumption is that c is unknown, and one resorts to various cluster validity schemes to validate the results of any algorithm. Since the KSO scheme uses many prototypes, without assuming an underlying "true but unknown" number of clusters, this is advantageous to the user. However, the dilemma of how to convert the prototypes into clusters, as discussed in (D1), persists.

(D4) KSO uses local orientation ($\cos\theta = \langle \mathbf{x}, \mathbf{v} \rangle$) on the unit ball as the measure of similarity between data and weights, whereas ISODATA uses cluster shape (via the eigenstructure of A) and location (via the lengths of the weights and the data) to assess (dis)similarity between the data and prototypes. Thus, the c-Means approach has a much more "statistical" flavor than KSO. On the other hand, KSO uses the dot product at each node, in the spirit of the McCulloch-Pitts neuron. Thus, local computations in the KSO scheme proceed on the basis assumed by many workers in neural network research, and make the KSO scheme more easily identifiable with this type of computational architecture.

(D5) KSO preserves "order" in a certain sense; ISODATA does not. This property of the KSO method is perhaps its most interesting distinction. There is little hope that c-Means has a similar property. Since cognitive science assures us that one aspect of intelligence is its inherent ability to order, this aspect of the KSO approach again shows well in its favor. A significant line of research concerns whether or not the FCM/HCM models possess this, or any similar property.

(D6) Weight updates in the KSO method are intuitively appealing; weight updates in ISODATA are mathematically necessary. Since the update formula in c-Means finds either real or generalized centroids, we might claim that this scheme is also intuitively appealing. In this regard the c-Means algorithms (including SHCM) have a clear theoretical advantage, at least in terms of justification of the procedure used.

(D7) FCM, HCM and SHCM are all well-defined optimization problems; KSO is an heuristic procedure. An interesting question about KSO is this: what function is being optimized during iteration? An answer to this question would be both useful and illuminating. The criterion functions that drive FCM, HCM and SHCM are well understood geometrically and statistically; discovery of a criterion function for Kohonen's algorithm might supply a great deal of insight about other properties of the algorithm and its outputs.

(D8) KSO partitions have so far been generated with the nearest prototype rule and the Euclidean norm, whereas FCM, HCM and SHCM can be used with any inner product and two Minkowski norms. Much research can be done on the issue of how best to use the Kohonen prototypes to find cluster substructure. There are many natural ways besides the nearest prototype rule to use KSO outputs with the weights $\{\mathbf{v}_i\}$. For example, one could simply distribute unit memberships satisfying (1b) across the KSO nodes at each step using distance proportions. This generalizes Kohonen's model from a "neighborhood take all" to a "neighborhood share all" concept. One certainly suspects that it is possible to incorporate $U \in M_{fcn}$ as an *unknown* in the KSO approach , so that an extended KSO algorithm creates partitions of the data that are necessary, rather than, as in the current use of the HCM labeling rule, a heuristic afterthought.

Major *similarities* between ISODATA and KSO include:

(S1) If we let $(U_F, \mathbf{v}_F)$, $(U_H, \mathbf{v}_H)$, $(U_S, \mathbf{v}_S)$, and $(U_K, \mathbf{v}_K)$ denote, respectively, the pairs found by FCM, HCM, SHCM and KSO, we note that $(U_F, \mathbf{v}_F)$ is a critical point for J_m, while $(U_H, \mathbf{v}_H)$, $(U_S, \mathbf{v}_S)$, and $(U_K, \mathbf{v}_K)$ are, because of the HCM theorem, (possibly different) critical points of J_1. However, $(U_H, \mathbf{v}_H) \neq (U_S, \mathbf{v}_S) \neq (U_K, \mathbf{v}_K)$ generally. This suggests that (i) HCM (and especially SHCM) and KSO as described herein are most definitely related, and (ii), there should be a generalized (fuzzy) KSO that bears the same relationship to FCM that the hard c-Means versions bear to the current version of KSO. It seems clear that there is a stronger mathematical link between FCM/HCM and KSO than is currently known. Connection of the two approaches begins with careful formulation of a constrained optimization problem that holds for KSO. This involves finding a global KSO criterion function and necessary conditions that *require* the calculation of the weight vectors $\{\mathbf{v}_i\}$ as in KSO <3b>.

(S2) Both algorithms find prototypes (weights or cluster centers) in the data that provide a compressed representation of it, and enable nearest prototype classifier design. Recent work by Huntsberger and Ajjimarangsee [13] indicates that FCM is at least as good as KSO in terms of minimizing apparent error rates. And further, FCM sometimes generates identical solutions to KSO on various well known data sets. This is another powerful indicator of the underlying (unknown) relationship between the KSO and c-Means methods. Much can be done empirically to confirm or deny specific relationships between the two methods.

We have itemized some similarities and differences between two approaches to the clustering of unlabeled data - Hard/Fuzzy c-Means and Kohonen's self-organizing feature maps (KSO), and posed some questions concerning each method. Successful resolution of these questions will benefit both models. Numerical convergence properties and the neural-like behavior of both the extended KSO and FCM algorithms should

be established. Issues to be studied should include : robustness, adaptivity , parallelism , apparent error rates, time and space complexity, type and rate of convergence, optimality tests, and initialization sensitivity.

5. REFERENCES

[1] Kohonen, T. Self-Organization and Associative Memory, 3rd Edition, Springer-Verlag, Berlin, 1989.

[2] Bezdek, J. Pattern Recognition with Fuzzy Objective Function Algorithms, Plenum, New York, 1981.

[3] Duda, R. and Hart, P. Pattern Classification and Scene Analysis, Wiley, New York, 1973.

[4] Pao, Y.H. Adaptive Pattern Recognition and Neural Networks, Addison-Wesley, Reading, 1989.

[5] Lippman, R. An Introduction to Neural Computing, *IEEE ASSP Magazine*, April, 1987, 4-22.

[6] Ball, G. and Hall, D. A Technique for Summarizing Multivariate Data, Behav. Sci., 12, 1967, 153-155.

[7] Dunn, J.C. A Fuzzy Relative of the ISODATA Process, *Jo. Cybernetics*, 3, 1974, 32-57.

[8] Gustafson, D. and Kessel, W. Fuzzy Clustering with a Fuzzy Covariance Matrix, in Proc. IEEE CDC, 1978, 761-766.

[9] Bezdek, J. C., Coray, C., Gunderson, R. and Watson, J. Detection and Characterization of Cluster Substructure, I and II, *SIAM Jo. of Appl. Math.*, 40(2), 1981, 339-372.

[10] Pedrycz, W. Algorithms of Fuzzy Clustering with Partial Supervision, *Patt. Recog. Letters*, 3, 1985, 13-20.

[11] Dave, R. Fuzzy Shell Clustering and Applications to Circle Detection in Digital Images, in press, *Int'l. Jo. of General Systems*, 1989.

[12] Bobrowski, L. and Bezdek, J. c-Means Clustering with the ℓ_1 and ℓ_∞ Norms, in review, IEEE Trans. SMC, 1990.

[13] Huntsberger, T. and Ajjimarangsee, P. Parallel Self-Organizing Feature Maps for Unsupervised Pattern Recognition, in press, *Int'l. Jo. General Systems*, 1990.

N91-21784

Single Board System for Fuzzy Inference

James R. Symon Hiroyuki Watanabe
Department of Computer Science
CB# 3175 Sitterson Hall
University of North Carolina
Chapel Hill, NC 27514-3175
TEL. (919) 962-1817, 962-1893

Abstract

The VLSI implementation of a fuzzy logic inference mechanism allows the use of rule-based control and decision making in demanding real-time applications such as robot control and in the area of command and control. We have designed a full custom VLSI inference engine. The chip is fabricated using 1.0 μ CMOS technology. The chip consists of 688,000 transistors of which 476,000 are used for RAM memory.

The fuzzy logic inference engine board system incorporates the custom designed integrated circuit into a standard VMEbus environment. The Fuzzy Logic system board uses TTL logic parts to provide the interface between the Fuzzy chip and a standard, double height VMEbus backplane allowing the chip to perform application process control through the VMEbus host. High level C language functions hide details of the hardware system interface from the applications level programmer. The first version of the board was installed on a robot at Oak Ridge National Laboratory in January of 1990.

1 Introduction

Fuzzy logic based control uses a rule-based expert system paradigm in the area of real-time process control [4]. It has been used successfully in numerous areas including train control [12], cement kiln control [2], robot navigation [6], and auto-focus camera [5]. In order to use this paradigm of a fuzzy rule-based controller in demanding real-time applications, the VLSI implementation of the inference mechanism has been an active research topic [1, 11]. Potential applications of such a VLSI inference processor include real-time decision-making in the area of command and control [3], and control of precision machinery.

An original prototype experimental chip designed at AT&T Bell Labs [7] was the precursor to the fuzzy logic inference engine IC that is the heart of our hardware system. The current chip was designed at the University of North Carolina in cooperation with engineers at the Microelectronics Center of North Carolina (MCNC) [8]. MCNC fabricated and tested fully functional chips.

The new architecture of the inference processor has the following important improvements compared

to previous work:

1. programmable rule set memory
2. on-chip fuzzifying operation by table lookup
3. on-chip defuzzifying operation by centroid algorithm
4. reconfigurable architecture
5. RAM redundancy for higher yield

The fuzzy chips are now incorporated in VMEbus circuit boards. One of the boards was designed for NASA Ames Research Center and another board was designed for Oak Ridge National Laboratory (ORNL). The latter board has been installed and is currently performing navigational tasks on experimental autonomous robots [9].

ORNL will soon receive the second version of the board system featuring seven Fuzzy chips in a software reconfigurable interconnection network. The network provides host and inter-chip I/O in any logical configuration of the seven chips.

2 Fuzzy Inference

The inference mechanism implemented is based on the compositional rule of inference for approximate reasoning proposed by Zadeh [13]. Suppose we have two rules with two fuzzy clauses in the IF-part and one clause in the THEN-part:

Rule 1: If (x is A_1) and (y is B_1) then (z is C_1),
Rule 2: If (x is A_2) and (y is B_2) then (z is C_2).

We can combine the inference of the multiple rules by assuming the rules are connected by OR connective, that is Rule 1 OR Rule 2 [7]. Given fuzzy proposition (x is A') and (y is B'), weights α_i^A and α_i^B of clauses of premises are calculated by :

$$\alpha_i^A = \max_x(A', A_i),$$
$$\alpha_i^B = \max_y(B', B_i), \quad for \quad i = 1, 2.$$

Then, weights w_1 and w_2 of the premises are calculated by :

$$w_1 = \min(\alpha_1^A, \alpha_1^B),$$
$$w_2 = \min(\alpha_2^A, \alpha_2^B),$$

Weight α_i^A represents the closeness of proposition (x is A_i) and proposition (x is A'). Weight w_i represents similar measure for the entire premise for the i^{th} rule. The conclusion of each rule is

$$C_i' = \min(w_i, C_i), \quad for\ i = 1, 2.$$

The overall conclusion C' is obtained by

$$C' = \max(C_1', C_2').$$

This inference process is shown in Figure 1. In this example, $\alpha_1^A = 0.5$ and $\alpha_1^B = 0.25$, therefore $w_1 = 0.25$. $\alpha_2^A = 0.85$ and $\alpha_2^B = 0.5$, therefore $w_2 = 0.5$.

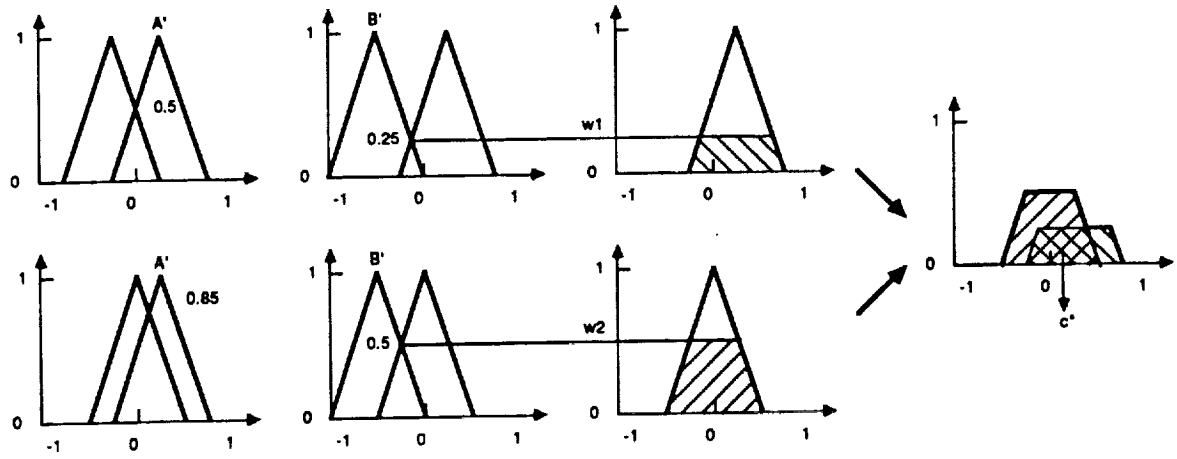


Figure 1: Inference.

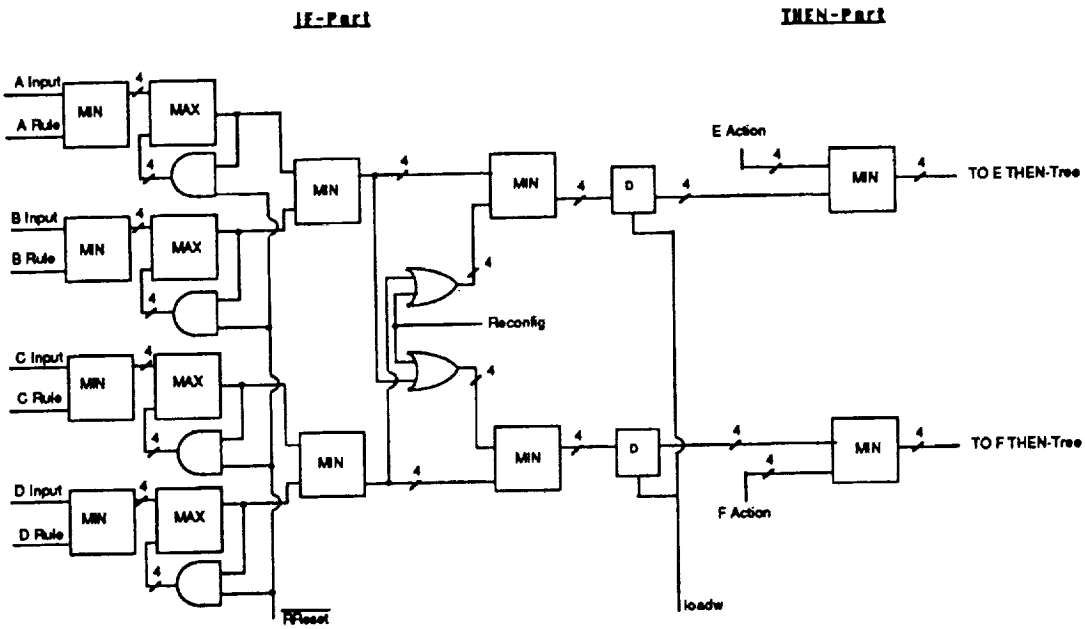


Figure 2: Fuzzy Chip Datapath.

3 Fuzzy Chip

The fuzzy logic inference engine is a fully custom designed 1.0 micron CMOS VLSI circuit of 688,000 transistors implementing a fuzzy logic based rule system. Included on chip are a programmable rule set memory, an optional input fuzzifying operation by table lookup, a minimax paradigm fuzzy inference processor, and an optional output defuzzifying operation using a centroid algorithm. The standard data path configuration is shown in Figure 2. The design has a reconfigurable architecture implementing either 50 rules with 4 inputs and 2 outputs, or 100 rules with 2 inputs and 1 output. Separately addressed status registers allow programmed control of the fuzzy inference processing and chip configuration. All the rules operate in parallel generating new outputs over 150,000 times per second.

The chip has 12 bidirectional data pins and 7 address pins for rule memory I/O. For process-control I/O, each of 4 inputs and 2 outputs has 6 pins. Each of 4 inputs has a corresponding load pin. The chip also has several control signals. Control signals **RW**(read high write low) and **CEN** (chip enable) are similar to that of a memory chip.

4 The System Boards

4.1 Single Chip Systems

The Fuzzy Logic system boards place the Fuzzy chip into a VMEbus environment to provide application process control through a VMEbus host. The single chip system designed for NASA Ames Research Center uses an off-the-self VMEbus prototyping board [10]. The overall configuration of the design is shown in Figure 3. In this design, the VMEbus interface is provided by the prototyping board system and needed a minimum of design for integration of the fuzzy chip. The fuzzy chip interface to the board is realized using discrete TTL parts and wire-wrapping. In the board system for ORNL, the VMEbus interface was designed by the first author and realized using a programmable logic device (PLD) and TTL parts. More robust printed circuit board (PCB) technology was used. The PCB architectural concept is shown in Figure 4. The UNIX device driver interfaces of these two boards are quite similar.

The ORNL board is designed to standard VMEbus specifications for a 24 bit address, 16 bit data, slave module as found in *The VMEbus Specification*, Revision C.1, 1985. It provides digital communication between the host and the Fuzzy chip. A large, UV erasable PLD generates the board control signals. VMEbus interface is through TTL parts. One Fuzzy Inference IC processes four 6-bit inputs to generate two 6-bit outputs. The interface with the host computer uses memory mapping to include the Fuzzy chip's I/O addresses in the application process storage space. All of the chip's memory as well as its inputs and outputs are accessed through addresses on the VMEbus so that the entire Fuzzy Logic board system responds like a section of memory.

The board's address space is 1024 bytes or 512 16-bit words in length. Most of the addresses in that space are not used by the board. The lower 128 word addresses of the board are mapped into the fuzzy chip. One hundred addresses are for rule memory. Another six addresses are mapped to four *fuzzification* tables and two status registers. The board has six addresses for I/O for the fuzzy chip, and addresses for hardware reset and board ID. On-board dip switches and signal jumpers allow the user to select the board base address comprised of the upper 14 bits of the 24 bit address, and the board's user privilege response characteristic determined by the VMEbus *address modifier* bits. Further design details are shown in Figure 5.

4.2 Multiple Fuzzy Chip System

The second version of the system board keeps the standard VMEbus interface of the first version but adds significant new capabilities. Seven Fuzzy chips communicate with each other and the host through a software reconfigurable interconnection network. Two Texas Instruments digital crossbar switch IC's

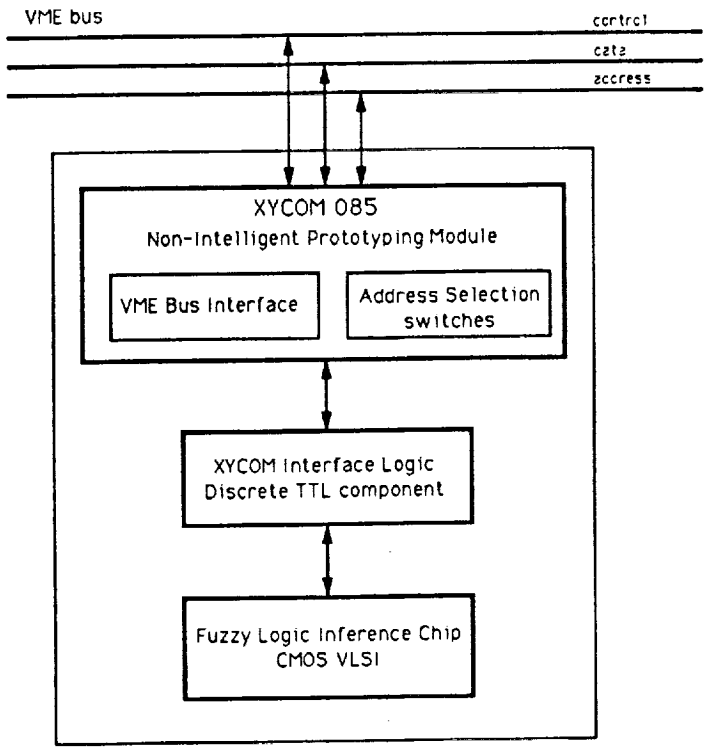


Figure 3: Single Chip System Based On Prototyping Board.

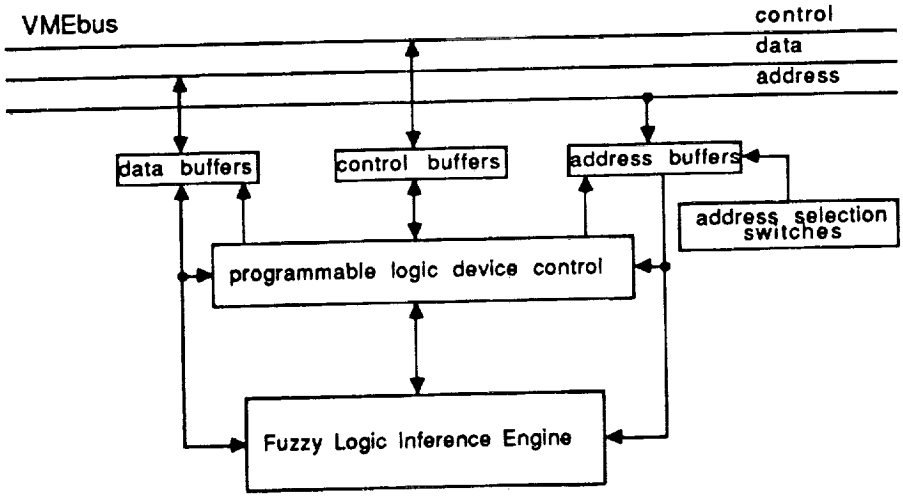


Figure 4: Single Chip System Based On Custom PCB.

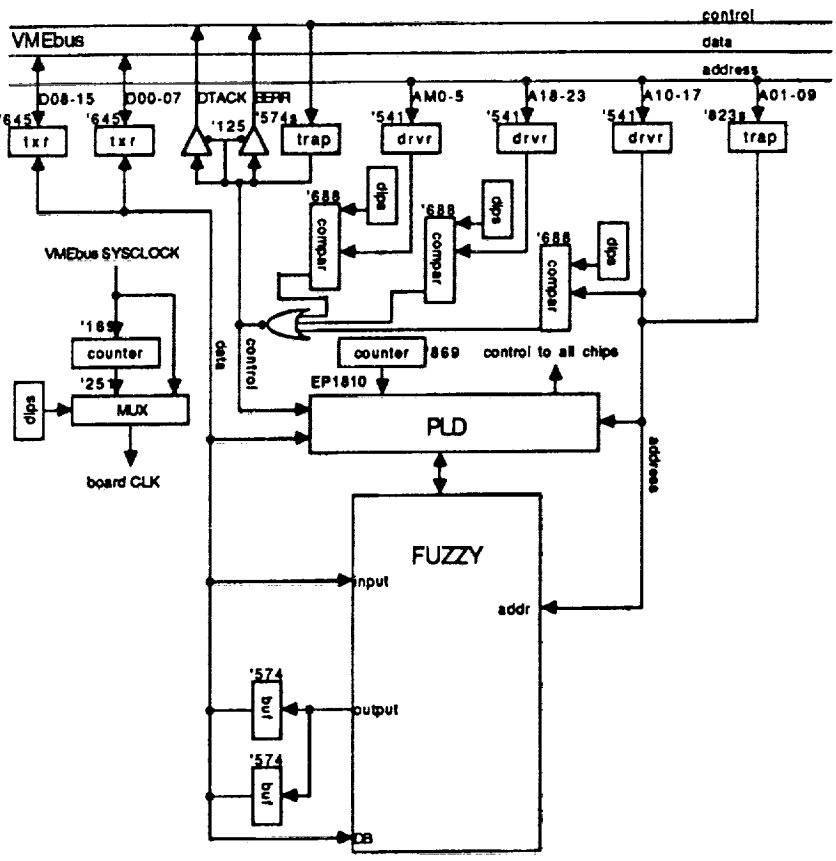


Figure 5: Details of PCB Architecture

implement the network. Any logical configuration of the seven chips may be specified in software, e.g. seven in parallel, 4-2-1 binary tree, etc. Any fuzzy output may be routed to any input. With the new board more inputs may be processed and hierarchies of rule sets may be explored. We can simulate rules with up to 16 conditions in the IF-part by using three layers of Fuzzy chips. Another application is to load multiple rule sets for different tasks in a single board. This is done by configuring multiple chips in parallel. The new printed circuit board architectural concept is shown in Figure 6.

This arrangement exploits an important feature of the Fuzzy chip. Normal input to the chip is by 6-bit integers which the chip *fuzzifies* into 64-value membership functions to be fed into the processing pipeline. The final output membership function is *defuzzified* into a 6-bit output integer. However, the chip has another mode of operation. Any input or output can bypass the [de]fuzzification process so that I/O occurs in *streaming* mode. The full 64-value input or output membership function is placed on the pins, one value per clock cycle. When an output of one chip is connected to an input of another chip (or itself), communication can be done in streaming mode without the loss of information inherent in the [de]fuzzification operations. On this system board, all inter-chip communication is done in streaming mode.

The new board also has four 64-value FIFO queues which allow final output to the host to be done in streaming mode. The application process is then free to perform its own custom operations on the full output membership functions. The final defuzzification is no longer limited to a centroid method. One can, also, generate the result in higher precision than 6 bits if necessary.

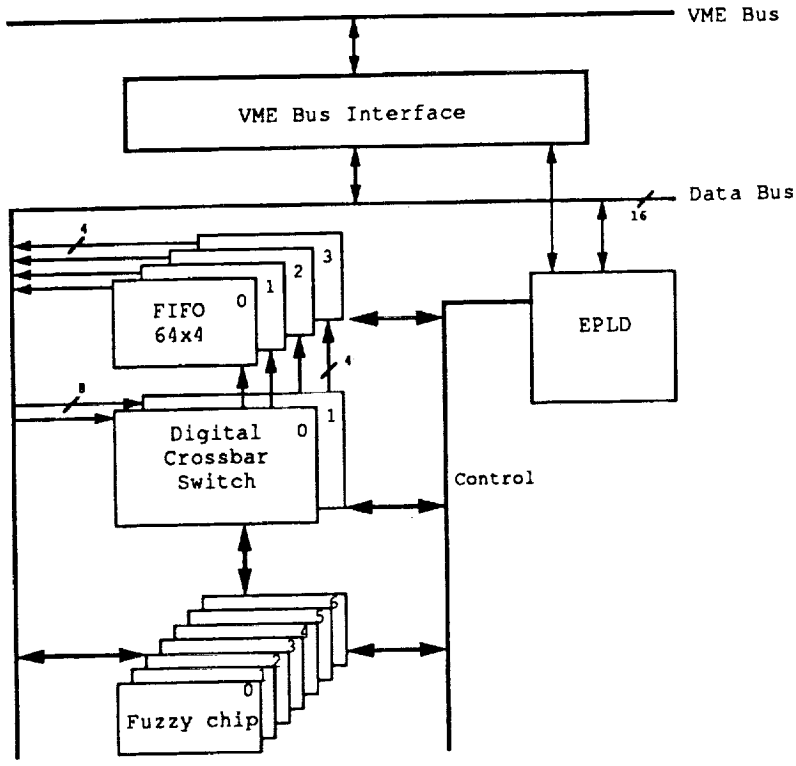


Figure 6: Seven Chip System Architecture.

The new board will be installed at ORNL in August, 1990. In addition to navigational tasks the system will be used to explore fuzzy logic control of manipulator arm functions.

5 Software Interface

High level C language functions can hide the operational details of the board from the applications programmer. The programmer treats rule memories and fuzzification function memories as local program structures passed as parameters to the C functions. Similarly, local input variables pass values to the system and outputs return in local variable function parameters. Programmers are only required to know the library procedures. Some procedures provided for the version 1 board are described in the following table.

1. *WriteRule(rulenum, ruledata)* – The rule data structure pointed to by *ruledata* is written to the board.
2. *ReadRule(rulenum, ruledata)* – Reads back into *ruledata* the rule identified by *rulenum* currently stored in the chip.
3. *WriteFuzz(fuzznum, fuzzdata)* – Fuzzification table is written to the board.
4. *StartFZIAC(inpA, inpB, inpC, inpD)* – Four inputs are sent to the fuzzy board and inference processing will be started.
5. *ReadOut(outE, outF)* – Both outputs are read from the board. Inference process will be continued.

6. *StopFZIAC(outE, outF)* – Both outputs are read from the board. Inference process will be halted.

6 Summary

We have described the architecture and associated high level software of two VME bus board systems based on a VLSI fuzzy logic chip. In addition to operating in the robot at ORNL, the single chip board is installed on a Sun-3 workstation at the University of North Carolina for further research and software development. For example, it is useful to provide an X-window based user interface to this fuzzy inference board. The complex and flexible architecture of the multiple chip board will require more sophisticated support software to facilitate exploration of various hierarchical interconnection schemes.

7 Acknowledgements

The research reported here is supported in part by Oak Ridge National Laboratory, by MCNC Design Initiative Program, and by NASA Ames Research Center.

References

[1] Corder, R. J., "A High-Speed Fuzzy Processor," *Proc. of 3nd IFSA Congress*, pp. 379-381, August 1989.

[2] Holmblad, L. P. and Ostergaard, J. J., "Control of a Cement Kiln by Fuzzy Logic," *Fuzzy Information and Decision Processes* (eds. M. M. Gupta and E. Sanchez) pp. 389-399, 1982.

[3] Kawano, K., M. Kosaka, and S. Miyamoto, "An Algorithm Selection Method Using Fuzzy Decision-Making Approach," *Trans. Society of Instrument and Control Engineers*, Vol. 20, No. 12, pp. 42-49, 1984. (in Japanese)

[4] King, P. J. and E. H. Mamdani, "The Application of Fuzzy Control Systems to Industrial Processes," *Automatica*, Vol. 13, No. 3, pp. 235-242, 1977.

[5] Maeda, Y., "Fuzzy Obstacle Avoidance Method for a Mobile Robot Based on the Degree of Danger," *Proc. of NAFIPS'90*, pp.169-172, June 1990.

[6] Shingu, T. and E. Nishimori, "Fuzzy-based Automatic Focusing System for Compact Camera," *Proc. of 3nd IFSA Congress*, pp. 436-439, August 1989.

[7] Togai, M. and H. Watanabe, "An Inference Engine for Real-time Approximate Reasoning: Toward an Expert on a Chip," *IEEE EXPERT*, Vol. 1, No. 3, pp. 55-62, August 1986.

[8] Watanabe, H., W. Dettloff and E. Yount "A VLSI Fuzzy Logic Inference Engine for Real-Time Process Control," *IEEE Journal of Solid-State Circuits*, Vol.25, No.2, pp.376-382, April 1990.

[9] Weisbin, C.R., G. de Saussure, J.R. Einstein, and F.G. Pin, "Autonomous Mobile Robot Navigation and Learning," *Computer*, Vol.22, No.6, June 1989.

[10] XYCOM, *XVME-85 Prototyping Module Preliminary Manual*, 1984.

[11] Yamakawa, T. and T. Miki, "The Current Mode Fuzzy Logic Integrated Circuits Fabricated by the Standard CMOS Process," *IEEE Transactions on Computers*, Vol. C-35, No. 2, pp. 161-167, February 1986.

[12] Yasunobu, S. and S. Miyamoto, "Automatic Train Operation System by Predictive Fuzzy Control," in *Industrial Applications Of Fuzzy Control*, M. Sugeno (Ed), pp. 1-18, 1985.

[13] Zadeh, L. A., "Outline of a New Approach to the Analysis of Complex Systems and Decision-Making Approach," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. SME-3, pp. No. 1, pp. 28-45, January 1973.

LEARNING CONTROL OF INVERTED PENDULUM SYSTEM BY NEURAL NETWORK DRIVEN FUZZY REASONING

– The Learning Function of NN–Driven Fuzzy Reasoning under Changes of Reasoning Environment –

Isao HAYASHI, Hiroyoshi NOMURA and Noboru WAKAMI

Central Research Laboratories, Matsushita Electric industrial Co. Ltd.
3–15, Yagumo–nakamachi, Moriguchi, Osaka, 570 Japan

Whereas conventional fuzzy reasonings are associated with tuning problems which are lack of membership functions and inference rule designs, a neural network driven fuzzy reasoning (NDF) capable of determining membership functions by neural network is formulated. In the antecedent parts of the neural network driven fuzzy reasoning, the optimum membership function is determined by a neural network, while in the consequent parts, an amount of control for each rule is determined by another plural neural networks. By introducing an algorithm of neural network driven fuzzy reasoning, inference rules for making a pendulum stand up from its lowest suspended point are determined for verifying the usefulness of the algorithm.

1. INTRODUCTION

Extensive applications of fuzzy reasoning for various control problems, and a number of actual examples of fuzzy control are reported [1] lately. However, the fuzzy reasoning is generally involved with a tuning problem [2], that is, the form of fuzzy number, and the fuzzy variables of antecedent parts and consequent parts of fuzzy inference rules, have to be adjusted for minimizing the difference between the estimation of fuzzy reasoning and the output data for given input data.

As a method to solve the tuning problem, a neural network driven fuzzy reasoning (NDF) [3, 4] by which inference rules are constructed from the learning function of neural network [5, 6] is previously reported. The NDF is a type of fuzzy reasoning having an error back – propagation type network [7] which represent fuzzy sets in its antecedent parts, while another error back – propagation type network represents an input – output relationship between input and output data of consequent parts of each rule.

In this paper, an algorithm for constructing inference rules based on NDF is introduced first, and an experimental verification of its effectiveness is performed taking an example for an inverted pendulum system.

In this experiment, a pendulum in its hanged position is surely swang up and is held at an inverted position by using a mechanism controlled by inference rules which are constructed by determining fuzzy sets from the observations of pendulum operator by utilizing NDF algorithm.

The inference period required for controlling the swing–up motion of pendulum is approximately 15 msec. As a parameter which governs the dynamic characteristics of inverted pendulum system, the length of pendulum is considered here, and changes of control characteristics of NDF caused by this is studied. Since the fuzzy set of antecedent parts and input – output relationship of consequent parts can be determined by means of NDF without finetuning of inference rules by utilizing the learning function of neural network acquired from the input–output data, it is an advantageous

method to solve tuning problems of fuzzy reasoning.

2.. ARTIFICIAL NEURAL NETWORK DRIVEN FUZZY REASONING (NDF)

The NN – driven fuzzy reasoning (NDF) is a fuzzy reasoning [8] using linear functions in its consequent parts. In a NDF, the membership functions in the antecedent parts is determined in a multi–dimensional space. For example, the following rules R1, R2, and R3 of the conventional fuzzy reasoning wherein x1 and x2 are input variables, y1, y2, and y3 are output variables, and a10 and a11 are coefficients, and FSL and FBG are fuzzy numbers where SL and BG mean small and big respectively, are considered.

R1 ; IF x1 is FSL and x2 is FSL,
THEN y1 = a10 + a11x11 + a12x12
R2 ; IF x1 is FSL and x2 is FBG,
THEN y2 = a20 + a21x21 + a22x22 (1)
R3 ; IF x1 is FBG,
THEN y3 = a30 + a31x31

Since the above condition means that x1 is small and x2 is small in the antecedent parts, the fuzzy sets F1 = FSL ∩ FSL can be constructed in a partial space of the input as shown in Fig. 1. The same can be applied for the fuzzy sets to be constructed for R2 and R3 likewise. Since the boundary between the each partial space is vague, the boundary is shown by the hatched lines. That means that the input space consisted of x1 and x2 is divided into individual partial spaces by the number of fuzzy rules, and the fuzzy sets of antecedent part of each inference rules are constructed in each partial space, while the NDF is determined by the fuzzy sets of antecedent parts by utilizing the back – propagation type network.

An explanation for the back – propagation type network is as follows. Since neural networks are constrained by a general type processing unit found in the neural system, and the processing unit in a neural network shares some of the physical properties of real neurons, the processing unit is called neuron here.

Fig. 2 shows an example of fundamental layered back propagation type networks containing M layers, where the first layer is called an input layer, the M–th layers are output layers, and other layers are called intermediate layers. Every neuron within these layers represents respective correlation between the multi–inputs xij and multi–outputs yi expressed by the following equations.

$$yi = f\left(\sum_{j=1} \alpha ij\ xij + \theta\right) \quad (2)$$

$$f(Z) = \frac{1}{1 + \exp(-Z)} \quad (3)$$

where θ is a weight showing a correlation between neurons.

In this paper, for a given input and output expressed by x = (x1, x2, ... , xn) and y = (y1, y2, ... , yw) respectively, the input–output correlation of back–propagation type network as a whole is expressed by;

$$y = NN(x) \quad (4)$$

The structure of model function NN (x) is characterized by M–layers [u1 x u2 x .. x uM] where ui , i = 1, 2, ... , M are the numbers of neurons within the input, hidden and output layers respectively. Fig. 2 shows a structure of back propagation type network consisting of four–layers [3 x 2 x 2 x 2].

Fundamental considerations made on the NDF are that the model equations y1, y2, and y3 in the consequent parts are identified as the non – linear Eq. (4) for obtaining the model equations.

The fundamental consideration made on the membership functions in the antecedent parts is a method shown in Fig.

3.

If the relationships between rules R1, R2, R3 and input data (xi1, xi2) where i = 1, 2, ... , N are considered, the first data x1 are (x11, x12) = (0.2, 0.15), and these data belong to rule R1. Thus the data attribution to the rule can be expressed by (R1, R2, R3) = (1, 0, 0). The back-propagation type network three-layers [2X3X2] of which input and output layer are (X11, x12) and (R1, R2, R3) respectively can be derived from the input-output data utilized in the learning process. However, the maximum number of learning is limited to be less than about 1000.

When another data different from the input-output data are assigned to the neural network, the estimated values of back propagation type neural network are considered as membership values of fuzzy sets in the antecedent parts since the estimated value represents the attribution of data to each rule. A rule division performed by NDF is typified in Fig. 4 which shows non-linear divisions unlike the rectangular divisions shown in Fig. 2.

Pao proposed a method for determining fuzzy sets by using a neural network [9], and obtained intersections and union sets of fuzzy sets. However, what he carried out were the determinations of intersection and union sets of fuzzy sets from the coupling patterns between each unit of neural network, and was not the type determining the shape of fuzzy sets from the input-output data such as excutable by NDF.

In a NDF, the control rules are represented by an IF-THEN format shown below.

$$\begin{aligned} Rs\ &;\ IF\ \ x = (x1, x2, \ldots, xn)\ \text{belongs to As,} \\ &\quad THEN\ \ ys = NNs(x1, x2, \ldots, xm) \\ &\qquad \text{where}\ \ s = 1, 2, \ldots, r,\ m \leq n \end{aligned} \tag{5}$$

The number of inference rules employed here is expressed by r, and As represents a fuzzy set in the input space area of antecedent parts. The degree of belongings of input x = (x1, x2, ... , xn) to the s-th inference rule is defined to as the membership value of fuzzy sets As to the input x. Furthermore, the amount of operations ys of consequent parts is an estimated value for a case where a combination of input variables (x1, x2, ... , xm) is substituted in the input layer of back propagation type network, wherein the number of variables employed in this case is m according to a method for selecting the optimum model employing back-propagation type network.

Although it is also possible to determine an overall non-linear relationship by using only one back-propagation type network, the determination of overall input-output relationship by applying back-propagation type network for each partial space is considered more advantageous than employing only one back-propagation type network for better clarification of overall non-linear relationship.

In order to carry out an optimum model selection for the back-propagation type network of consequent parts, a stepwise method [10] by which a specified input variable derived from a combination of input variables is introduced and removed for obtaining a model which outputs an optimum estimated value, is available.

In the present work, only an elimination of input variables from a combination of input variables by utilizing back-propagation type network is performed for deriving an optimum combination of input variables and model formula. A summation of the second powers of residuals is employed for evaluating and dtermining the input variables.

An explanation for the algorithm of NDF is given in the following referring a block diagram of NDF shown in Fig. 5. The stepwise procedures taken for obtaining the inference rules and the control value yi* for the input data xi are as follows.

Step 1: Selection of input variables, x1, x2, ... , xn, which are related to the control value y. This is for an assumed case where the input

– output variables (yi, xi) = (yi, xi1, xi2, ... , xin) where i = 1, 2, ... , N, are obtained and the input data xij where j = 1, 2, ... , n, are the i–th data of input variable xj.

Step 2: Division of input–output data into r classes of Rs where s = 1, 2, ... , r. As mentioned before, each partition is regarded as an inference rule Rs, and the input–output data for each Rs are expressed by (yi (s), xi(s)) where i = 1, 2, ... , Ns providing that Ns is a number of input–output data for each Rs.

Step 3: Decision of membership functions in the antecedent parts by using the neural network NNmem shown Fig. 5 providing that the structure of a back – propagation type network is a M–layered [n × u2 × ... × uM–1 × r]. The method for determining the form of membership functions is described previously.

Step 4: Decision of control models in the consequent parts by using the neural networks NN1, NN2, ... , NNr shown in Fig. 5 providing that the structure of each back–propagation type network NNs is a M – layered [k × u2 × ... × uM–1 × 1] where k = n, n–1, ... , 1, and selections of optimum model for each NNs are performed.

Consequently, the stepwise procedures for determining input variables by utilizing back – propagation type network, and the method for determining the structure of consequent parts are described in the following.

Setting a condition at k = n, the input variables xi = (xi1, xi2, ... , xik) where i = 1, 2, ... , N, are assigned for the input layer of each NNs, and the output variables yi is assigned for the output layer of each NNs, where the input variables assigned for the input layer and the output variables assigned for the output layer are respectively expressed by:

$$s = \{x1, x2, \ldots, xk\} \quad (6)$$

$$s = \{y\} \quad (7)$$

where s represents a set of input variables assigned for the input layer of each back–propagation type network NNs, and s represents a set of output variables assigned for the output layer of NNs.

An estimation eyi for the input data xi1, xi2, ... , xik can be obtained after repeated learnings made on the back – propagation type network of NNs. However, the number of learnings is set at approximately 3000. Then the sum of mean squared errors of the output data yi and estimation eyi is calculated for obtaining an evaluation value ⊛ks required for determining the input variables.

$$\circledast ks = \left(\sum_{i=1}^{N} (yi - eyi)^2\right) / N, \quad s = 1, 2, \ldots, r. \quad (8)$$

In order to study the degree of correlation of the input variables xj to the output variables y, the input variable xj is temporarily removed from the set of input variables {x1, x2, ... , xk}. The input data from which the input variables xj is removed, xi1,.., xij–1,.., xij+1, ... , xjk where i = 1, 2, ... , N, are assigned to the input layer of M – layer of the back propagation type network [k – 1 u2 ... uM – 1 1], and the output data yi are assigned to the output layer. Then, the estimation eyi' for the input data xi1, ... , xij–1, ... , xij+1, can be obtained after the back – propagation learning. An evaluation value ⊛k – 1sj required for determining the input variables is derived by calculating the sum of mean squared errors of the output data yi for this estimation eyi'.

$$\circledast k-1sj = \left(\sum_{i=1}^{N} (yi - eyi')^2\right) / N, \quad s = 1, 2, \ldots, r. \quad (9)$$

The same calculations are conducted for the input variables other than xj for determining the evaluations ⊛k–1s1, ⊛k–1s2, ... , ⊛k–1sj , ... , ⊛k–1sk. The calculation of evaluation which takes a minimum value,

$\Theta k-1sc$, can be obtained by;

$$\Theta k-1sc = \min \Theta k-1sj, \quad \text{where } j = 1, 2, \ldots, k. \tag{10}$$

Eq. 10 shows that the evaluation $\Theta k-1sc$ obtained by removing the input variables xc from the set of input variables takes a minimum value among evaluations $\Theta k-1s1$, $\Theta k-1s2, \ldots, \Theta k-1sj, \ldots, \Theta k-1sk$. By comparing the value of $\Theta k-1sc$ of Eq. 10 to the value of Θks of Eq. 8, the set of variables, Λs, is altered as follows.

$$\Lambda s = \{x1, x2, \ldots, xc-1, xc+1, \ldots, xk\}, \quad \text{If } \Theta k-1sc < \Theta ks \tag{11}$$

$$\Lambda s = \{x1, x2, \ldots, xk\}, \quad \text{If } \Theta k-1sc \geqq \Theta ks \tag{12}$$

When Eq. 11 is established, the sum of mean squared errors can be decreased by removing the input variables xc, and this means that the estimation eyi' represents yi better than eyi.

Therefore, the correlation of input variables xc to the output variables y is considered weak, and the input variables are removed from the input variable sets Λs. As a result of this, a set of newly established input variables is then consisted of k−1 input variables.

On the other hand, the effectiveness obtained by removing input variables temporarily can not be attained when Eq. 12 is established, and this fact means that the input variables xc are strongly correlated with the output variables y, and the number of sets of input variables Λs is left unchanged as k.

In cases where the input variables can be reduced, k is altered to n−1, n−2, ..., 1, and Step 4 is repeated until Eq. 12 can be established, and the procedures for reducing the input variables of back−propagation type network NNs are completed until Eq. 12 can be established.

Thus, the back − propagation type network NNs having the final set of input variables, $\Lambda s = \{x1, x2, \ldots, xm\}$ obtained at the time of procedure completion, becomes an optimum back − propagation type network representing the structure of consequent parts of rule Rs. The same step procedures are conducted for each NNs for determining the consequent parts of all the inference rules. This procedure to reduce the number of input variables is called a stepwise variable reduction method utilizing back − propagation type network.

Step 5: The estimation yi* can be derived by the equation shown below.

$$yi* = \frac{\sum_{s=1}^{r} \mu As\ (xi1, xi2, \ldots, xin) \times meyi(s)}{\sum_{s=1}^{r} \mu As\ (xi1, xi2, \ldots, xin)} \tag{13}$$

$$i=1, 2, \ldots, N.$$

where meyis is an estimation obtained by the optimum back − propagation type network derived by Step 4.

Fig. 5 shows that the estimation yi* can be derived from the results obtained by conducting product operations between the membership values of antecedent parts of each inference rules, or $\mu As(xi1, xi2, \ldots, xin)$ and the estimation of consequent parts, or meyi (s), and by conducting summation operations between each rule continuously. However, Fig. 5 shows a case where a condition of $\mu As(xi1, xi2, \ldots, xin) = 1$ is established.

3. APPLICATION TO INVERTED PENDULUM SYSTEM

The NDF proposed by the authors is capable of forming inference rules automatically, i. e., the function of self − autotuning, and proposed here is an inverted pendulum system to which a learning function by using a NDF is applied. In the algorithm employed for the experiment, four inputs and one output data are acquired by observing manual operating controls, and

fuzzy inference rules and membership functions are then automatically constructed from the acquired data by using the algorithm of NDF.

Fig. 6 shows a structure of inverted pendulum system consisting of four elements explained in the following;

1) Cart which runs on a rail.
2) Pendulum rotatable freely around an axis of cart.
3) Motor which drives the cart.
4) Fixed pulleys and belt system which connect above three parts.

The pendulum angle apart from the perpendicular θ degree and the distance from the original position of cart are detected by the potentiometer b and a shown in Fig. 6 respectively. These are digitized by an AD converter, and the digitized signals are fed to a personal computer wherein the velocities of inverted pendulum angle and the cart distance are calculated from the differences in those obtained at every sampling. The output for the motor control system is then calculated from four variables, i.e., the pendulum angle, angular velocity, cart distance, and the cart velocity by using an algorithm of NDF. As the motor control signal is derived by a personal computer in a digital form, this is converted into an analog value through a DA converter.

The inverted pendulum system has two control areas consisting of a linear-controlling area where the pendulum is standing, and a non-linear controlling area where the pendulum falls. The authors constructed an inverted pendulum system in the linear - controlling area by using a conventional fuzzy control, and a control model constructed in the non - linear controlling area by utilizing NDF, is reported here.

The configuration of inverted pendulum system and the control computer are as follows.

Body : Length of 1,410mm; width of 400mm, height of 880mm.
Pendulum : Length of 400mm , weight of 40g , diameter of 4mm.
Drive force : 25W DC motor with a gear ratio of 12.5 : 1.
Sensors : Potentiometer to measure the distance from the original position of the cart, and another potentiometer to measure the pendulum angle.
Micro-computer : CPU 80286
Program : C-language, 21K bytes.

The preparation of control rules applicable to an inverted pendulum made according to an algorithm developed for constructing the inference rules by applying NDF is now described in the folowing.

Step 1: Preparation of input - output data. This is acquired by an operator who tries to swing up a pendulum by moving the cart right or left direction on the rail by pressing either of corresponding controller buttons until the pendulum is brought to its inverted position, and the following input-output data with a sampling period of 4 msec are recorded:

Output variable
y : Motor control signal (V).
input variables
x1 : Distance from the original cart position.
x2 : Velocity of x1 (cm/sec).
x3 : Pendulum angle (deg).
x4 : Velocity of x3 (deg/sec).

where the input variables x2 and x4 are derived from the differences produced in x1 and x3 values. Approximately 1,000 to 3,000 data are acquired from these manual operations, and from these, 98 input-output data shown in Table 1 applied for the NDF are extracted.

Step 2: Setting of two rules for the input-output data considering data distributions.

Step 3: Determination of membership functions of antecedent parts. A three - layered [4 × 6 × 2] back - propagation type network employed here for determining the antecedent part construction is employed here, and the number of learnings is set at about 1000.

Step 4: Determination of consequent part structure. A three-layered [k × 6 × 1] where k = 4, 3, 2, 1, back-propagation type network for determining the consequent part structure is employed here, and the number of learnings of each back-propagation type network is set at about 3000.

By using a stepwise variable reduction method, we obtain;

$$\circledast 41 = 0.016 \quad (14)$$

$$\circledast 311 = \min \circledast 31j \ (= 0.007),\ j = 1, 2, 3, 4. \quad (15)$$

Therefore,

$$\circledast 311 < \circledast 41 \quad (16)$$

Thus, by removing the input variables x1, we obtain $\Lambda s = \{x2, x3, x4\}$. As for $\Lambda s = \{x2, x3, x4\}$, a stepwise variable reduction method is applied again. By combining Eqs. 8, 9, and 10, we obtain the followings.

$$\circledast 311 = 0.007 \quad (17)$$

$$\circledast 213 = \min \circledast 21j \ (= 0.021),\ j = 2, 3, 4. \quad (18)$$

This means,

$$\circledast 213 > \circledast 311 \quad (19)$$

Thus, no reduction of input variables is made, and the algorithm for Rule 1 is completed by the second calculation process. The inference rules consequently obtained by these are as follows:

R1 ; IF x = (x1, x2, x3, x4) belongs to A1,
THEN y1 = NN1(x2, x3, x4),
R2 ; IF x = (x1, x2, x3, x4) belongs to A2,
THEN y2 = NN2(x1, x2, x4) (20)

Photographs 1 and 2 show the swing-up motions of pendulum controlled by fuzzy inference rules expressed by Eq. (20). Photograph 1 shows sequential motions of pendulum swang from its stable equillibrium state to an inverted stand-still state. The estimation yi* can be derived from Eq. (13). The pendulum can be surely brought to its inverted position regardless the cart position on the rail, or a disturbance applied to the pendulum. Photograph 2 shows the controls of swing-up motion for various given pendulum angles.

An experimental study for the limitation of control performed by NDF is conducted by changing the parameters which govern the dynamic characteristics of controlled object, and the length of pendulum is taken as a parameter governing the dynamic characteristics of pendulum here. The initial position of cart is set at the center position of belt on which the inverted pendulum device is mounted, and the pendulum angle is set at 0 degree when it is hanged down initially and ±180 degree is specified when the pondulum is in an inverted position. The angle is incremented for its clockwise rotation, and decremented for its anti-clockwise rotation.

The inference rules are constructed for a case where pendulum length is 40 cm, and Fig. 7 shows a response of pendulum of such. Figs. 8, 9, and 10 respectively show the responses of the 20, 30, and 50 cm long pendulums. The shifts of pendulum angle are shown by solid lines, and the changes of angular velocity are shown by broken lines in these figures. However, only the changes of pendulum angle and angular velocity until the pendulum comes to an inverted position, and no response after completion of inversion are shown there.

As for the learning of inverted pendulum, the swing-up process of pendulum is learnt for constructing an inference rules applicable to the process of pendulum starting from the hanged down postion to a nearly inverted position. The inverted position is defined as a pendulum angle close to ±180 degree and its angular velocity nearly zero at that time.

As shown in Fig. 7, the pendulum reaches at −180 degrees at 5.4 seconds after starting of control attaining an angular velocity of about 0 deg / sec, and the

pendulum stand still at an inverted position. This is rather natural consequence since an inference rules are established for a 40 cm long pendulum.

In a case where the length of pendulum is set at 20 cm as shown in Fig. 8, a large velocity change is observed, and the angle became 180 degrees at 6.2 seconds attaining an angular velocity of about 0 deg /sec. Although the pendulum reaches at an inverted position and stays there, the angular velocity is larger and a longer lead-in period is required.

Fig. 9 shows a transient response of a 30 cm long pendulum. The pendulum is brought to its inverted position showing a response similar to that obtained with the 40 cm long pendulum, but the angle reaches at −180 degrees at 3.9 sec yielding a higher anlgular velocity which equals to about one half of that obtained with the 20 cm long pendulum. The overall controllable characteristics is silimar to that of 40 cm long pendulum.

Fig. 10 shows a transient response obtained with a 50 cm long pendulum which was unable to brought to its inverted position. As seen in Fig. 10, the pendulum angle could not be brought to its ±180 degree position despite of longer lead-in period. The correlation between dynamic characteristics of pendulum and the variable length of pendulum can be summarized as follows.

1) By applying a NDF to a pendulum system of which length is varied from 40 to 20 cm, a stable operation to bring the pendulum to its inverted position became feasible despite of lead−in period required for its motions. That is to say, the robustness of NDF is higher for the shorter length pendulum.

2) for the cases of longer pendulums, however, the suppression of deviations of the control system can not be attained, and this means that a relearning or additional learning is necessary for the NDF applied for a longer pendulum.

4. CONCLUSION

While the conventional fuzzy reasoning is associated with inherent tuning problems, NDF is, upon input − output variables are given, capable of determing an optimum inference rules and membership functions by utilizing its nonlinearity of back−propagation type network and learning capabilities. In order to verify the usefullness of NDF, it is applied to an experimentally constructed pendulum system wherein the pendulum is brought to its inverted position and stayed there starting from its stable hanged position. The length of pendulum is also altered for confirming itrs effects on the control characteristics of NDF.

Since this method is capable of deriving an inference rules by using the learning function of back−propagation type network, the learning function can be introduced in the fuzzy control. The development of learning function adaptive to the changes of dynamic inference environment should be an important subject to be discussed in future.

5. REFERENCES

1) Hirota, Kaoru, " Robotics and Automation Industrial Applications in Japan", 3rd IFSA Congress, Seattle, 1989, pp.229−230.

2) Lee, Chuen−Chien, "A Self−Learning Rule − Based Controller with Approximate Reasoning", Memorandom, No. UCB / ERL, M89/84, Univ. Calf. Berk, 1989.

3) Hayashi, Isao, Nomura, Hiroyoshi and Wakami, Noboru, "Artificial Neural Network Driven Fuzzy Control and it's Application to the Learning of Inverted Pendulum System", 3rd IFSA Congress, Seattle, 1989, pp.610−613.

4) Takagi, Hideo and Hayashi, Isao, " Artificial Neural Network Driven Fuzzy Reasoning", Int. J. Approximate Reasoning, 1990, (in press).

5) Anderson, James and Rosenfeld, Edward, " Neurocomputing", MIT Press, Cambridge, Mass., 1988.

6) Tank, David and Hopplield, John, "Collective Computations in Neuronlike Circuits", Scientific American, December, 1987, pp.104-114.
7) Rumlhart, David E., Hinton, Geoffrey E., and Williams, Ronald J., "Learning Representaions by Back-propagating Errors", Nature, Vol.323, No.9, 1986, pp.533-536.
8) Sugeno, Michio and Kang, G.T., "Structure identification of fuzzy model", Fuzzy Sets and Systems, Vol.28, No.1, 1988, pp.15-33.
9) Pao, Yoh-Han, "Adaptive Pattern Recognition and Neural Networks", Addison-Wesley, 1989.
10) Draper, N. R. and Smith, H., "Applied Regression Analysis", John Wiley & Sons, 1966.

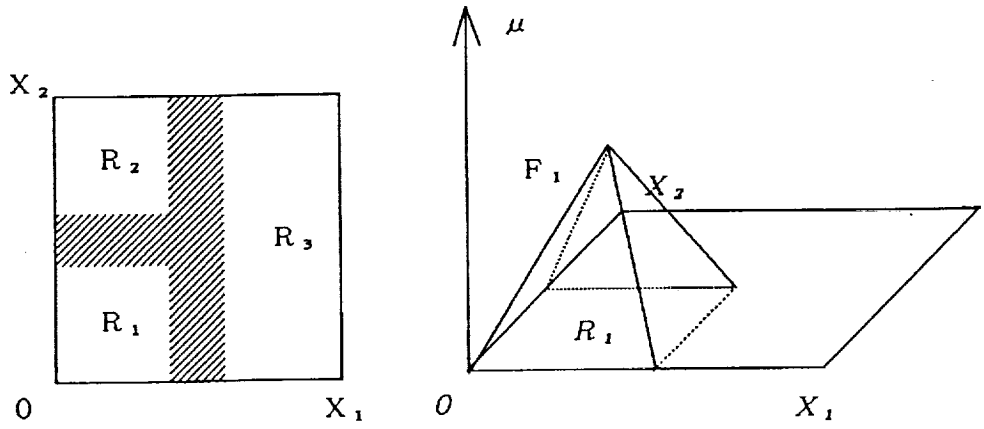


Fig.1 Conventional Fuzzy Partition of Rules

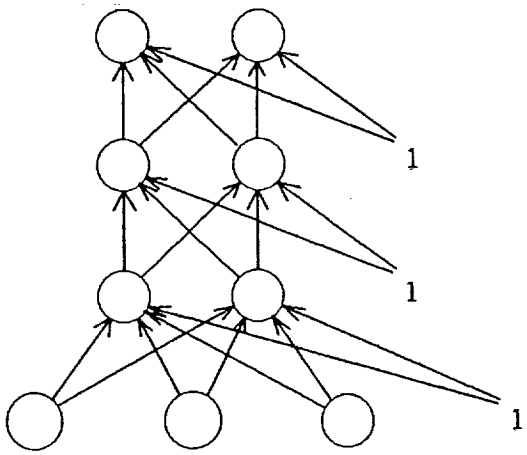


Fig.2 Example of Neural Network

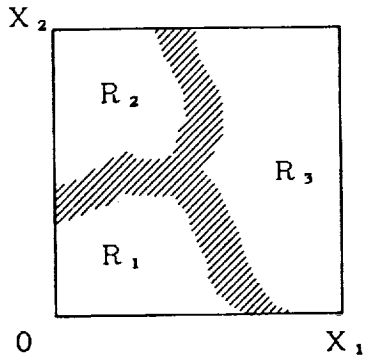


Fig.4 Proposed Fuzzy Partition of Rules

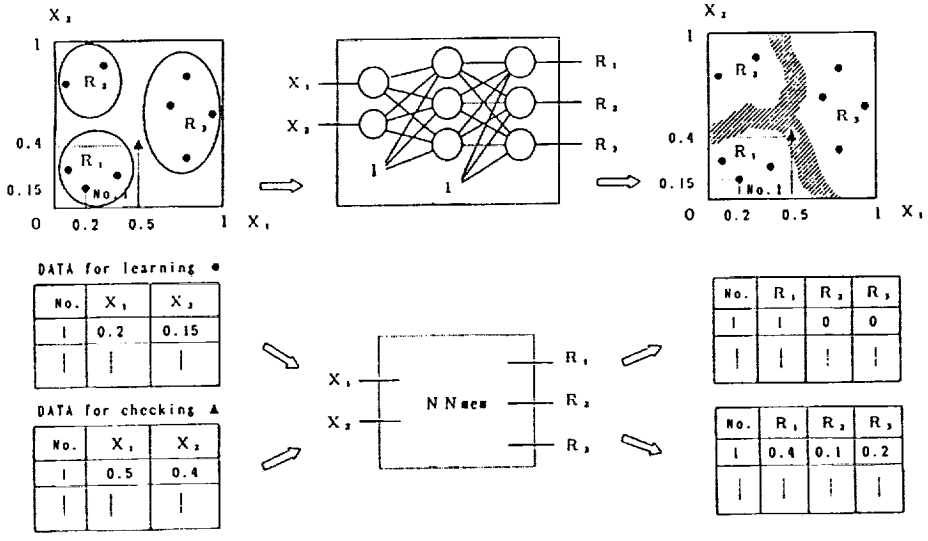


Fig.3 Decision of Membership Function in Antecedent Parts of Rules

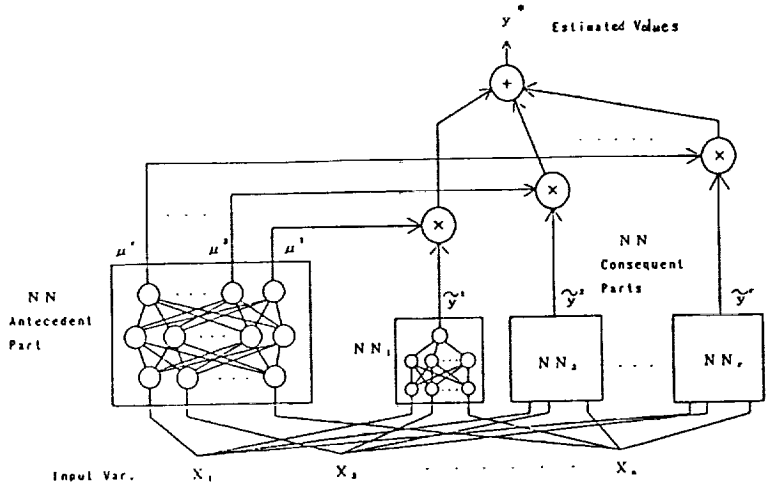


Fig.5 Block Diagram of Neural Network Driven Fuzzy Reasoning

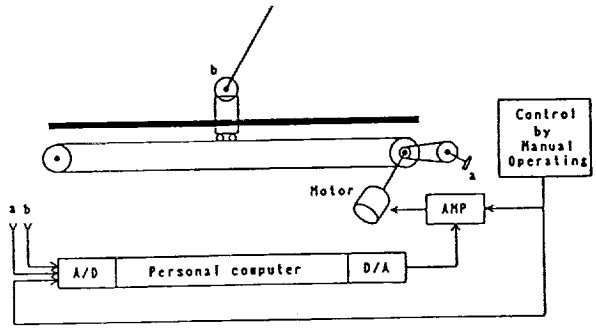


Fig.6 Structure of Inverted Pendulum System

Table1 Input and Output data of Inverted Pendulum System

No.	Input Data				Output Data
	x_1 [cm]	x_2 [cm/sec]	x_3 [deg]	x_4 [deg/sec]	y [v]
1	-1.1482	0.0000	178.5074	0.0000	0.7597
2	-0.0201	8.5486	180.9129	34.5660	0.7421
3	3.2197	29.9073	185.6439	34.5660	0.7617
4	7.8338	38.4472	188.4660	0.0000	0.0039
5	10.9510	4.2697	182.7386	-121.0536	-0.7168
6	9.3718	-21.3586	165.3085	-155.6554	-0.7968
7	5.3319	-38.4560	151.5283	-69.1678	-0.7519
8	0.1432	-42.7261	150.9487	51.8840	-0.7519
93	7.9980	42.7261	85.663	-380.4464	-0.0136
94	11.7713	4.2701	199.1743	-639.8375	-0.7265
95	10.6843	-17.0885	117.4961	-622.5536	-0.7519
96	7.0135	-42.7261	56.6514	-345.8786	-0.7519
97	1.9891	-38.4560	27.0170	-138.3357	0.0039
98	-2.9937	-34.1774	16.6419	-51.8822	-0.0019

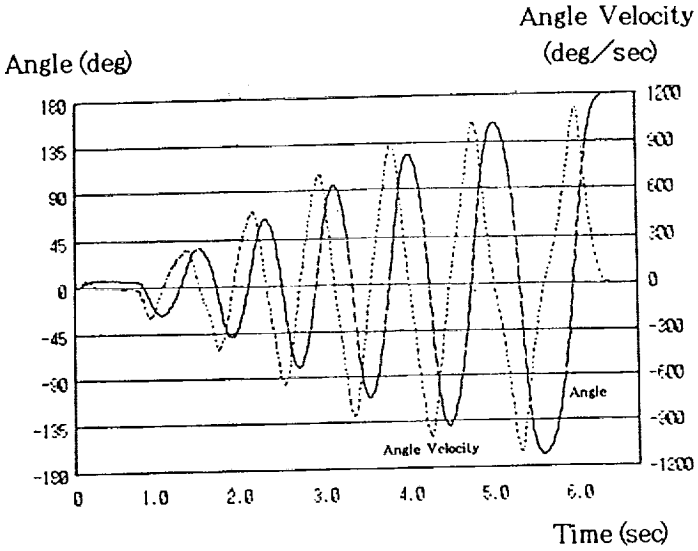


Fig.7 Angle and Angular Velocity of 40 cm Long Pendulum

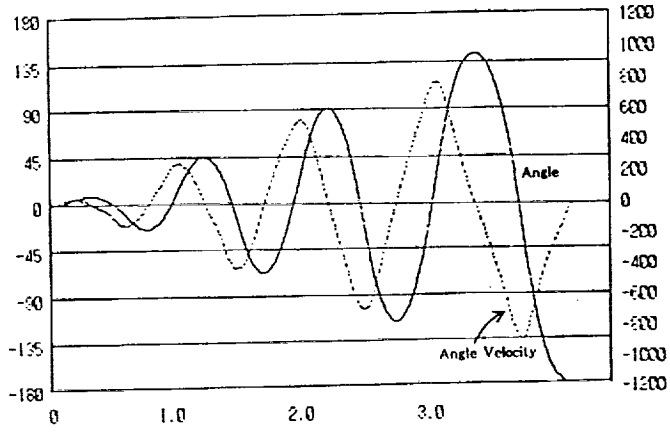


Fig.8 Angle and Angular Velocity of 20 cm Long Pendulum

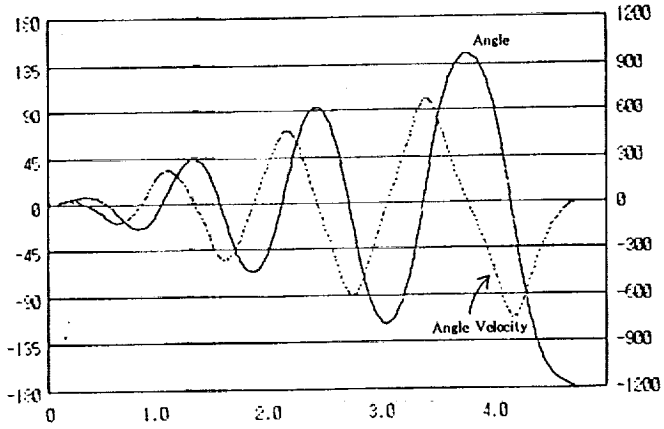


Fig.9 Angle and Angular Velocity of 30 cm Long Pendulum

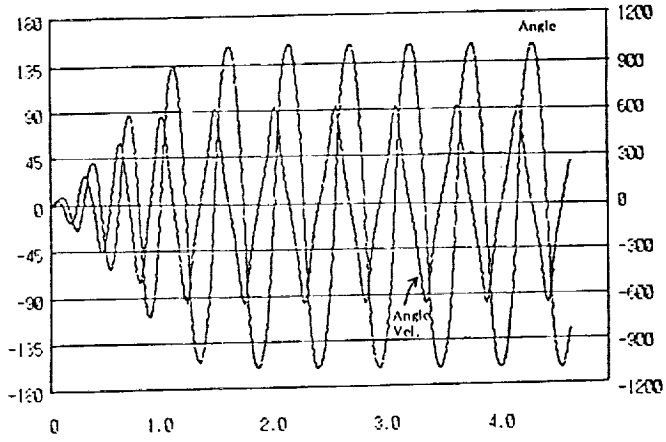


Fig.10 Angle and Angular Velocity of 50 cm Long Pendulum

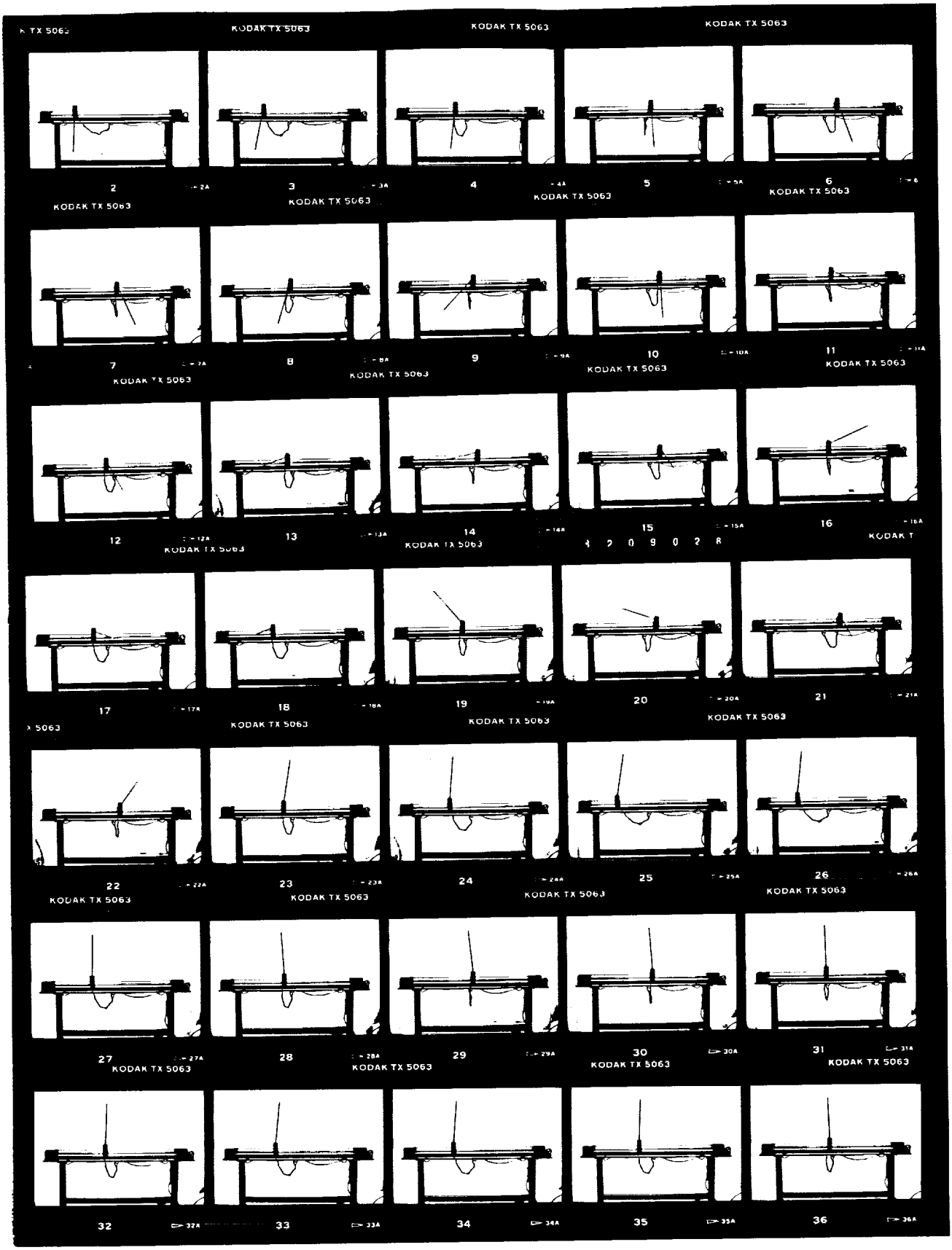


Photo.1 Control of Inverted Pendulum System (No.1)

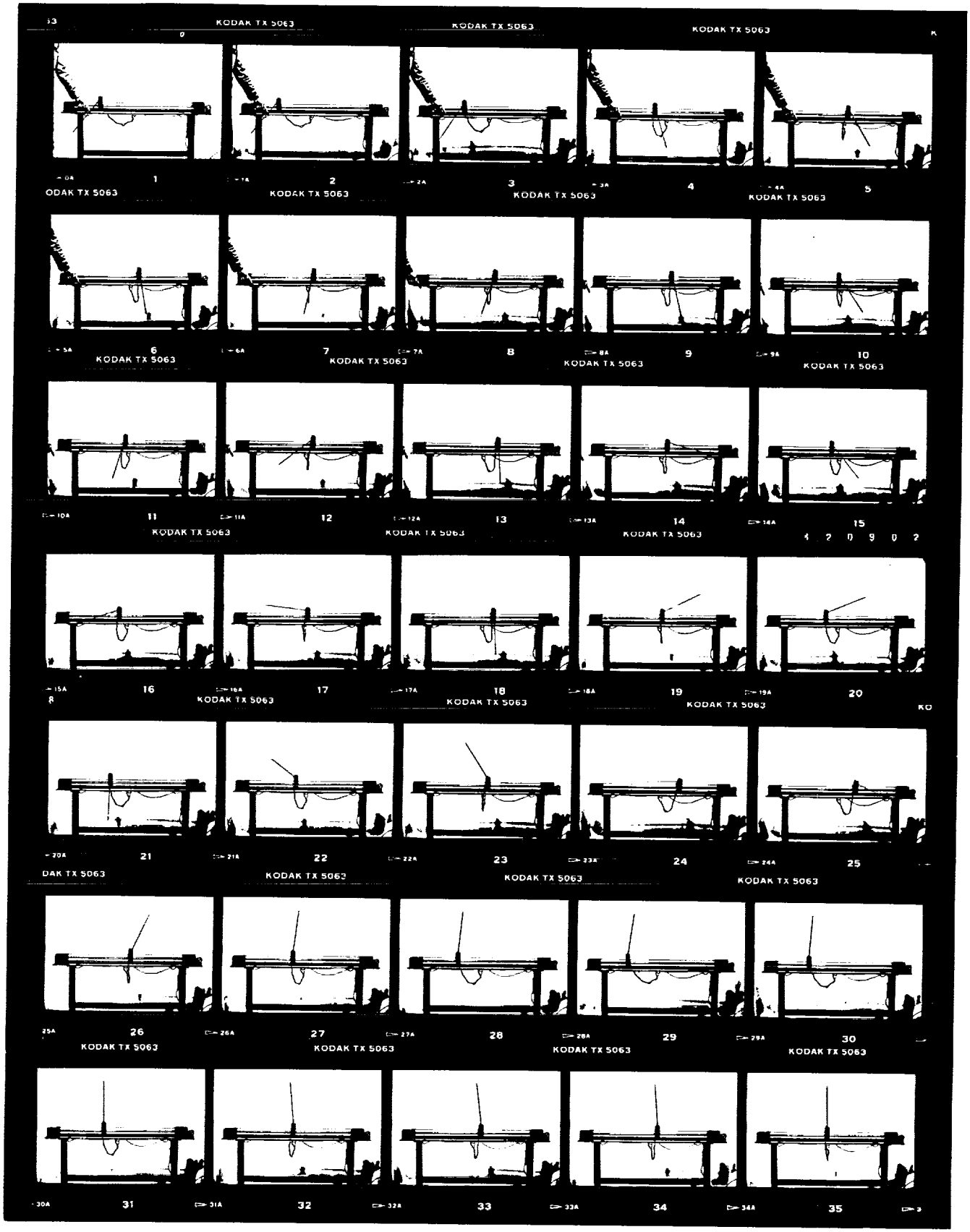


Photo.2 Control of Inverted Pendulum System (No.2)

Solution of Inverse Problem of Fuzzy Relational Equation by using Perceptron Model

Kaoru HIROTA *

Norikazu IKOMA **

* Department of Instrument and Control Engineering , College of Engineering, HOSEI University
** The HOSEI University Graduate School, Engineering Division, System Engineering

ABSTRACT

Max-min fuzzy relational system can be regarded as a network of max and min operational elements. Thus the inverse problem of fuzzy relational equation is interpreted as an input estimation problem from output values in the corresponding network. An approximate network model of fuzzy relational system is proposed. An algorithm of obtaining an approximate solution of the system is presented by using a neural network technique. The availability is discussed with a numerical experiment.

Key words : fuzzy relation, fuzzy inverse problem, neural network, perceptron model

Introduction

Inverse problem of fuzzy relational equation (Fuzzy Inverse Problem) was proposed by E.Sanchez in 1976[1]. The solution of fuzzy inverse problem was shown by Tukamoto et al in 1977[2]. And now, it is used for diagnosis of complicated systems.

Max-min fuzzy relational system can be regarded as a network which consists of max and min operational elements. Thus the fuzzy inverse problem is interpreted as an input value estimation problem from output values in the corresponding network.

In this view point, input of network can be identified when output and the network structure are given. Assuming the network of fuzzy relational system can be approximately constructed by the perceptron, we can regard the input of fuzzy system as the input of perceptron when output and the perceptron structure are given.

In this paper, an input value estimation algorithm based on perceptron model is proposed, and it is applied to solving the fuzzy inverse problem. Numerical experiment is done to investigate the availability of this method, and the result of experiments is discussed.

Input Estimation Algorithm of Perceptron Model

Perceptron neural network model[3] is used in this paper. It is summarized as follows;

a. Output value of j -th neuron in the k -th layer is denoted by

$$u_j^k .$$

b. Threshold value of j -th neuron in the k -th layer is denoted by

$$\theta_j^k .$$

c. Connection coefficient from the i -th neuron in $(k\text{-}1)$ th layer to j -th one in k -th layer is denoted by

$w^{k-1}{}_i{}^k{}_j$.

d. The relation of above three values are described by (1)-(3).

$$u_j^k = f(s_j^k) \tag{1}$$

$$s_j^k = \sum_i w^{k-1}{}_i{}^k{}_j \cdot u_i^{k-1} - \theta_j^k \tag{2}$$

$$f(s) = \frac{1}{1+exp(-s)} \tag{3}$$

Input estimation algorithm of perceptron model is described as follows.

Preparation

[1] Assume the perceptron model has n -input items and m -output items.

[2] Define the evaluation function as

$$E(\mathbf{s}) = \frac{1}{2}\sum_{j=1}^{m}\{y_j - u_j^l(\mathbf{s})\}^2 \tag{4}$$

where $\mathbf{s} = \left[s_1, s_2, s_3, \cdots s_n\right]$ *and* y_j is the j-th output value of the perceptron.

Algorithm

[1] Set the initial input s as an arbitrary value.

[2] Calculate output

$$\mathbf{u} = \left[u_1, u_2, u_3, \cdots u_m\right]$$

for the current input value s.

[3] Change the value s according to

$$\frac{d\mathbf{s}}{dt} = -e\frac{\partial E}{\partial \mathbf{s}}$$

$$= -e\left(\sum_{\cdot=1}^{m}\frac{\partial E}{\partial u_\cdot^l}\frac{\partial u_\cdot^l}{\partial s_\cdot^l}\sum_{\cdot\cdot}\frac{\partial s_\cdot^l}{\partial u_{\cdot\cdot}^{l-1}}\frac{\partial u_{\cdot\cdot}^{l-1}}{\partial s_{\cdot\cdot}^{l-1}}\cdots\sum_{\cdots\cdot=1}^{n}\frac{\partial s^2{}_{\cdot\cdot}}{\partial u_{\cdots\cdot}^1}\frac{\partial u^1{}_{\cdots\cdot}}{\partial s_{\cdots\cdot}^1}\right) \tag{5}$$

where e is a positive value.

[4] Repeat [2]-[3] until the value E attains a sufficiently small value.

[5] Final value s is the estimated value by the perceptron.

Fuzzy Inverse Problem

Fuzzy relation R between set X and set Y is regarded as a fuzzy set on the direct product of X and Y , and its membership function is denoted by

$$\mu_R : X \times Y = \{(x,y) \mid x \in X, y \in Y\} \tag{6}.$$

Assume that A is a fuzzy set on X and B is another fuzzy set on Y , where these membership functions are μ_A and μ_B respectively, then fuzzy relation R satisfies (7) which means (8).

$$B = R \circ A \tag{7}$$

$$\mu_B(y) = \max_{x \in X}\{\mu_R(x,y) \wedge \mu_A(x)\} \tag{8}$$

If A and B are fuzzy input and output, respectively, then (7) is interpreted as an equation of the system which has fuzzy input and output.

Then the fuzzy inverse problem is the inverse problem of fuzzy relational equation, i. e. identifying A for the given B and R in the equation (7).

Method of Solving Fuzzy Inverse Problem by using Perceptron Model

This method is divided into two phases , i. e. the learning phase and the solving phase. The learning phase is summarized as follows;

[1] Let A_i and B_i be the i -th input and output of fuzzy relation system R (i = 1,2, ... M) , respectively.

[2] Encode A_i and B_i to $\mathbf{x}_i$ and $\mathbf{y}_i$,respectively, according to

$$[0,1] \ni a_i \mid \rightarrow x_i = 2.2 \times a_i - 1.1 \in [-1.1, 1.1] \tag{9}$$

$$[0,1] \ni b_i \mid \rightarrow y_i = 0.8 \times b_i + 0.1 \in [0.1, 0.9] \tag{10}.$$

[3] Let multi layer perceptron be learned by using input-output pair $\mathbf{x}_i$ and $\mathbf{y}_i$ obtained at [2].

After finishing this learning phase, we can get the solution as follows;

[4] Let B be a fuzzy set to be solved in the fuzzy inverse problem.

[5] Encode B to y by using eq. (10).

[6] Apply the input estimation algorithm to the learned perceptron and estimate the input for given output y.

[7] Decode y to A by using eq. (9), then we can get the solution of the problem.

Numerical Experiment

To investigate the availability of the method discussed in the previous section, a numerical experiment on a digital computer has been done as follows;

[1] Fuzzy relation, i. e. the solution in the fuzzy inverse problem, is given as (11) in this experiment.

$$R = \begin{bmatrix} \mu_R(x_1,y_1) & \mu_R(x_2,y_1) & \cdots & \mu_R(x_5,y_1) \\ \mu_R(x_1,y_2) & \mu_R(x_2,y_2) & \cdots & \mu_R(x_5,y_2) \\ \mu_R(x_1,y_3) & \mu_R(x_2,y_3) & \cdots & \mu_R(x_5,y_3) \\ \mu_R(x_1,y_4) & \mu_R(x_2,y_4) & \cdots & \mu_R(x_5,y_4) \\ \mu_R(x_1,y_5) & \mu_R(x_2,y_5) & \cdots & \mu_R(x_5,y_5) \end{bmatrix} = \begin{bmatrix} 0.6 & 0.5 & 0.8 & 0.3 & 0.2 \\ 0.4 & 0.1 & 0.9 & 0.6 & 0.4 \\ 0.1 & 0.1 & 0.9 & 0.8 & 0.5 \\ 0.9 & 0.2 & 0.9 & 0.1 & 0.5 \\ 0.4 & 0.5 & 0.3 & 0.8 & 0.9 \end{bmatrix} \tag{11}$$

[2] The learning data for the perceptron is generated as follows;

i) Make fuzzy sets A_1 - A_{7776} whose membership values at each element take all combination of values {0, 0.2, 0.4, 0.6, 0.8, 1.0} (c.f. 6^5 = 7776).

ii) Operate each A_i to the fuzzy relation R by using max-min composition, then we can get the fuzzy set B_i.

iii) Encode fuzzy sets A_i and B_i by eqs' (9) and (10), then we will obtain the learning data $\mathbf{x}_i$ and $\mathbf{y}_i$.

[3] Let's move to the learning phase by using $\mathbf{x}_i$ and $\mathbf{y}_i$ as leaning data. The structure of perceptron used in this experiment is shown as in table 1.

table 1 : Outline of the Perceptron

layer number	1	2	3	4
number of neurons	5	10	10	5

Error back propagation algorithm[4] is used for learning. Counting the learning process is defined as follows : one learning process is a learning operation for a paired input-output data. The perceptron used in this experiment learns about 500 thousand times. The distribution of evaluation function (4) value for learning data is shown in figure 1.

[4] The test-data in the fuzzy inverse problem is made as follows;

i) Make fuzzy sets A_i whose membership values at each element are random values from [0,1] (where i = 1 - 1000 in this experiment). Get B_i by compositing A_i to the fuzzy relation R.

[5] Encode B_i by eq. (10), and get y_i. Applying input estimation algorithm to the perceptron, we can get estimated input value x^*_i. Then we get the solution of fuzzy inverse problem by decoding x^*_i to A^*_i by eq. (9).

[6] Composite the obtained solution A^*_i to fuzzy relation R , and we get B^*_i. The correctness of solution is considered as the closeness of membership value by each element between B^*_i and B_i.

[7] Distribution of the values of evaluation functions (12)-(14) for all test-data are shown in figure 2 - figure 4 .

$$E_{mean}=\Sigma|b^*_j-b_j| \tag{12}$$

$$E_{max}=Max\,|b^*_j-b_j| \tag{13}$$

$$E_{min}=Min\,|b^*_j-b_j| \tag{14}$$

$$for\ all\ j\ ,\ b_j\in B_i, b^*_j\in B^*_i$$

Discussion

The availability of using perceptron for fuzzy inverse problem was shown trough a numerical experiment in previous chapter. The approximate solution of the fuzzy inverse problem is obtained , but its precision is not enough. This is mainly because that the error of approximation of fuzzy relation by perceptron is not small enough.

Distribution of approximation error of fuzzy relation was shown in figure 1 in the previous chapter, but the inputs for error measuring are the same one as learning inputs, precisely. It is necessary to measure the error for no learning inputs. So the distribution of evaluation function (4) for no learning inputs is shown in figure 5 . By comparing error in learning and no learning cases, it should be noted that the distributions have the same shape. It depends on the generalization factor of neural networks. Hence it could be considered that the differences between learning and no learning are independent with the precision of solution.

The distribution of figure 1 and figure 5 are *bell* shaped but not *exponential*. This is considered that the approximation error is not less than a certain threshold value. This is because a confliction occurs between one learning input-output pair and others. Therefore, if we can increase the learning times or use more input-output pairs for learning, the improvement of precision for approximating fuzzy relation is not expected.

Now let's discuss how small the approximation error of fuzzy relation is. The measurement was done as follows; Let change membership value of fuzzy set A at one element in the interval [0,1], and compare the membership value of fuzzy set B obtained by compositing fuzzy relation R in eq. (11) and another one obtained from the output of perceptron. The membership values used for comparison are shown in table-2 and the results are shown in figure 6 - figure 10. In these figures, the horizontal axis represents membership value of a variable element of fuzzy set A , and vertical axes represent membership values of each element of fuzzy set B. There exist two lines in these graphs, where one (which has linear shape) represents the characteristics of approximated fuzzy relation, another (which has not linear shape) indicates the characteristic of perceptron which approximates the fuzzy relation.

table 2 : Conditions of membership value for fuzzy set A						
No.	element number					figure number
	1	2	3	4	5	
1	[0,1]	0.3	0.5	0.2	0.7	fig.6
2	0.3	[0,1]	0.2	0.6	0.4	fig.7
3	0.2	0.6	[0,1]	0.4	0.7	fig.8
4	0.8	0.7	0.2	[0,1]	0.5	fig.9
5	0.2	0.9	0.3	0.4	[0,1]	fig.10

Conclusion

Input estimation algorithm of perceptron model has been proposed and applied to the fuzzy inverse problem. Numerical experiments have been done in order to get the solution of fuzzy inverse problem. The precision of approximate solution obtained by this method is discussed, and the approximation error distribution is investigated. From the results of these numerical experiments, it is concluded that this method is available to obtain the approximate solution of fuzzy inverse problem.

References

1. E.Sanchez, *Resolution of Composite Fuzzy Relation Equations*, Information and Control, 30(1), pp.38-48(1976).
2. Tukamoto Y., Tasiro T., *Method of Solution to Fuzzy Inverse Problem*, Transactions of the Society of Instrument and Control Engineers, vol 15(1), pp. 21-25(1977) (in Japanese)
3. F.Rosenblatt, *The perceptron : A probabilistic model for information sorage and organization in the brain.* Psychol. Rev. 65(6), pp. 386-408(1958).
4. Runmelhart et al., *Learning Representations by Back-propagating errors*, Nature 323-9, pp.533-536(1986)

Table 1. Outline of the Perceptron

Outline of the Perceptron				
layer number	1	2	3	4
number of neurons	5	10	10	5

Table 2. Conditions of membership value for fuzzy set A

Conditions of membership value for fuzzy set A						
No.	element number					figure number
	1	2	3	4	5	
1	[0,1]	0.3	0.5	0.2	0.7	fig.6
2	0.3	[0,1]	0.2	0.6	0.4	fig.7
3	0.2	0.6	[0,1]	0.4	0.7	fig.8
4	0.8	0.7	0.2	[0,1]	0.5	fig.9
5	0.2	0.9	0.3	0.4	[0,1]	fig.10

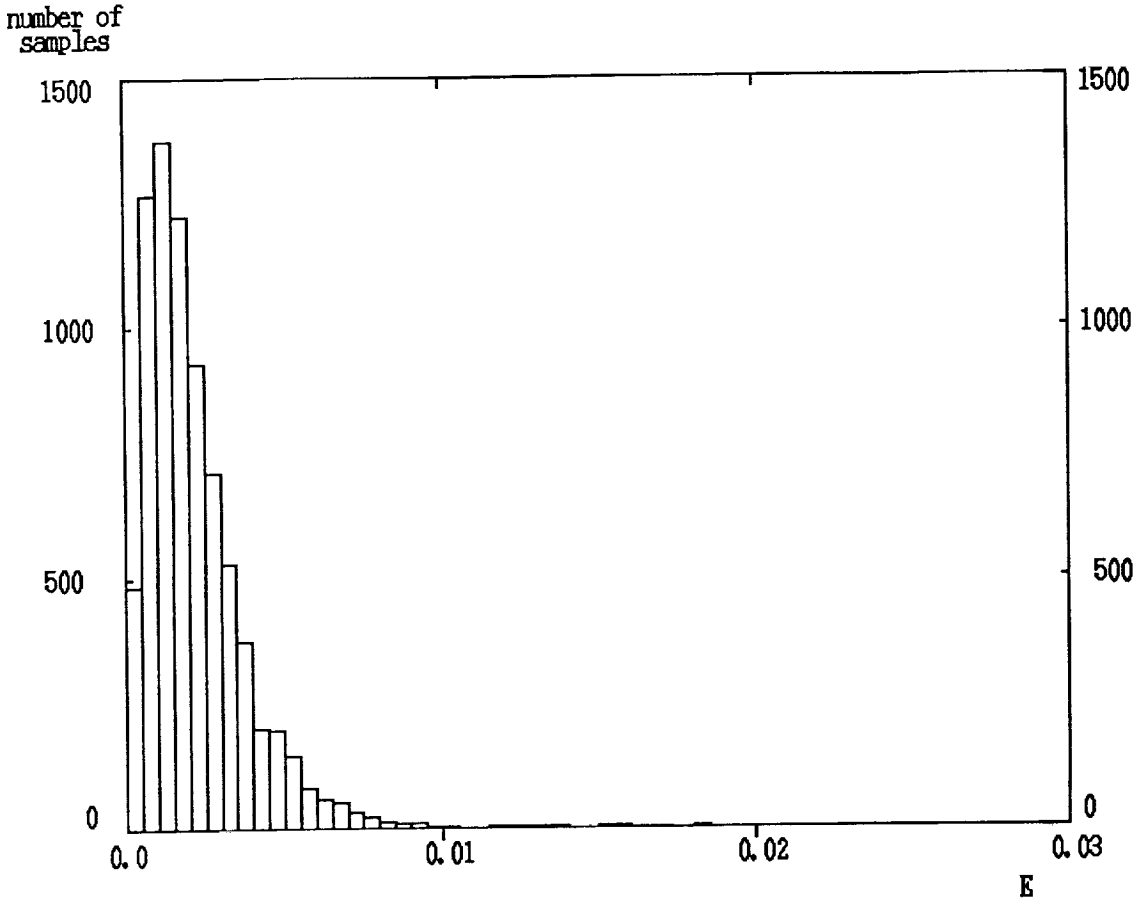


Figure 1. Distribution of evaluation function E for learning data

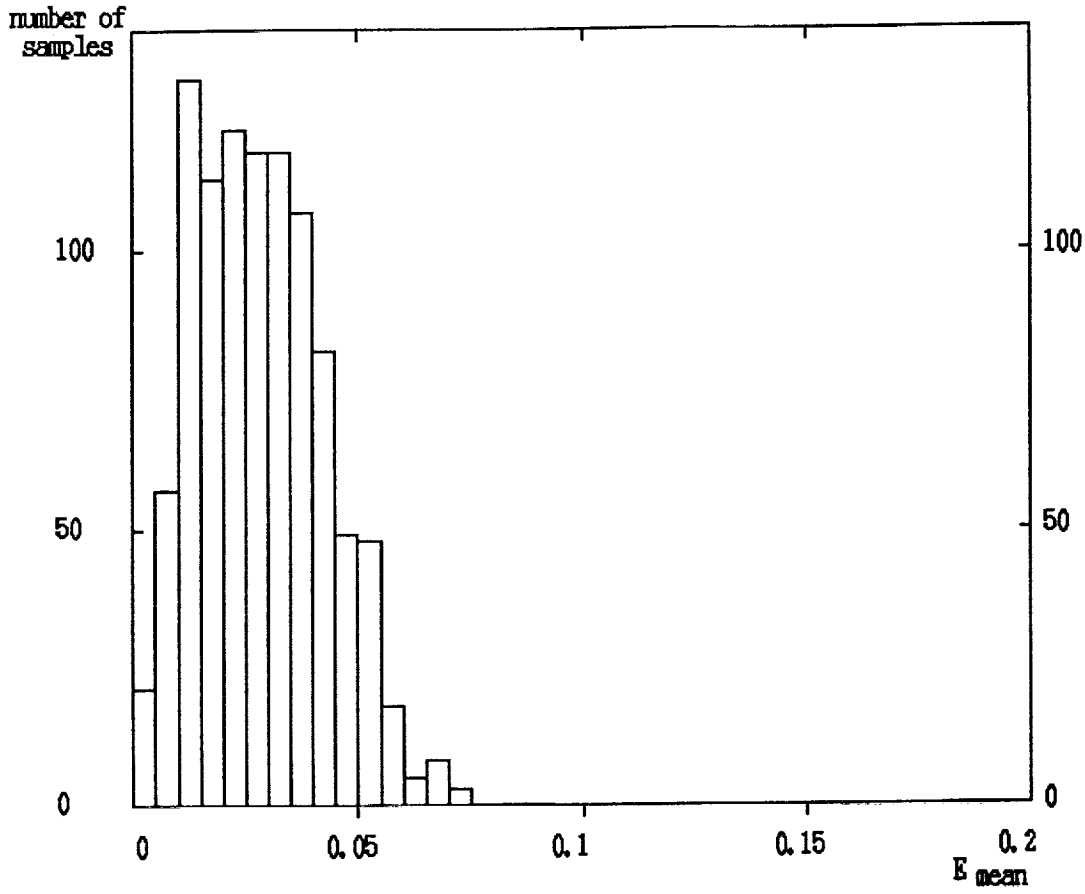


Figure 2. Distribution of evaluation function E_{mean} for meaning evaluation

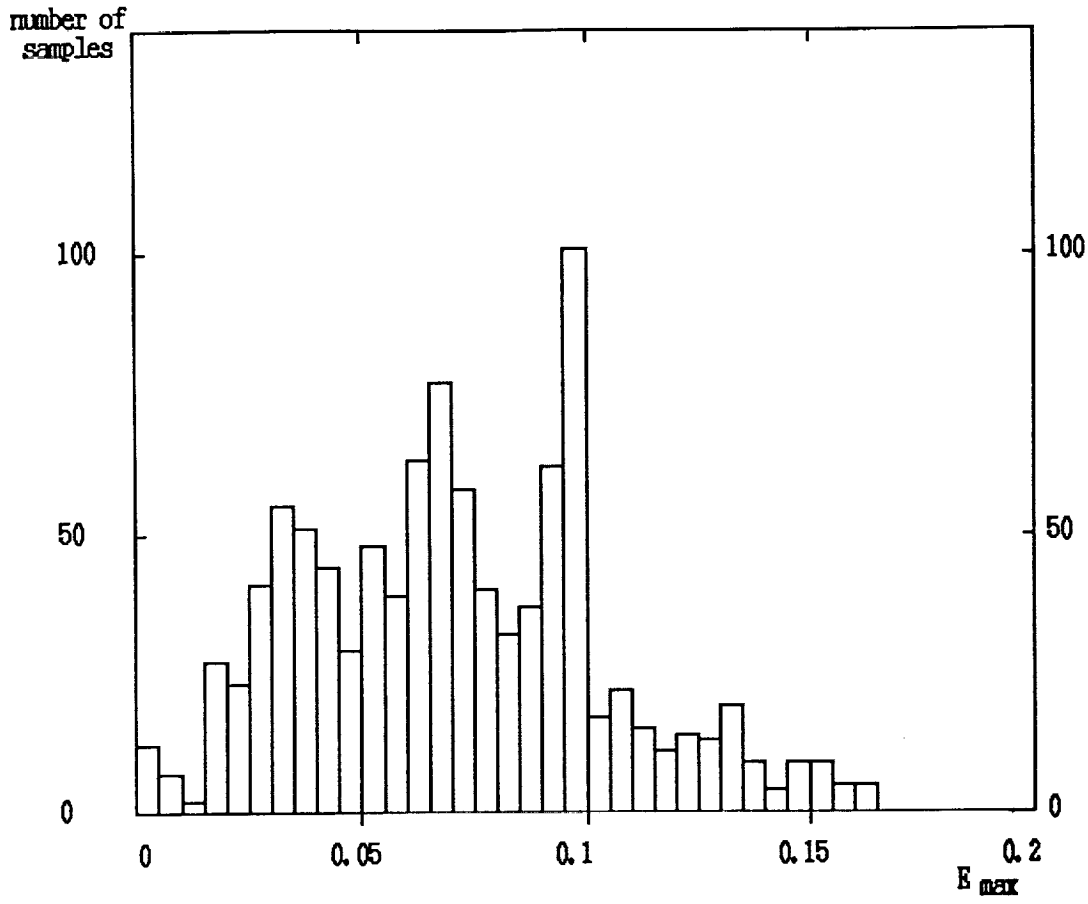


Figure 3. Distribution of evaluation function E_{max} for maximum evaluation

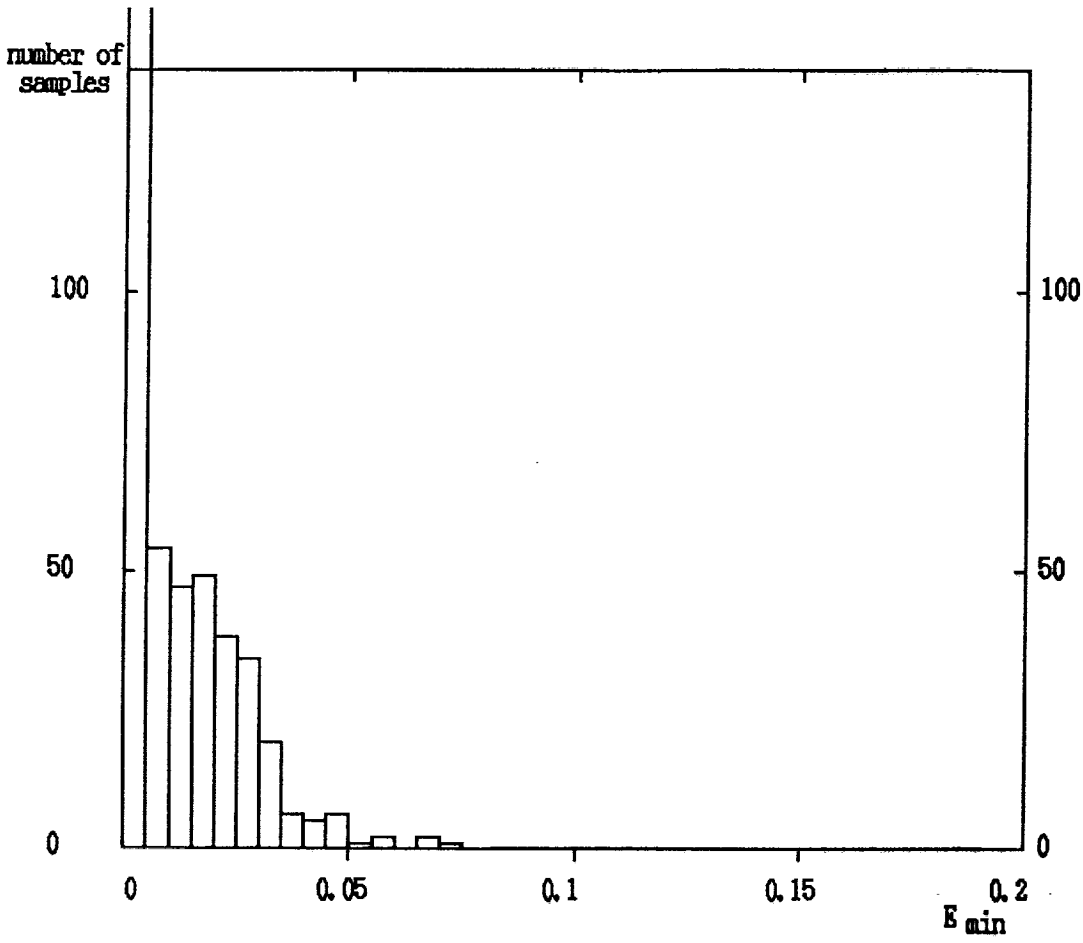


Figure 4. Distribution of evaluation function E_{min} for minimum evaluation

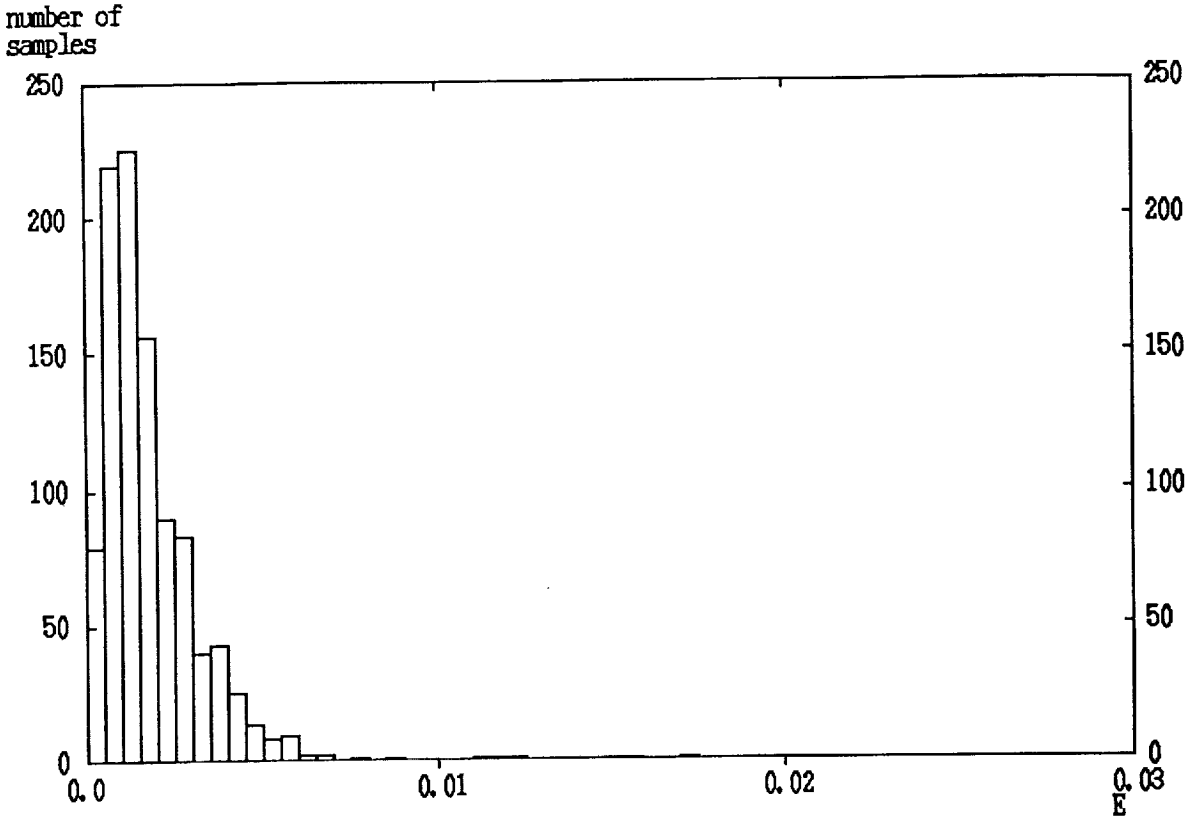


Figure 5. Distribution of evaluation function E for no learning data

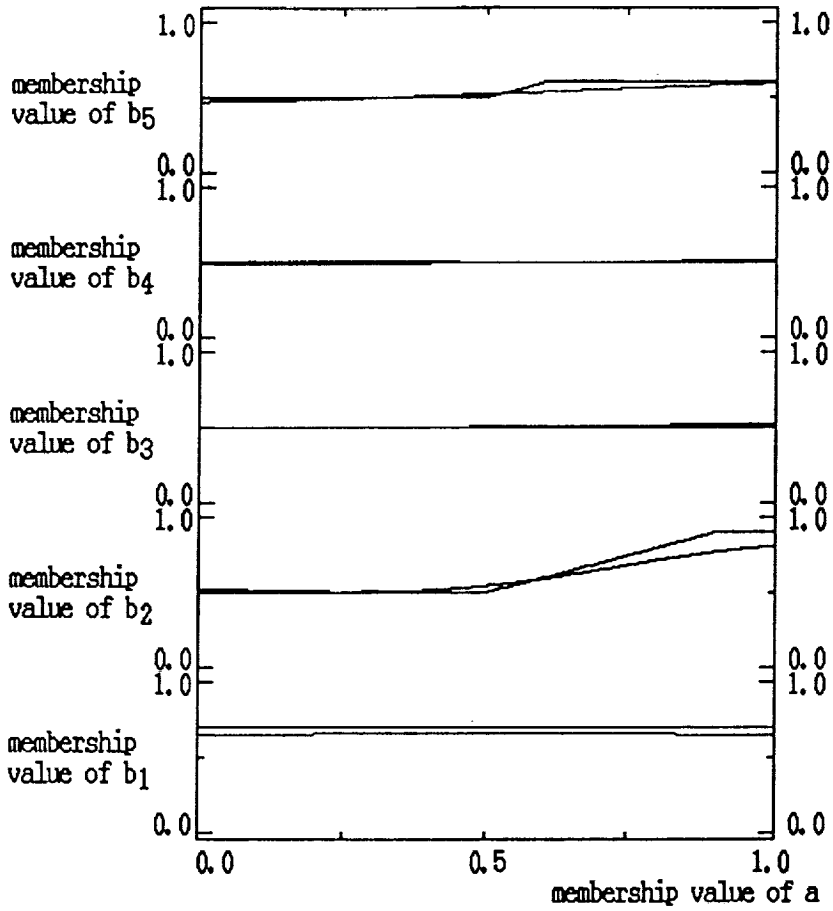


Figure 6. Comparison of fuzzy set B and output of perceptron (No.1)

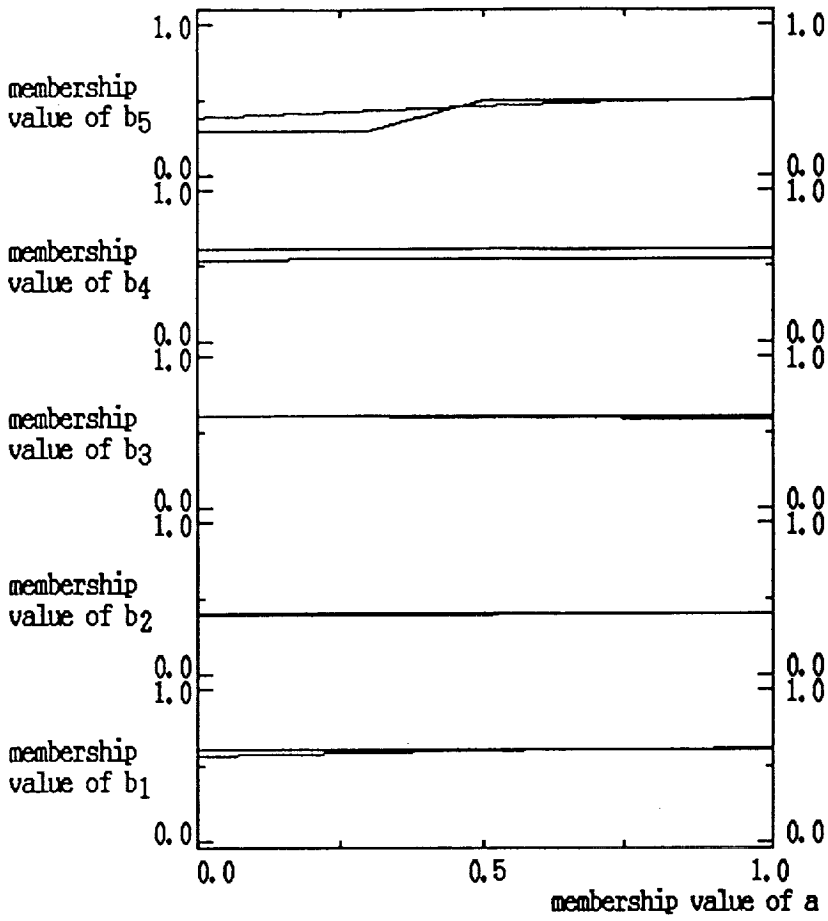


Figure 7. Comparison of fuzzy set B and output of perceptron (No.2)

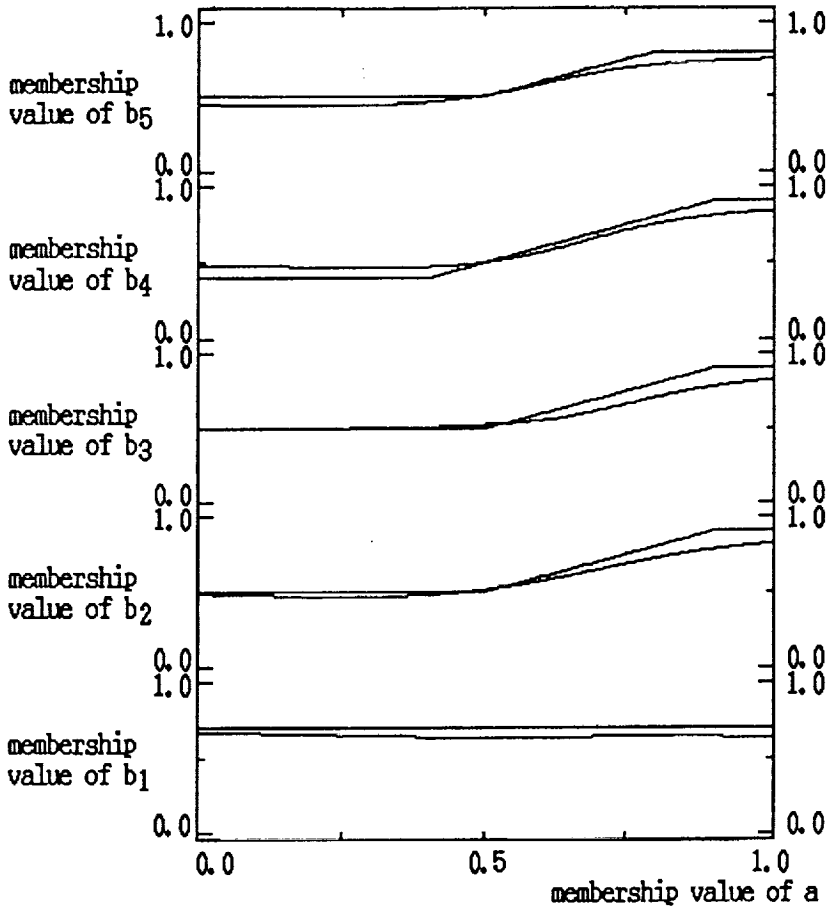


Figure 8. Comparison of fuzzy set B and output of perceptron (No.3)

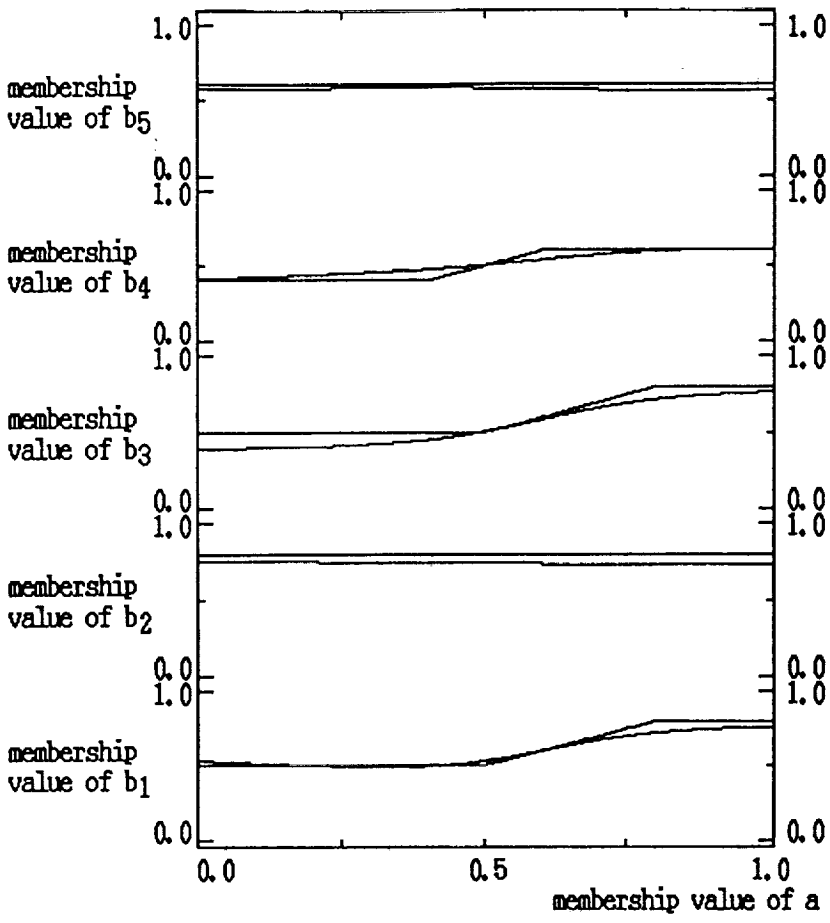


Figure 9. Comparison of fuzzy set B and output of perceptron (No.4)

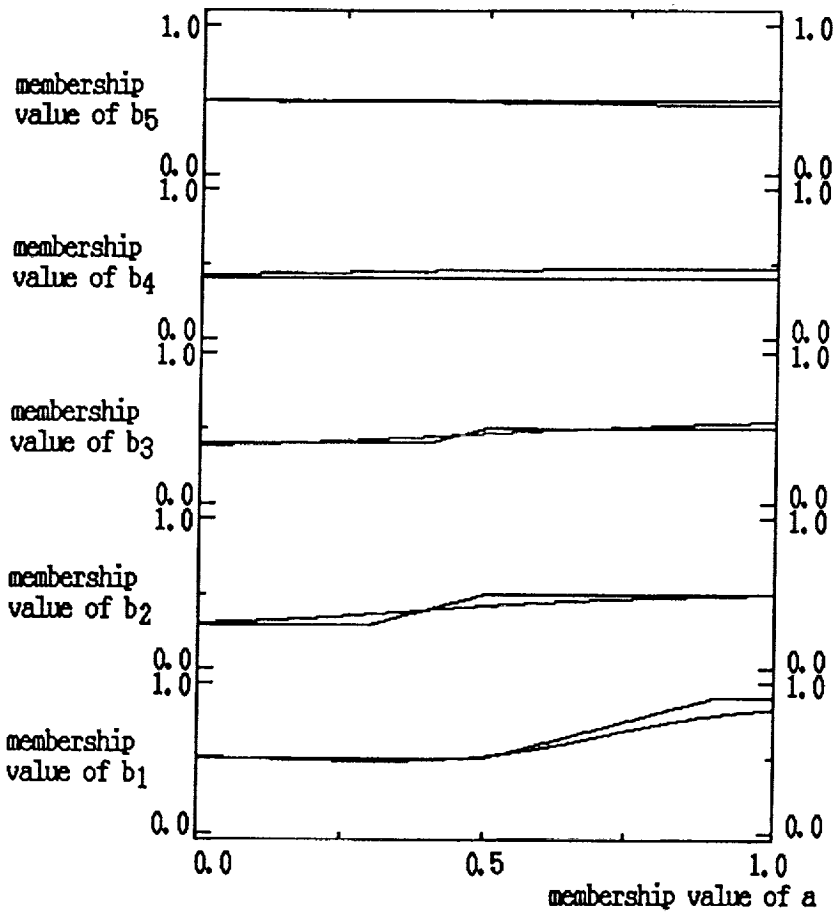


Figure 10. Comparison of fuzzy set B and output of perceptron (No.5)

Overview of LIFE (Laboratory for International Fuzzy Engineering) Research

(Paper not provided by publication date)

EXPERIMENTS ON NEURAL NETWORK ARCHITECTURES FOR FUZZY LOGIC

James M. Keller
Electrical and Computer Engineering
University of Missouri - Columbia
Columbia, MO 65211

ABSTRACT

The use of fuzzy logic to model and manage uncertainty in a rule-based system places high computational demands on an inference engine. In an earlier paper, we introduced a trainable neural network structure for fuzzy logic. These networks can learn and extrapolate complex relationships between possibility distributions for the antecedents and consequents in the rules. In this paper, the power of these networks are further explored. The insensitivity of the output to noisy input distributions (which are likely if the clauses are generated from real data) is demonstrated as well as the ability of the networks to internalize multiple conjunctive clause and disjunctive clause rules. Since different rules (with same variables) can be encoded in a single network, this approach to fuzzy logic inference provides a natural mechanism for rule conflict resolution.

1. INTRODUCTION.

In dealing with automated decision making problems, and computer vision in particular, there is a growing need for modeling and managing uncertainty. Computer vision is beset with uncertainty of all types. A partial list of the causes of such uncertainty include:

complexity of the problems,
questions which are ill-posed,
vagueness of class definitions,
imprecisions in computations,
noise of various sorts,
ambiguity of representations, and
problems in scene interpretation.

Rule-based approaches for handling these problems have gained popularity in recent years [1-6]. They offer a degree of flexibility not found in traditional approaches. The systems based on classical (crisp) logic need to incorporate, as an add-on, the processing of the uncertainty in the information. Methods to accomplish this include heuristic approaches [7, 8], probability theory [9,10], Dempster-Shafer belief theory [4,5,11], and fuzzy set theory [5,6,12-14].

Fuzzy logic, on the other hand, is a natural mechanism for propagating uncertainty explicitly in a rule base. All propositions are modeled by possibility distributions over appropriate domains. For example, a computer vision system may have rules like

IF the range is LONG, THEN
the prescreener window size is SMALL;

or

IF the color is MOSTLY RED, THEN
the steak is MEDIUM RARE is TRUE.

Here, LONG, SMALL, MOSTLY RED and TRUE are modeled by fuzzy subsets over appropriate domains of discourse. The possibility distributions can be generated from various histograms of feature data extracted from images, fuzzification of values produced by pattern recognition algorithms, experts expressing (free form) opinions on some questions, or possibly generated by a neural network learning algorithm.

The generality inherent in fuzzy logic comes at a price. Since all operations involve sets, rather than numbers, the amount of calculations per inference rises dramatically. Also, in a fuzzy logic system, generally more rules can be fired at any given instant. One approach to combat this computational load has been the development of special purpose

chips which perform particular versions of fuzzy inference [15]. Artificial Neural Networks offer the potential of parallel computation with high flexibility. In an earlier paper [16], we introduced a backpropagation neural network structure to implement fuzzy logic inference. In this paper we demonstrate further properties of that network. In particular, we show the insensitivity of the networks to noisy input distributions and to their ability to internalize rules with multiple conjunctive and disjunctive antecedent clauses.

2. FUZZY LOGIC AND NEURAL NETWORKS.

The original fuzzy inference mechanism extended the traditional modus ponens rule which states that from the propositions

P_1: If **X is A** Then **Y is B**

and P_2: **X is A,**

we can deduce **Y is B**. If proposition P_2 did not exactly match the antecedent of P_1, for example, **X is A'**, then the modus ponens rule would not apply. However, in [17], Zadeh extended this rule if A, B, and A' are modeled by fuzzy sets, as suggested above. In this case, P_1 is characterized by a possibility distribution:

$$\prod_{(x|y)} = R \text{ where}$$

$$\mu_R(u,v) = \max\{(1-\mu_A(u)),\ \mu_B(v)\}.$$

It should be noted that this formula corresponds to the statement "not A or B", the logical translation of P_1. An alternate translation of the rule P_1 which corresponds more closely to multivalue logic is

$$\mu_R(u,v) = \min\{1,\ \{(1 - \mu_A(u)) + \mu_B(v)\}\},[17],$$

called the bounded sum.

In either case, Zadeh now makes the inference **Y is B'** from μ_R and $\mu_{A'}$ by

$$\mu_{B'}(v) = \max_u \{\min\{\mu_R(u,v), \mu_{A'}(u)\}\}.$$

This is called the compositional rule of inference.

While this formulation of fuzzy inference directly extends modus ponens, it suffers from some problems [18,19]. In fact, if proposition $\mathbf{P_2}$ is **X is A**, the resultant fuzzy set is not exactly the fuzzy set B. Several authors [18-20] have performed theoretical investigations into alternative formulations of fuzzy implications in an attempt to produce more intuitive results.

In using fuzzy logic in real rule-based systems, the possibility distributions for the various clauses in the rule base are normally sampled at a fixed number of values over their respective domains of discourse, creating a vector representation for the possibility distribution. Table I shows the sampled versions of the "trapezoidal" possibility distributions, used in the simulation study, sampled at integer values over the domain [1,11]. Clearly , the sampling frequency has a direct effect on the faithfulness of the representation of the linguistic terms under consideration and also on the amount of calculation necessary to perform inference using a composition rule. For a single antecedent clause rule, the translation becomes a two dimensional matrix and the inference is equivalent to maxtrix-vector multiplication. As the number of antecedent clauses increases, the storage (multidimensional matrices) and the computation in the inference process grows exponentially.

Neural network structures offer a means of performing these computations in parallel with a compact representation. But the ability of such a network to generalize from an existing training set is the most valuable feature. In [16], we introduced the neural network architecture for fuzzy logic. Figure 1 displays a three layer feed-forward neural network which is used in fuzzy logic inference for conjunctive clause rules. It consisted of an input layer to receive the possibility distributions of the antecedent clauses, one hidden layer to internalize a representation of the relationships, and an output layer to produce the possibility distributions of the consequent.

The input layer is not fully connected to the hidden layer. Instead, each antecedent clause has its own set of hidden neurons to learn the desired relationship. This partitioning of the hidden layer was done to ease the training burden for multiple clause rules, and to treat each input clause with its hidden units as a functional block. The training was performed using the standard back propagation technique [21].

3. EXPERIMENTS.

The neural network architecture performed very well in generalizing the complex relationships between inputs and outputs. Table II (from [16]) shows the results of the training and testing of a network to implement the rule: IF X is LOW Then Y is HIGH; whereas Table III gives the situation for a rule with two conjunctive antecedent clauses. In both cases, the performance of the networks matched our intuitive expectation.

Figure 2 shows typical responses of a neural network to noise in the input clause. It can be seen that the errors in the result are of the same order as the error in the input . If the networks are trained with fewer relationships, e.g. the traditional modus ponens

expectations, this error drops significantly.

In order to implement rules with disjunctive antecedent clauses, networks with two hidden layers were necessary. Table IV displays training relationships for a two clause disjunctive rule. Note that there are 23 input/output triples necessary to enable the network to respond appropriately. The training, using backpropagation, of a single hidden layer network, of the type shown in figure 1, failed to converge on this complex training set. This caused us to investigate a two hidden layer structure where the first hidden layer was the same as in figure 1 and the second hidden layer contained 6 neurons totally connected to those of the first hidden layer and to the nodes of the output layer. This network converged in 4073 passes through the training set with a total-sum-of-squared error of less than 0.001 for the entire training ensemble. We feel that this is a remarkable achievement, given the diversity of the responses to the antecedent possibility distributions which were necessary.

This disjunctive structure was further tested with 18 input pairs of clauses including twelve pairs with varying amounts of additive gaussian noise. For this test set the average total-sum-of-squared-error per trial was 0.075. In other words, the match to the expected output in all cases was very good.

As a final note, in [16] we demonstrated that a neural network structure of this type could encode multiple different rules which shared common antecedent clause variables. The packing of several rules into a single network has a surprising side benefit of providing a natural means of conflict resolution in fuzzy logic.

4. CONCLUSION.

Fuzzy logic is a powerful tool for managing uncertainty in rule-based systems. Neural network architectures offer a means of relieving some of the computational burden inherent in fuzzy logic. Also, these structures can be trained to learn and extrapolate complex relationships between antecedents and consequents, they are relatively insensitive to noise in the inputs, and provide a natural mechanism for conflict resolution.

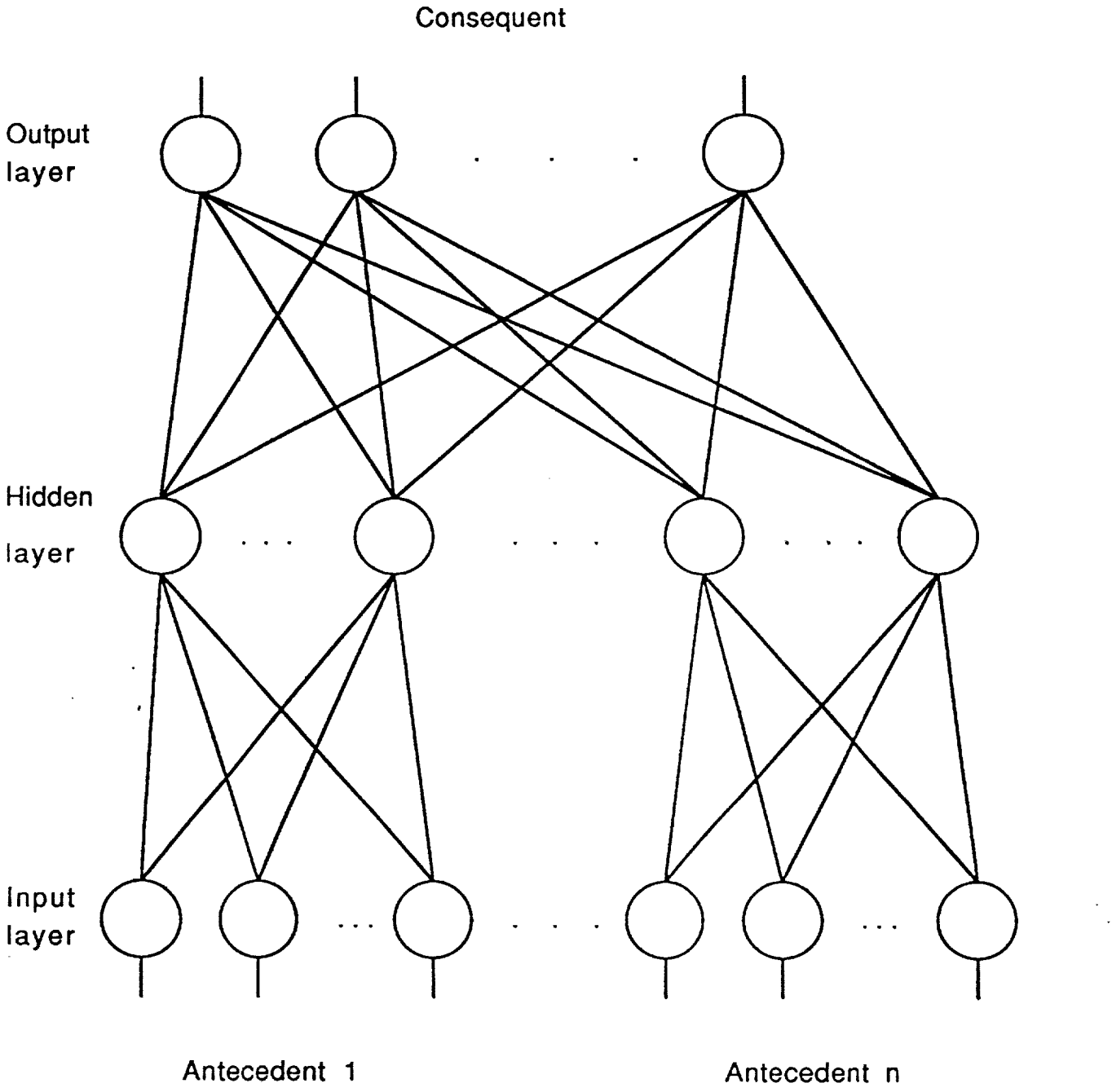
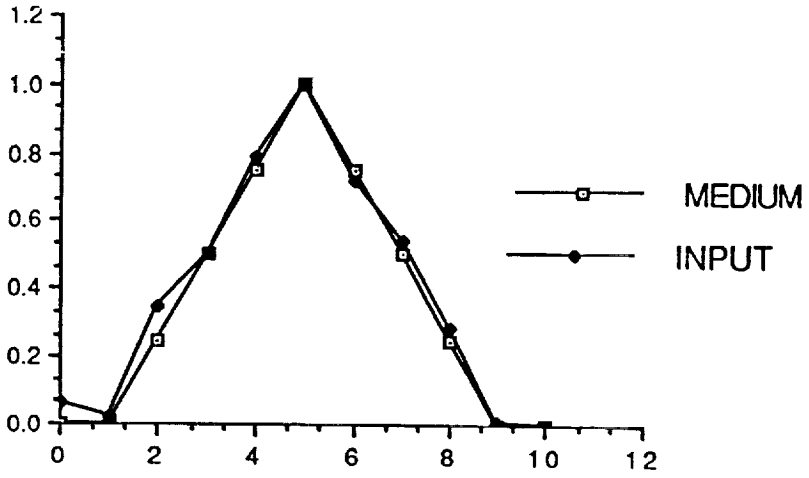


Figure 1. A three layer feed forward neural network for fuzzy logic inference

Rule: IF X is MEDIUM THEN Y is HIGH

MEDIUM	.00	.00	.25	.50	.75	1.0	.75	.50	.25	.00	.00
INPUT	.06	.02	.35	.50	.79	1.0	.72	.54	.29	.01	.00

TSS error = 0.020



HIGH	.00	.00	.00	.00	.00	.00	.20	.40	.60	.80	1.0
OUTPUT	.00	.00	.00	.00	.00	.00	.28	.48	.67	.84	1.0

TSS error = 0.019

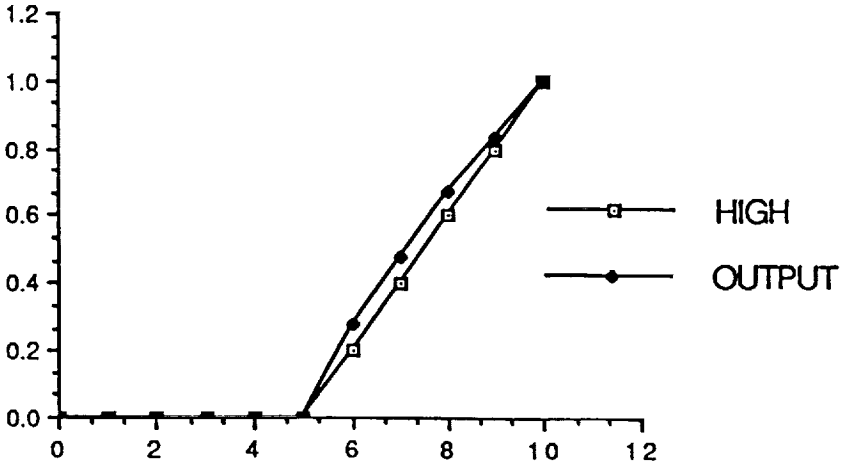
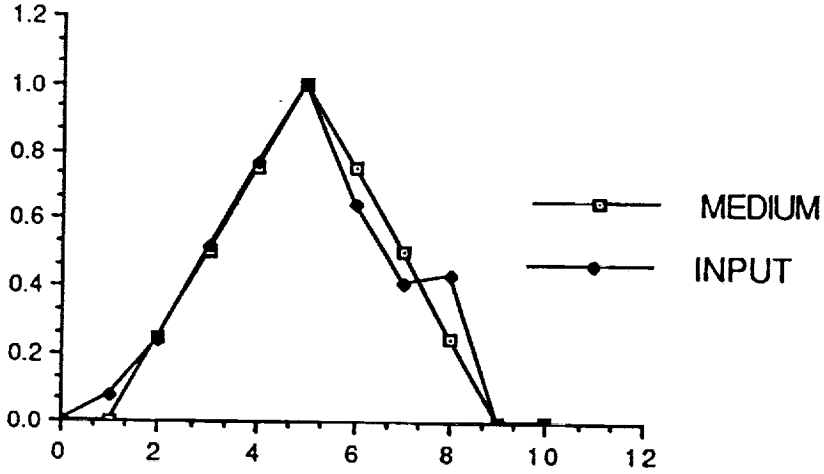


Figure 2(a) Response of rule network to an input with small amount of additive gaussian noise.

MEDIUM	.00	.00	.25	.50	.75	1.0	.75	.50	.25	.00	.00
INPUT	.00	.08	.24	.52	.77	1.0	.64	.41	.43	.00	.00

TSS error = 0.060



HIGH	.00	.00	.00	.00	.00	.00	.20	.40	.60	.80	1.0
OUTPUT	.00	.00	.00	.00	.00	.00	.25	.46	.65	.83	1.0

TSS error = 0.010

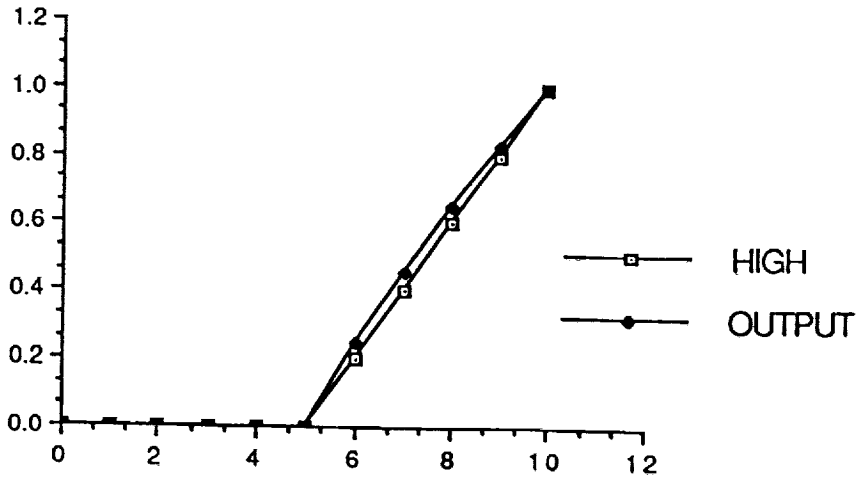


Figure 2(b) Response of rule network to an input with a larger amount of additive gaussian noise.

Table I. The meaning of linguistic terms defined on the domain [1,11] and sampled at integer points.

Label	Membership										
LOW	1.00	0.67	0.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
VERY LOW	1.00	0.45	0.11	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
MORL LOW	1.00	0.82	0.57	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
NOT LOW	0.00	0.33	0.67	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
NOISY LOW (1)	1.00	0.70	0.40	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
NOISY LOW (2)	1.00	0.70	0.30	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
NOISY MEDIUM	0.00	0.00	0.30	0.53	0.81	1.00	0.80	0.50	0.20	0.00	0.00
SHIFTED LOW	1.00	1.00	1.00	0.67	0.33	0.00	0.00	0.00	0.00	0.00	0.00
MEDIUM	0.00	0.00	0.25	0.50	0.75	1.00	0.75	0.50	0.25	0.00	0.00
MORL MEDIUM	0.00	0.00	0.50	0.71	0.87	1.00	0.87	0.71	0.50	0.00	0.00
NOT MEDIUM	1.00	1.00	0.75	0.50	0.25	0.00	0.25	0.50	0.75	1.00	1.00
HIGH	0.00	0.00	0.00	0.00	0.00	0.00	0.20	0.40	0.60	0.80	1.00
VERY HIGH	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.16	0.36	0.64	1.00
MORL HIGH	0.00	0.00	0.00	0.00	0.00	0.00	0.45	0.63	0.77	0.89	1.00
UNKNOWN	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

MORL = more or less.

Note: Very^{n}A is determined by $\mu_{\text{Very}^n A}(x) = \mu_A(x)^{n+1}$

MORL^{n}A is determined by $\mu_{\text{MORL}^n A}(x) = [\mu_A(x)]^{1/n+1}$

Table II. Performance of Fuzzy Logic Rule network with 8 hidden neurons for rule **IF X is LOW THEN Y is HIGH.**

A. Training Data*

Input	Output
LOW	HIGH
VERY LOW	VERY HIGH
MORL LOW	MORL HIGH
NOT LOW	UNKNOWN

* Training terminated when the total sum of squared error dropped below $\epsilon = .001$

B. Testing Results

Input	Expected Output	Actual Output											Total Sum Squared Error
VERY^2 LOW	VERY^2 HIGH	.00	.00	.00	.00	.00	.00	.03	.10	.27	.56	1.0	.007
MORL^2 LOW	MORL^2 HIGH	.00	.01	.01	.01	.00	.01	.56	.71	.82	.91	1.0	.030
MEDIUM	UNKNOWN	.99	.99	.99	.99	.99	.99	.99	.99	.99	.99	1.0	.001
VERY MEDIUM	UNKNOWN	.98	.98	.98	.98	.98	.98	.99	.99	.99	.99	1.0	.003
MORL MEDIUM	UNKNOWN	.99	.99	.99	.99	.99	.99	.99	.99	.99	.99	.99	.001
HIGH	UNKNOWN	.99	.99	.99	.99	.99	.99	.99	.99	.99	.99	.99	.001
NOISY LOW (1)	HIGH	.00	.00	.00	.00	.00	.00	.26	.47	.66	.83	1.0	.013
NOISY LOW (2)	HIGH	.00	.00	.00	.00	.00	.00	.19	.39	.59	.80	1.0	.0001
SHIFTED LOW	———	.09	.09	.12	.09	.09	.09	.91	.92	.94	.97	1.0	———

Table III. Performance of a two antecedent clause Fuzzy Logic Rule network with 16 hidden neurons (two groups of eight).

A. Training Data*

Input	Output
(LOW,MEDIUM)	HIGH
(VERY LOW,VERY MEDIUM)	VERY HIGH
(MORL LOW,MORL MEDIUM)	MORL HIGH
(NOT LOW,MEDIUM)	UNKNOWN
(LOW,NOT MEDIUM)	UNKNOWN

* Training converged in 1823 iterations.

B. Testing Results

Input	Actual Output											Closest Linguistic Term
(NOISY LOW(1),MEDIUM)	.00	.00	.00	.00	.00	.00	.20	.40	.60	.80	1.0	HIGH
(NOISY LOW(2),MEDIUM)	.00	.00	.00	.00	.00	.00	.19	.40	.60	.80	1.0	HIGH
($VERY^2$ LOW,MEDIUM)	.00	.00	.00	.00	.00	.00	.19	.38	.60	.80	1.0	HIGH
(NOISY LOW(1),NOISY MEDIUM	.00	.00	.00	.00	.00	.00	.20	.41	.61	.81	1.0	HIGH
(LOW,$VERY^2$ MEDIUM)	.00	.00	.00	.00	.00	.00	.05	.17	.36	.64	1.0	VERY HIGH
($VERY^2$ LOW,$VERY^2$ MEDIUM)	.01	.01	.01	.01	.01	.01	.03	.12	.29	.58	1.0	$VERY^2$ HIGH
($MORL^2$ LOW,$MORL^2$ MEDIUM)	.01	.01	.01	.01	.01	.01	.55	.70	.81	.91	1.0	$MORL^2$ HIGH
(NOT LOW,NOT MEDIUM)	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	UNKNOWN
(LOW,SHIFTED MEDIUM)	.97	.97	.97	.97	.97	.97	.99	.99	.99	1.0	1.0	UNKNOWN
(MEDIUM,LOW)	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	UNKNOWN

Table IV. Training Data for the two disjunctive clause rule:
IF X is LOW OR Y is MEDIUM THEN Z is HIGH.

Input	Output
(Very, MorL) LOW: *	(Very, MorL) HIGH
*; (Very, MorL) MEDIUM	(Very, MorL) HIGH
Not LOW; Not MEDIUM	UNKNOWN
MEDIUM; LOW	UNKNOWN
HIGH; LOW	UNKNOWN
HIGH; Very LOW	UNKNOWN
UNKNOWN, HIGH	UNKNOWN

* = LOW, MEDIUM, HIGH

Training converged in 4073 iterations, with TSS error for entire training set less than 0.001

5. REFERENCES.

1. Fikes, R., and Nilsson, N., "STRIPS: a new approach to the application of theorem proving to problem solving", Artificial Intelligence, 2, 3/4, 1971, pp. 189-208.

2. Barrow, H. and Tenenbaum, J. "MSYS: a system for reasoning about scenes", Technical Note 121, AI Center, SRI International, March 1976.

3. Brooks, R., Greiner, R., and Binford, T., "Progress report on a model-based vision system", Proc. Image Understanding Workshop, L. Baumann, ed., 1978, pp. 145-151.

4. Riseman, E., and Hanson, A, "A methodology for the development of general knowledge-based version systems", in Vision, Brain, and Cooperative Computation, M. Arbib and A. Hanson, Eds., MIT Press, Cambridge, MA, 1988, pp. 285-328.

5. Wootton, J. Keller, J., Carpenter, C., and Hobson, G., "A multiple hypothesis rule-based automatic target recognizer," in Pattern Recognition, Lecture Notes in Computer Science, Vol. 301, Kittler, J. ed., Springer-Verlag, Berlin, 315-324, 1988.

6. Nafarieh, A., and Keller, J. "A fuzzy logic rule-based automatic target recognizer", Int. J. Intell. Systems, accepted for publication, 1990.

7. Shortliffe, E., and Buchanan, "A model of inexact reasoning in medicine", Math Biosci, 23, 1975, pp. 351-379.

8. Cohen, P., Heuristic Reasoning About Uncertainty: An Artificial Intelligence Approach. Pitman Advanced Publishing Program, 1985.

9. Pearl, J., "Fusion propagation and structuring in belief networks", Art. Intell., 29, no. 3, 1986, pp. 241-288.

10. Cheeseman, P. "A method of computing generalized bayesian probability values for expert systems", Proc. Eight Int. J. Conf. on AI, Karlsrulie, West Germany.

11. Li, Z. "Uncertainty management is a pyramid vision system", Int. J. Approx. Reasoning, 3, no. 1, 1989, pp. 59-85.

12. Bonissone, P. and Tong, R. "Editorial: Reasoning with uncertainty in expert systems," Int. J. Man-Machine Studies, Vol. 22, 241-250, 1985.

13. Zadeh, L. "Fuzzy logic and approximate reasoning," Syntheses, Vol. 30, 407-428, 1975.

14. Keller, J., Hobson, G., Wootton, J. Nafarieh, A., and Leutkemeyer, K., "Fuzzy confidence measures in midlevel vision", IEEE T. Syst., Man, Cybern, 17, no. 4, 1987, pp. 676-683.

15. Togai, M. and Watanabe, H., "Expert system on a chip: an engine for real-time approximate reasoning", IEEE Expert, Fall 1986, pp. 55-62.

16. Keller, J. and Tahani, H. "Backpropagation neural networks for fuzzy logic", Info. Sciences, accepted for publication, 1990.

17. Zadeh, L. "The concept of a linguistic variable and its application to approximate reasoning," Information Sciences, Part 1, Vol. 8, pp. 199-249; Part 2, Vol. 8, pp. 301-357; Part 3, Vol. 9, pp. 43-80, 1975.

18. Nafarieh, A., "A new approach to inference in approximate reasoning and its application to computer vision," Ph.D. Dissertation, University of Missouri-Columbia, 1988.

19. Mizumoto, M., Fukami, S., and Tanaka, K., "Some methods of fuzzy reasoning," in Advances in Fuzzy Set Theory and Applications, Gupta, M., Ragade, R., and Yager, R., eds., North-Holland, Amsterdam, 1979, pp. 117-126.

20. Baldwin, J. and Guild, N. "Feasible algorithms for approximate reasoning using fuzzy logic," Fuzzy Sets and Systems, Vol. 3, 1980, pp. 225-251.

21. Rumelhart, D., McClelland, J., and the PDP Research Group, Parallel Distributed Processing, Vol. 1, MIT Press, Cambridge, MA, 1986.

Using Fuzzy Logic to Integrate Neural Networks and Knowledge-based Systems

John Yen
Department of Computer Science
Texas A&M University
College Station, TX 77843
(409) 845-5466

April 9, 1990

Abstract

Even though the technology of neural nets has been successfully applied to image analysis, signal processing, and pattern recognition, most real world problems are too complex to be solved purely by neural networks. Two important issues regarding the application of neural networks to complex problems are (1) the integration of neural computing and symbolic reasoning, and (2) the monitoring and control of neural networks. Most hybrid models attempt to integrate neural net and symbolic processing technologies at the level of basic data representation and data manipulation mechanisms. However, intrinsic differences in the low-level data processing of the two technologies limit the effectiveness of that approach. This paper discusses the role of fuzzy logic in a hybrid architecture that combines the two technologies at a higher, *functional* level. Fuzzy inference rules are used to make plausible inference by combining symbolic information with soft data generated by neural nets. Neural networks are viewed as modules that perform flexible classification from low-level sensor data. The symbolic system provides a global shared knowledge base for communications and a set of *control tasks* for object-oriented interface between neural network modules and the symbolic system. Fuzzy action rules are used to detect situations under which certain control tasks need to be invoked for neural network modules. The hybrid architecture, which supports communication and control across multiple cooperative neural nets through the use of fuzzy rules, enables the construction of modular, flexible, and extensible intelligent systems, reduces the effort for developing and maintaining such systems, and facilitates their application to complex real world problems that need to perform low-level data classification as well as high-level problem solving in the presence of uncertainty and incomplete information.

1 Introduction

Recent development of neural network technology has demonstrated many promising applications in the areas of pattern recognition, image processing, and speech recognition. However, most real world problems are too complex to be solved purely by current neural network technologies. This paper addresses two important issues regarding building complex intelligent computer systems based on neural networks.

1. *How to integrate neural computing with symbolic reasoning?*
   A complex application usually can benefit from a synergistic integration of neural computing and symbolic reasoning. For example, in anti-submarine warfare, one might like to combine signal processing results computed in a neural net with symbolic analyses of evidence such as database information (e.g., records of confirmed vessel departures from port) and extended inference procedures (e.g., hypotheses about plausible mission plans). Many other problems, ranging from speech and vision to space applications, share this property of needing synergy between neural nets and symbolic approaches.

2. *How to monitor and control the behavior of neural networks?*
   If one wishes to construct a real world application such as anti-submarine warfare using neural networks, it is crucial to have mechanisms for interpreting and reacting to the results produced by the neural nets, so that the overall system can cope with the rapidly changing and unanticipated situations. For example, after being activated by an input pattern, a bidirectional associative memory, or BAM[11], might converge to a pattern not belonging to the set of training patterns. This misclassification phenomenon can be caused by having overly similar or numerous training patterns. In either case, the BAM needs to be modified (i.e., certain training patterns need to be removed from the training set) to improve its performance. Therefore, the system needs a *controller* that oversees the behavior of the neural networks. A general mechanism that supports the control across multiple cooperative neural nets will enable the construction of modular, flexible, and extensible neural net systems, reduce the effort for developing and maintaining such systems, and facilitate their application to complex real world problems. The need of a higher-level system for evaluating the performance of neural networks has also been suggested by other researchers [14].

This paper discusses the role of fuzzy logic in integrating neural networks and symbolic systems and in supervising the behavior of neural networks. To do this, we propose a hybrid architecture that uses fuzzy logic to combine the two technologies at a higher,

functional level. Two types of fuzzy rules are supported by the architecture: fuzzy inference rules and fuzzy action rules. Fuzzy inference rules are used to assimilate the outputs of neural nets, which are often soft data [24], into the symbolic system. Fuzzy action rules are used to issue control tasks, which are implemented by methods in object-oriented programming, for activating, training, and modifying neural nets. Neural networks are viewed as modules that perform flexible classification. The symbolic system provides a global shared knowledge base for communications and a fuzzy rule interpreter for performing rule-based reasoning.

Most hybrid models attempt to integrate neural net and symbolic processing technologies at the level of basic data representation and data manipulation mechanisms. However, intrinsic differences in the low-level data processing of the two technologies limit the effectiveness of that approach. In contrast, our approach combines the two technologies at a higher, *functional* level. The symbolic system views neural networks as modules that (1) extend its reasoning capabilities into flexible classification and data associations, and (2) extend its learning capabilities into adaptive learning. Neural nets each view the symbolic system as providing a global shared memory for communications and a controller, built using fuzzy action rules, for activating, training, and monitoring them. Fuzzy inference rules are used to pass data between the two subsystems; and fuzzy action rules are used to pass action between the two.

The key features of the proposed architecture that will provide these desirable properties include the following:

1. Fuzzy rules can invoke neural nets for testing "soft" (fuzzy) conditions in their left-hand-sides.

2. Recognition of situations requiring actions on neural networks is accomplished via fuzzy action rules, whose actions are modified by the degree that the rules' conditions are matched.

3. Both high-level descriptions (e.g., input-output characterizations) and the behavior (e.g., performance evaluations) of neural networks will be modeled using a principled frame-based language.

4. The symbolic system will interact with neural nets through a set of generic functions called *control tasks*. Control tasks will be implemented using methods in object-oriented programming so that common methods can be shared, and specific methods can override general ones.

In the following sections, we first discuss the background of this work, then we describe the hybrid architecture with an emphasis on the features mentioned above. Finally, we summarize the benefits of our approach.

2 Background

2.1 Two Complementary Technologies: Neural Networks and Artificial Intelligence

Neural networks and symbolic reasoning are two complementary approaches for achieving the same goal: building autonomous intelligent systems. The major strengths of Neural Networks are their capabilities for performing flexible classification and adaptive learning. By automatically capturing similarities among training instances (i.e., adaptive learning), neural networks are often able to perform flexible classification. That is, when given input data which is similar, but not identical, to inputs upon which the system has been trained, the network generates output similar to the trained responses. Consequently, a trained neural network is able to classify data approximately even when that data is incomplete or noisy. Thus, while most AI systems cannot tolerate such data, neural networks promise a system whose performance *gracefully degrades* under those circumstances.

On the other hand, neural networks have several major weaknesses. They have trouble handling multiple instances of the same concept. Viewed as a pattern-matcher, they have trouble dealing with patterns containing variables. They tend to be specialized for a specific task. Solving complex tasks is likely to require cooperation between many neural networks, but managing their intercommunication is not well-understood. Control of the activation and learning behavior of these networks by higher-level modules is also not well-understood. Because their internal representation is in a form that cannot be comprehended by the user easily, it is hard to explain the rationale behind the output of neural networks. Although some of these problems have been addressed by neural network researchers (e.g., schema theory[2] addresses the first two issues), a neural net approach that addresses all these problems is yet to be developed. The goal of this research is to develop a comprehensive solution to these concerns using fuzzy logic and existing AI techniques.

Certain AI techniques suggest solutions to the problems illustrated above. Different instances of a concept are easily represented using frame-based knowledge representation systems. Variables often occur in patterns, which can be matched with data using a pattern matching facility. The notion of supporting many independent modules that communicate through a *global knowledge base* accessible to all modules is an idea central to many AI systems. For example, blackboard architectures maintain a data structure (the "blackboard") where all knowledge sources can post or retrieve information. Production system architectures also have a working memory that all productions match their conditions against and act upon. An AI system may also provide a higher-level controller, often called the meta-level architecture, that has knowledge about the lower-level

system and is able to control the lower-level system in various ways. The explanation capabilities of AI systems have been enhanced by explicitly representing problem solving strategies [15].

Our integration of AI capabilities with neural nets is designed to address these issues. In Section 2.2, we explain the concerns driving the design. In Section 3, we detail our approach.

2.2 Problems with Current Hybrid Approaches

Combining neural networks and AI is certainly not a new idea, but previous efforts have not addressed the important issues raised above. A number of researchers have used neural networks to reimplement AI techniques such as production systems and semantic networks [19, 7]. Work in this area mainly demonstrates what neural networks can do, not that their implementations are better than the conventional ones. Others have applied neural networks to expert systems, natural language understanding, and other areas that have mainly utilized conventional AI techniques[9]. Work in these first two categories applies current neural net technologies, rather than addressing weaknesses of neural nets. Furthermore, it has demonstrated neural net implementations of things that AI can easily handle, rather than things that AI has great difficulties in doing (e.g., partial matching). A few researchers have introduced ideas from neural networks into conventional AI techniques or architectures. For example, Anderson's *ACT** architecture incorporates the notion of "activation values" into the memory structure and the rule base of a production system architecture [1]. Although such hybrid models do attempt to augment the weaknesses of AI, they do not attempt to address issues regarding multiple neural nets because there are no neural net modules in these connectionist models at all. Finally, some efforts have introduced ideas from AI into neural nets. Network regions, for instance, impose hierarchical structures from frame-based systems onto neural networks[6]. Although concerned with the weakness of neural nets, these efforts have not been able to overcome the two technologies' intrinsic differences in data representation and data manipulation mechanisms.

In neural networks, data are represented in a distributed fashion within dynamic networks and data manipulation involves numeric computations. In artificial intelligence, each conceptual entity is represented as a unit composed of symbols and pointers to other units, and data manipulation involves logical deduction and pattern matching. Our approach to this mismatch of representations is to integrate AI, not with these basic mechanisms of neural networks, but rather with their high-level functions: i.e., *classification* and *data association.* These refer to the capability of a neural net to take an input pattern and either classify it with respect to some set of classes, or generate an

output pattern most closely associated with the input pattern. Viewed at this functional level, these capabilities are closely related to pattern matching and automated reasoning functions in symbolic systems.

Based on these observations, we will describe a novel hybrid architecture that alleviates the difficulties encountered by current hybrid models through the use of fuzzy logic in integrating the two paradigms at their functional levels. The architecture provides an extremely high degree of synergy between the approaches, along precisely the dimensions required to facilitate ease of programming and enable scaling-up to larger problems.

2.3 Fuzzy Logic and Neural Networks

Several techniques for integrating fuzzy logic and neural networks have been suggested. For instance, neural nets have been suggested for learning the membership functions of a fuzzy set [16]. The learning techniques in neural nets have been applied to learning fuzzy control rules [12]. Finally, fuzzy cognitive map suggests an approach for capturing fuzzy knowledge within the framework of associative memories [10]. Our discussion here will be focused on the roles of fuzzy logic in integrating multiple neural networks and knowledge-based systems and in monitoring the performance of neural networks.

3 A Hybrid Architecture

A high-level block diagram of the proposed hybrid architecture is shown in Figure 1. The architecture has four major components: (1) a set of neural net modules, (2) a symbolic system consisting of a global knowledge base, (3) a fuzzy rule system that supports fuzzy inference rules and fuzzy action rules, (4) and an object-oriented interface between the symbolic system and the neural nets. The neural nets process data obtained either from external sensor devices or from the knowledge base of the symbolic system. The global knowledge base consists of a fuzzy database and a *neural-network taxonomy* that describes meta-level knowledge about the neural nets themselves. The fuzzy database stores data and hypotheses that can be uncertain, imprecise, or vague. The neural-net taxonomy consists of *neural-net classes*, (shown as circles in Figure 1) and individual *neural-net objects* that form the leaves of the taxonomy (shown as rectangles). For instance, the neural-net object BAM_1 belongs to the neural net class **BAM** (Bidirectional Associative Memory), and inherits all the general properties (e.g., its training procedure and its activation process) of the **BAM** class. There is one neural-net object for each neural net module. The fuzzy rule base consists of two types of rules: *fuzzy inference rules* and *fuzzy action rules.* Fuzzy inference rules make plausible inferences by combining symbolic

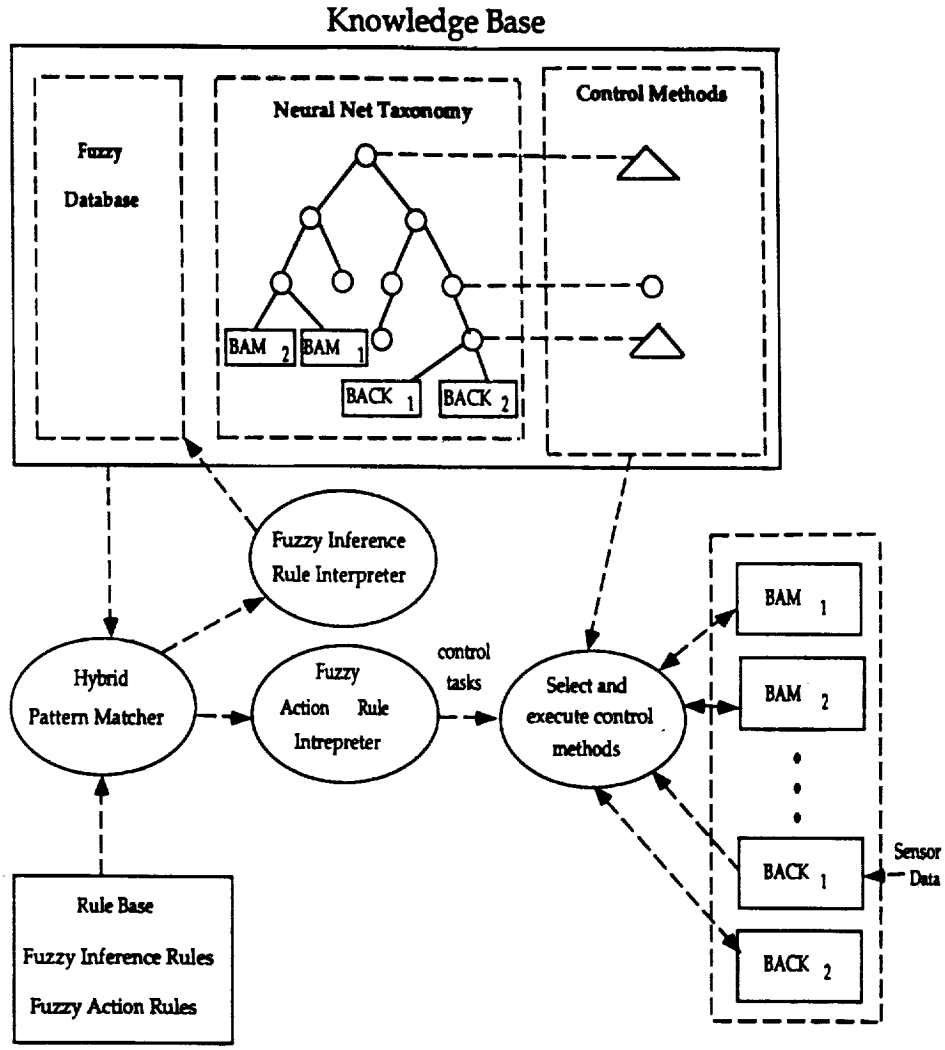


Figure 1: The Hybrid Architecture

information with the outputs of neural networks. *Control tasks* can be invoked either by procedure calls or by fuzzy action rules to effect activation, learning, and modification of neural networks. These control tasks are performed by selecting and executing methods that are inherited through the neural network taxonomy.

The hybrid architecture is an extension of CLASP [23], an advanced AI programming environment that fuses the best aspects of frames, rules, and object-oriented programming. In the following sections, we discuss four major technical issues of the proposed hybrid architecture:

1. Using fuzzy inference rules to combine the output of multiple neural networks with symbolic information;
2. Modeling meta-level knowledge about neural networks in a symbolic knowledge base;
3. Using a set of *control tasks*, which are implemented by methods in object-oriented programming, to define the interface between symbolic systems and neural nets;
4. Using fuzzy action rules to recognize situations necessitating actions upon neural networks.

Throughout the following discussion, we will use a sensor fusion system for anti-submarine warfare as an example to illustrate our approach. This hypothetical system consists of multiple neural nets for classifying various kinds of sensor input and for integrating various information about submarines, along with a symbolic expert system for analyzing the findings and planning anti-submarine strategies.

3.1 Fuzzy Inference Rules

We use fuzzy inference rules to assimilate the outputs of neural networks into the symbolic system, because neural networks often generate classification results that are imprecise in nature. For instance, a neural network that determines the hostility classification of a submarine could generate a qualitative measure of hostility (e.g., hostility degree is 0.7), or a membership values of several fuzzy sets (e.g., membership value of very-hostile is 0.6, membership value of hostile is 0.8, ...).

A fuzzy inference rule checks certain soft conditions, than make a plausible conclusion based on the degree those conditions are satisfied. The condition side of a fuzzy rule consists of fuzzy conditions as well as non-fuzzy condition. A fuzzy condition can be checked by invoking a neural net module in a data-driven fashion (i.e., the neural net

```
If Source was lost due to fade-out in the NEAR-PAST, and
   Similar source started up in an another frequency, and
   Locations of the sources are relatively CLOSE
Then
   The possibility that they are the same Source is MEDIUM.
```

Figure 2: An Example of Fuzzy Inference Rules and Data-driven Neural Nets

```
If Report exists for a vessel class Rose to be in the vicinity, and
   Source likely to be associated with Rose has been detected,
Then Expect to find other Source types associated with Rose class.
```

Figure 3: An Example of Fuzzy Inference Rules

is activated by the arrival of data). From the symbolic system's point of view, neural net modules act as predicates in a fuzzy rule's condition side that check a "soft" (fuzzy) condition and return a number between zero and one indicating the degree of matching (e.g., the membership value of a fuzzy set). Figure 2 shows an example of fuzzy inference rule[1] where source refers to some noise-producing objects, such as propellers and shafts on ships. Fuzzy sets in the rules are expressed in uppercase. Suppose a neural net NN_1 classifies sensor data from hydrophones into possible sources of the noise. The fuzzy inference rule will combine the output of the neural net with other symbolic information (e.g., the reason a source was lost, the location of the sources) to determine the applicability of the rule.

In addition to use the output of a neural net in a data-driven fashion, a fuzzy inference rule can also invoke a neural net in a goal-driven fashion. For instance, the fuzzy inference rule in Figure 3 creates an expectation about the existence of certain source types. This expectation can be verified by several neural net modules that classifies noise sources associated with Rose class vessel.

[1]The examples in Figures 2 and 3 are two rules in HASP, a Blackboard system that analyzes sensor data from hydrophone arrays for ocean surveillance mission [8].

3.2 Modeling Meta-level Knowledge about Neural Networks

For a symbolic system to control neural nets and to use them as modules that extend its reasoning capabilities, it needs some information about the performance and the functional behaviors (e.g., input/output descriptions) of the neural nets. Such information is particularly crucial for integrating neural nets and symbolic systems, as they can not easily communicate with each other otherwise. Our approach is to symbolically represent information about classes of neural networks and individual neural networks, using a principled frame-based knowledge representation mechanism, called *term subsumption languages*[17]. Doing so offers three important advantages.

1. The model describes the functional behavior of neural networks in a way that helps the symbolic system invoke neural nets to extend its capabilities. For instance, an input/output description of a neural net allows the symbolic expert system to tell when a question it is working on can be answered by activating a particular neural net.

2. It provides the basic structure for our method inheritance mechanism (see Section 3.3). This allows general methods and specific methods to be described at their appropriate abstraction level, which facilitates the sharing of common methods and a saving of effort in developing and modifying them.

3. Finally, this approach enables the symbolic system to reason about the behavior of neural networks using automatic classification reasoning capabilities of term subsumption systems[18], which extend the system's knowledge about neural nets beyond what's stated explicitly in the model.

Figure 4 shows an example of meta-level knowledge that might be kept about a neural net for classifying the hostility of a submarine based on its location, speed, direction of movement, and depth. Several attributes need explanation. **Reliability** is the cumulative performance measure of the neural net, while **performance-measure** records the performance of the neural net's last activation. The **reliability-threshold** is the minimum reliability of the neural network that the system can tolerate. A neural net needs to be modified when its reliability is below its threshold value.

CLASP provides a rich term subsumption language, LOOM [13], for modeling meta-level knowledge about neural nets. *Term Subsumption Languages* are knowledge representation formalisms that employ a formal language, with a formal semantics, for the definition of terms (more commonly referred to as concept or classes), and that deduce whether one term subsumes (is more general that) another [17]. These formalisms generally descend from the ideas presented in KL-ONE [5]. Term subsumption languages

```
Name: BACK1
Type: Three-layer-feedforward
Learning: Back-propagation
Input: Location, speed, direction, depth
Output: Hostility
Training-status: Trained
Performance-measure: Satisfactory
Reliability: 0.9
Reliability-threshold: 0.7
```

Figure 4: Meta-level Knowledge about a Neural Net

are a generalization of both semantic networks and frames because the languages have well-defined semantics, which is often missing from frames and semantic networks [20, 4]. The major benefit of using a term subsumption language (e.g., LOOM) to model the neural nets lies in its strong support for developing a consistent and coherent class taxonomy. This can be illustrated by the following example. Suppose the model defines that (1) a **possible-spurious-recognition-net** is any **noise-sensitive-net** which has two examplars that differ in less than two pixels; and, (2) CG_1 is a neural net module of type **Carpenter-Grossberg-net**, which is a kind of **noise-sensitive-net**. If CG_1 has two examplars that differ only in one pixel, LOOM will infer that CG_1 is a **possible-spurious-recognition-net**. Thus, using a term subsumption language to model the neural net taxonomy improves the consistency of the taxonomy, avoids redundancy in the model, and minimizes human errors introduced into the meta-level knowledge base.

3.3 Control Tasks and Methods

To link a symbolic system and neural net modules, a hybrid system needs to define a set of functions that interface between them. These functions facilitate the construction of a layered hybrid system by serving as the intermediate layer between the symbolic system and the neural nets. This layered approach means that hybrid systems will be built in a flexible and extensible way because we can extend the intermediate layer with minimum modification to the symbolic system and the neural nets.

Our approach to building the intermediate level has two major aspects. First, we use a set of generic functions (called *control tasks*) to define the interaction between the symbolic system and the neural networks. Second, we use methods in object-oriented

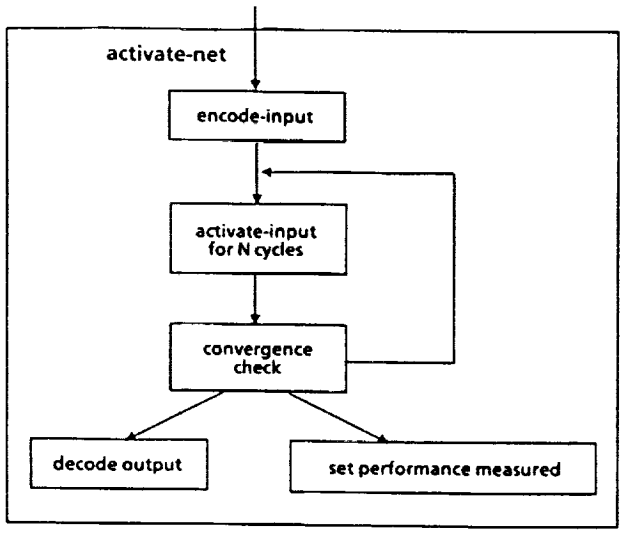


Figure 5: An example of control task decomposition

programming to implement control tasks.

Conceptually, we can view control tasks as messages sent back and forth between symbolic systems and neural networks. Symbolic systems use control tasks to activate and modify neural network modules; these, in turn, use control tasks to inform the symbolic system about their input/output behaviors. For example, the symbolic system would send an **activate-net** message to a neural network object in order to activate its corresponding neural network module[2]. Conversely, the neural network module would send a **set-performance-measure** message to the neural net object in order to update the neural net's **performance-measure** (possibly causing monitoring rules to be triggered). Some of the basic control tasks supported by the architecture may include: **activate-net**, **train-net**, **set-training-status**, **set-performance-measure**, **update-reliability**, and **remove-training-pattern**.

Our approach increases the reusability of modules and reduces the cost of developing and maintaining the system in two ways. First, it separates the purpose of a task from its implementation. Using control tasks to indicate "what needs to be done" allows the symbolic system and the neural nets to interact at an abstraction level that is independent of their detailed implementations. Second, our approach facilitates decomposing tasks into subtasks that can be shared by multiple neural nets. For example, the control task **activate-net** can be further decomposed into five subtasks as shown in Figure 5. By decomposing control tasks into subtasks, which are functional modules, we sep-

[2]A neural network module can also be activated by the arrival of sensor data

arate application-specific modules (such as **encode-input** and **decode-output**[3]) from application-independent modules (such as **activate-input**).

CLASP offers a mechanism for defining generic functions (also called *operators*) that can be invoked by rules or by function calls in any program [22]. CLASP's capability to invoke generic functions by rules and by procedural call is important because it allows the symbolic system to invoke control tasks by rule triggering, and the neural net modules to initiate control tasks through procedural invocations.

Control tasks will be implemented using method inheritance mechanisms in CLASP's object-oriented programming capabilities[4]. The **methods** implementing control tasks are attached to the neural net objects, which are organized into a taxonomy. An individual neural net inherit all its methods from its parents in the taxonomy. To implement a control task for a neural net N, the architecture finds a method for the task that is inherited from the most specific parent of N. This approach increases the reusability of methods, and avoids redundancy in defining similar methods. For example, although different bidirectional associative memories (BAM's) may differ in how they encode and decode symbolic information, they could all share the same **activate-input** method.

3.4 Fuzzy Action Rules

In addition to storing meta-level information about neural nets and specifying possible control actions on a neural net, the symbolic system needs a mechanism for recognizing situations within neural nets that indicate a need for action. Even though production systems in artificial intelligence offers such a capability, they do not address the issue of *partial matching* (accepting an approximate fit between observed data and a rule's condition). A production system that takes into account the degree of partial matching will enable the system to respond in a flexible way even in the face of incomplete or noisy data.

Our approach is to use fuzzy action rules, a generalization of production rules, to issue control task to neural net modules. A fuzzy action rule can use the degree its condition is satisfied to adjust its action[5]. Depending on the partial matching result, a fuzzy action rule may or may not be deemed applicable. For example, a rule may be viewed applicable

[3]In our terminology, *encoding* refers to transforming raw sensor data or symbolic information into neural net representations, and *decoding* refers to transforming neural net representations back into symbolic form.

[4]Actually, the method-dispatching mechanism in CLASP is more general than those in object-oriented programming languages (e.g., SMALLTALK-80) in that it allows programmers to describe more complex situations in which a method applies [21].

[5]The partial matching results of fuzzy productions can also be used for conflict resolution[3].

```
If neural net N is a kind of bidirectional associative memory, and
   its classification results are UNSATISFACTORY,
Then decrease its reliability SLIGHTLY.

If the reliability of a neural net is VERY LOW,
Then set a goal to diagnose and fix the neural net and initialize
     the priority of the goal to be proportional to the degree of matching.
```

Figure 6: Two Rules that Monitor the Performance of a Neural Net

only if the degree of matching is greater than a threshold value.

To illustrate how we use fuzzy action rules to control activation, training, and performance of neural nets, two monitoring rules (paraphrased into English) are shown in Figure 6. They monitor neural net modules by updating and acting on the modules' performance measures. The first rule illustrates how our neural net taxonomy allows rules to apply over whole classes of neural net modules. The second rule demonstrates that actions of rules can be high level tasks which cause the symbolic system to pursue further problem solving and diagnostic reasoning.

4 Summary

We have outlined a novel hybrid architecture that uses fuzzy logic to integrate neural networks and knowledge-based systems. Our approach offers important synergistic benefits to neural nets, approximate reasoning, and symbolic processing. Fuzzy inference rules extend symbolic systems with approximate reasoning capabilities, which are used for integrating and interpreting the outputs of neural networks. The symbolic system captures meta-level information about neural networks and defines its interaction with neural networks through a set of control tasks. Fuzzy action rules provides a robust mechanism for recognizing the situations about neural networks that require certain control actions. The neural nets, on the other hand, offers flexible classification and adaptive learning capabilities, which is crucial for dynamic and noisy environment. By combining neural nets and symbolic systems at their functional level through the use of fuzzy logic, our approach alleviates current difficulties in reconciling differences between the low-level data processing mechanisms of neural nets and AI systems.

Our technical approach to achieving this high-level integration also offers several advantages concerning the development and the maintenance of applications based on

the hybrid architecture:

1. Fuzzy logic serves as a natural bridge that brings together subsymbolic processing of neural networks and symbolic reasoning in knowledge-based systems.
2. The interface between symbolic system and neural nets can be modified easily because it is implemented using a layered and modular approach.
3. Meta-level knowledge about neural nets is stored in a taxonomic structure that facilitates the sharing of information and procedures (e.g., methods).
4. Representing information about neural nets using a principled AI knowledge representation language enables the system to reason about the behavior of neural nets using AI deductive reasoning capabilities.

The hybrid architecture, which supports communication and control across multiple cooperative neural nets through the use of fuzzy rules, enables the construction of modular, flexible, and extensible intelligent systems, reduces the effort for developing and maintaining such systems, and facilitates their application to complex real world problems that need to perform low-level data classification as well as high-level problem solving in the presence of uncertainty and incomplete information.

Acknowledgements

I would like to thank Bart Kosko, Robert Neches, Paul Rosenbloom, John Granacki, and C. C. Lee for helpful discussions and comments about the paper. I am mostly in debt to Professor Lotfi A. Zadeh, from whom I receive constant encouragements and inspirations.

References

[1] John R. Anderson. *The Architecture of Cognition.* Harvard University Press, 1983.

[2] M. Arbib, E. Conklin, and J. Hill. *From Schema Theory to Language.* Oxford University Press, 1987.

[3] James Bowen and Jianchu Kang. Conflict resolution in fuzzy forward chaining production systems. In *Proceedings of AAAI-88*, pages 117–121, 1988.

[4] R. J. Brachman. What is-a is and isn't: An analysis of taxonomic links in semantic networks. *Computer*, 16(10):30–36, October 1983.

[5] R.J. Brachman and J.G. Schmolze. An overview of the kl-one knowledge representation system. *Cognitive Science*, pages 171–216, August 1985.

[6] Hon Wai Chun, Lawrence A. Bookman, , and Niki Afshartous. Network regions: Alternatives to the winner-take-all structure. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 380–387. IJCAI, Morgan Kaufman, 1987.

[7] M. G. Dyer, M. Flowers, and Y. A. Wang. Weight mattrix = pattern of activation: Encoding semantic networks as distributed representations in dual, a pdp architecture. Technical Report UCLA-AI-88-5, Artificial Intelligence Lab., University of California at Los Angeles, 1988.

[8] Robert Engelmore and Tony Morgan. *Blackboard Systems.* Addison-Wesley Publishing, 1988.

[9] Stephen I. Gallant. Connectionist expert systems. *Communications of the ACM*, 31(2), February 1988.

[10] Bart Kosko. Fuzzy associative memories. In A. Kandel, editor, *Fuzzy Expert Systems.* Addison-Wesley, MA, 1987.

[11] Bart Kosko. Bidirectional associative memories. *IEEE Transactions on Systems, Mand, and Cybernetics*, 18(1), 1988.

[12] C. C. Lee. A self-learning rule-based controller employing fuzzy logic and neural net concepts. *International Journal of Intelligent Systems (to appear)*, 1990.

[13] R. M. MacGregor. A deductive pattern matcher. In *Proceedings of AAAI-88*, 1988.

[14] James L. McClelland and David E. Rumelhart. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 2. Bradford Books, Cambridge, MA, 1986.

[15] R. Neches, W. Swartout, , and J. Moore. Enhanced maintenance and explanation of expert systems through explicit models of their development. *Transactions On Software Engineering*, SE-11(11):1337–1351, November 1985.

[16] Yoh-Han Pao. *Adaptive Pattern Recognition and Neural Networks.* Addison-Wesley Publishing, 1989.

[17] Peter F. Patel-Schneider, Bernd Owsnicki-Klewe, Alfred Kobsa, Nicola Guarino, Robert MacGregor, William S. Mark, Deborah McGuinness, Bernhard Nebel, Albrecht Schmiedel, and John Yen. Report on the workshop on term subsumption languages in knowledge representation. to appear in AI Magazine, 1990.

[18] James Schmolze and Thomas Lipkis. Classification in the kl-one knowledge representation system. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*. IJCAI, 1983.

[19] David S. Touretzky and Geoffrey E. Hinton. Symbols among the neurons: Details of a connectionist inference architecture. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 238–243. IJCAI, Morgan Kaufman, 1985.

[20] William A. Woods. Whats's in a link: Foundations for semantic networks. In Daniel Bobrow and Allan Collins, editors, *Representation and Understanding: Studies in Cognitive Science*. Academic Press, 1975.

[21] John Yen. A principled approach to reasoning about specificity of rules. In *Proceedings of AAAI-90*, August 1990.

[22] John Yen, Hsiao-Lei Juang, and Robert MacGregor. Using object-oriented programming to enhance the maintainability of expert systems. in review, 1990.

[23] John Yen, Robert Neches, and Robert MacGregor. Using terminological models to enhance the rule-based paradigm. In *Proceedings of the Second International Symposium on Artificial Intelligence*, Monterrey, Mexico, October 25-27 1989.

[24] L. A. Zadeh. Possibility theory and soft data analysis. In L. Cobb and R. M. Thrall, editors, *Mathematical Frontiers of the Social and Policy Sciences*, pages 69–129. Westview Press, Boulder, Colorado, 1981.

AUTHOR INDEX

Anderson, James A. 109

Berenji, Hamid R. 1
Bezdek, James C. 145
Bishop, Thomas 115

Collins, Dean R. 109

Gallant, A. R. 61
Gately, Michael T. 109
Greenwood, Dan 115

Hayashi, Isao 171
Hirota, Kaoru 185
Hornik, K 61

Ikoma, Norikazu 185

Jani, Yashvant 81

Keller, James M. 205
Kosko, Bert 3

Lea, Robert N. 81
Lee, Chuen-Chien 197

Nomura, Hiroyoshi 171

Pal, Sankar K. 213
Penz, P. Andrew 109

Ruspini, Enrique H. 235

Shelton, Robert O. 63
Shew, Kenneth 115
Stevenson, Fareed 115
Stinchcombe, M. 61
Symon, James R. 161

Taber, Rod 31
Teh, H. H. 97

Villarreal, James A. 63

Wang, P. Z. 97
Wakami, Noboru 171
Watanabe, Hiroyuki 161
Werbos, Paul J. 153
White, Halbert 61
Wu, Z. Q. 97

Yen, John 221

NASA
National Aeronautics and Space Administration

REPORT DOCUMENTATION PAGE

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
CP 10061		

4. Title and Subtitle	5. Report Date
Proceedings of the Second Joint Technology Workshop on Neural Networks and Fuzzy Logic	April 1990
	6. Performing Organization Code PT4
7. Author(s) Robert N. Lea, Editor James Villarreal, Editor	8. Performing Organization Report No. S-624
9. Performing Organization Name and Address Lyndon B. Johnson Space Center Information Technology Division Houston, TX 77058	10. Work Unit No.
	11. Contract or Grant No.
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546	13. Type of Report and Period Covered Conference Publication
	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

Documented here are papers presented at the Neural Networks and Fuzzy Logic Workshop sponsored by the National Aeronautics and Space Administration and cosponsored by the University of Houston, Clear Lake. The workshop was held April 11 - 13 at the Lyndon B. Johnson Space Center in Houston, Texas. During the three days, approximately 30 papers were presented. Technical topics addressed included adaptive system; learning algorithms; network architectures; vision; robotics; neurobiological connections; speech recognition and synthesis; fuzzy set theory and application, control and dynamics processing; space applications; fuzzy logic and neural network computers; approximate reasoning; and multiobject decision making.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
fuzzy logic, non-Lipschitzian dynamics, parallel distributed models, algorithms, neural network, spatiotemporal patterns, neuron ring, fuzzy controllers, signal processing, pattern recognition	Unclassified Volume I - unlimited Volume II - unlimited Subject Category - 63

19 Security Classification (of this report)	20. Security Classification (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified		

For sale by the National Technical Information Service, Springfield, VA 22161-2171