INCORPORATING FUZZY LOGIC IN AN ARTIFICIAL NEURAL NETWORK

A Thesis

by

ELIZABETH ANN RAMIREZ

April 1996

Major Subject: Electrical Engineering

INCORPORATING FUZZY LOGIC IN AN ARTIFICIAL NEURAL NETWORK

A Thesis

by

ELIZABETH ANN RAMIREZ

April 1996

Major Subject: Electrical Engineering

[illegible]

Karen L. Butler
(Advisor)

[illegible]

Undergraduate Fellows Director

ABSTRACT

Incorporating Fuzzy Logic in an Artificial

Neural Network. (April 1996)

Elizabeth Ann Ramirez, Texas A&M University

Advisor: Dr. Karen L. Butler, P.E.

A clustering neural network that uses unsupervised learning to generate clusters is described. A fuzzy logic classifier is used in conjunction with the network to assign membership functions to patterns that describes the degree of membership that the pattern has for a cluster. The input that is fed into the neural network are nonfuzzy (crisp) values. A two-stage clustering neural network that uses unsupervised learning in the first stage and supervised learning in the second stage is to be revised so that the supervised function is disabled. Therefore, the resulting clusters will be generated from the unsupervised stage of the original neural network. The fuzzy logic classifier is developed by implementing a similar algorithm used by fuzzy c-means for generating fuzzy membership functions. By assigning a membership function to the data points, the patterns are allowed to hold membership to more than one cluster. This is in contrast to the crisp membership that is held by the patterns in the original neural network. Data that is ambiguous can be represented and processed using an algorithm such as this. The neural network is written in FORTRAN and the fuzzy logic classifier is written in MATLAB.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

The topic to be investigated is incorporating a fuzzy logic classifier in an artificial neural network. The neural network is a two-stage model that was developed based on the ISODATA algorithm for clustering. Unsupervised learning is used in the first stage and supervised learning is used in the second stage. The classifier uses fuzzy classification in it's reasoning system to define clusters rather than using conventional crisp classification. The fuzzy membership function used to define the fuzzy clustering follows the fuzzy c-means algorithm.

Section I gives a brief introduction of the research project that is to be covered in this thesis. Section II covers the topics of neural networks, fuzzy logic, and clustering algorithms which are relevant subjects to this project. Section III states the problem that the research is to investigate and the tasks that need to be accomplished. Section IV gives a description of the software that will be manipulated in this research. Section V justifies why this thesis topic is relevant. Section VI summarizes the changes that are to be made to the original software. Section VII reports the studies performed and the results obtained for this research project. Finally, Section VIII summarizes and concludes the findings found during this research project.

II. RELEVANT SUBJECT MATTER

A. Neural Networks

Artificial neural networks consist of several simple processing elements called neurons arranged in a massively paralleled architecture that collectively function in order to simulate the processes of the human brain [5]. They can be used for pattern matching, classification, or other nonnumeric problems by employing many parallel computations and approximations without logical rules or mathematical equations.

The neurons are related to each other through weighted arc links. A network can be trained to recognize a pattern by strengthening the relevant arc weights associated with a pattern and weakening the incorrect weights. It can learn a pattern and remember it when processing new data. Learning is usually accomplished using supervised or unsupervised learning procedures.

In supervised learning, the pattern recognition process uses data that are assigned class labels [8]. The network uses these labels to create boundaries between classes that minimize misclassification of the labeled data. These class labels serve as the supervision and establishes validity which the neural network uses to find clusters and learn patterns.

Unsupervised learning is a learning mode utilized by a neural network. using unlabeled data in order to find natural groupings that are independent of class labels [9]. Therefore, the neural network is allowed to find clusters without supervision provided from class labels.

B. Fuzzy Logic

Fuzzy logic is an extension of conventional Boolean logic that can handle the concept of partial classification [10]. Traditional logic systems assign precise class values to an element. Fuzzy logic has unclear values that can represent a range of numbers.

A classical, nonfuzzy (crisp) function describes an element as either 1, meaning it is 100% classified to a set, or 0, meaning it is 100% not classified to a set. Alternatively, the fuzzy function allows an element to be classified to a set with a degree of membership that is anywhere in the interval [0,1]. This means it can hold any value between 0% and 100% to describe if it is classified to a set or not classified to a set. The membership function describes the degree of membership that an element has for a set.

Fuzzy logic does not require an element to be true OR false, but true AND false to varying degrees as described by the membership function. By using fuzzy sets as pattern classes, data with built-in ambiguities can be described allowing artificial logic to deal with vague information. Dynamic real systems need fuzzy sets and fuzzy logic in order to represent the uncertainty that is inherent in their complexity.

C. Clustering Algorithms

Clustering is a method that tries to define the relationship between patterns of a data set by organizing them clusters so that patterns within a cluster are more similar to each other than to patterns that belong in different clusters [4]. These clusters contain data in the same group that are close to each other and not close to points in other groups.

A classical clustering algorithm divides and assigns data to only one cluster. Alternatively, a fuzzy clustering algorithm assigns data to more than one cluster and gives a degree of membership that relates how strongly an element belongs to that cluster. The advantage of fuzzy representations in cluster analysis is that patterns which represent uncertainties, such as stray or isolated points, can be classified as such.

1. ISODATA Clustering Algorithm

The ISODATA algorithm used in unsupervised cluster analysis partitions a data set into clusters using an iterative process and also finds the cluster centers which are major representative elements of a data set [2]. The ISODATA algorithm first partitions up the data set into initial clusters and finds the centers of these clusters. A new set of clusters is then constructed where the distance between the data points and the cluster center is minimized. If the new clusters are the same as the old clusters, the process stops. Iteration continues until all of the clusters have a minimum distance from their respective data points to their centers. The process is done using the following method [7]:

(1) Initial partition of the data set into m clusters, $F=F_1,..., F_m$

(2) Compute the centers v_i of the cluster F_i

(3) Create a new partition of clusters, F', if $d(x,v_i) = \min d(x,v_j)$

(4) If F'=F stop iteration, otherwise set F'=F and go to step (2)

2. Fuzzy C-Means Clustering Algorithm

A procedure similar to that used by the fuzzy c-means algorithm for obtaining a fuzzy membership function was used in designing the fuzzy logic classifier [7]. Therefore, only the membership function process used in fuzzy c-means will be described, as opposed to describing the complete clustering algorithm.

The membership function, μ_{ij}, is obtained by first being initialized to an arbitrary value. The distance, *d,* between the cluster centers, *V*, and the data points, *X*, are found. The process is done using the following method [1]:

(1) Initialize the membership function, μ_{ij}, of point *X* belonging to cluster *i* such that

$$\sum_{i=1}^{c} \mu_{ij} = 1 \qquad (1)$$

(2) Compute the cluster centers, V_i, for i=1,...,c using the formula

$$V_i = \frac{\sum_{j=1}^{n} (\mu_{ij})^m X_j}{\sum_{j=1}^{n} (\mu_{ij})^m} \tag{2}$$

(3) Update the fuzzy membership function, μ_{ij}, using the formula

$$\mu_{ij} = \frac{\left(\frac{1}{d^2(X_j, V_i)}\right)^{\frac{1}{(\exp-1)}}}{\sum_{i=1}^{c}\left(\frac{1}{d^2(X_j, V_i)}\right)^{\frac{1}{(\exp-1)}}} \tag{3}$$

Where *c* is the number of clusters, *j* is the index of the data points, *i* is the index of the cluster centers, and *exp* is the defined fuzziness index.

Incorporating fuzzy logic with neural networks provides complementing methods of reasoning and computing. Neural networks learn rules for fuzzy logic and fuzzy logic infers from unclear neural network parameters [6]. The network has learning capabilities that produce output from fuzzy input while avoiding time-consuming arithmetic operations.

III. PROBLEM STATEMENT

There are several tasks which must be accomplished in order for the fuzzy classifier to be implemented in the neural network. The first task is to revise the original neural network so that the supervised stage is disengaged and only the unsupervised stage can generate clusters based on the input patterns spatial proximity. The neural network FORTRAN programs must be manipulated in order for this to occur.

Another task to be accomplished is developing the fuzzy logic classifier that will assign a degree of membership to an input pattern. This task involves developing the membership function algorithm which will be used in conjunction with the new neural network clusters. The fuzzy logic classifier must be created so that it can be implemented in software that is compatible to the neural network's output format.

IV. DESCRIPTION OF COMPUTER PROGRAMS

A. Original Neural Network

The neural network utilized in this research consists of a two-stage clustering model that uses distinct, crisp classes to define the membership of input data to the clusters [3]. The input data set used in the neural network represent patterns that have a class and a position in space associated with them. The first stage is the unsupervised learning mode that "blindly" processes the input set without knowing the class membership labels. A natural set of clusters representing decisive regions in the pattern space is formed which are defined by a spherical radius [9].

The second stage of the neural network is the supervised learning mode that uses the class membership labels of the input data set to evaluate the resulting clusters generated from the first stage of the algorithm [8]. Only the clusters which contain patterns of the same class are placed into the final output subset. The remaining heterogeneous clusters are repeatedly executed through the two-stage process, which reduces the spherical radius in the unsupervised learning mode after each subsequent execution, until a set of homogeneous clusters is formed.

The final output set of clusters contain patterns that belong to the same class [3]. The spherical radius for each cluster varies according to the number of times the patterns were executed through the network. Each cluster is assigned a distinct, crisp class according to the class of the patterns that it contains. The following sections describe the steps used by the network in further detail.

1. Neural Network Input

The input information used by the original neural network to create clusters for two dimensional patterns is the dimension, the total number of patterns, the pattern class, and the pattern position in space [3]. These values are read from the input file ANNIN.DAT which is shown in Section VII.

2. Cluster Generation

The information from ANNIN.DAT is fed into the two-stage neural network algorithm until homogeneous clusters are formed with crisp classes. FORTRAN programs implemented by the algorithm are TRAIN.FOR, UNSUPER.FOR, and SUPER.FOR [3]. The code and a brief description of the FORTRAN programs used by the original neural network is given in Appendix A.

The first step that the neural network does is process the data from the input file ANNIN.DAT through the program UNSUPER.FOR. Clusters are generated that are dependent on the patterns position in space where all patterns are contained within a radius RHO. All class membership information is disregarded in this step [3].

The second step that the neural network does is process the data from the program UNSUPER.FOR through the program SUPER.FOR. This information includes the dimension, the number of patterns, the normalized pattern position, the class membership of the pattern, and the cluster number that the pattern belongs to.

The supervised stage merges clusters of the same class that are close together and tightens the radius of the clusters so that RHO is the minimum possible distance. Clusters that are determined to be homogeneous are placed into the output subset and the remaining clusters are fed back to UNSUPER.FOR. The clusters that are fed back have their value of RHO decremented before they are reprocessed [3]. The steps performed along with their corresponding FORTRAN programs are shown in Figure 1.

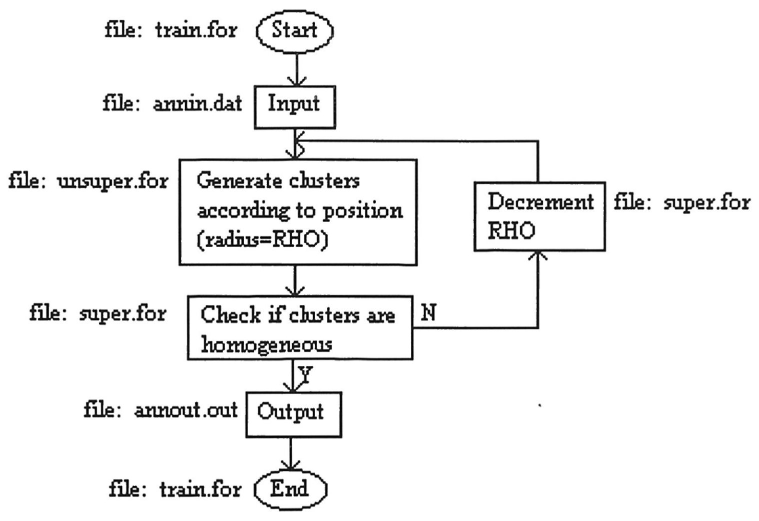

Figure 1: Original Neural Network Algorithm

3. Neural Network Output

The clusters generated through the training procedure are output to a file that gives the number of clusters, the dimension, the cluster number, the radius, the number of patterns in the cluster, the cluster class, and the cluster center [3]. These values are given in the output file ANNOUT.OUT which is shown in Section VII.

B. REVISED NEURAL NETWORK

The research revised the clustering neural network algorithm by affecting how it partitions up the pattern space into clusters. Originally, the generation of the final set of clusters was based on class membership of the patterns and whether the clusters were homogeneous. The altered neural network partitions up the pattern space into equal clusters with a radius of RHO and assigns a membership function to each pattern that describes it's cluster membership.

The final set of clusters uses an unsupervised mode because they are independent of the class labels of the contained patterns. The clusters have the same radius, but can contain different classes of patterns.

1. Neural Network Input

The information used by the revised neural network is identical to that used by the original neural network. The information includes the dimension of the pattern, the total number of patterns, the pattern class, and the pattern position in space [3]. These values can be seen by referring to the input file ANNIN.DAT in Section VII.

2. Cluster Generation

The information from ANNIN.DAT is fed into the revised neural network algorithm using the FORTRAN programs TRAIN.FOR, UNSUPER.FOR, and SUPER.FOR. The code and a brief description of the FORTRAN programs used by the revised neural network is given in Appendix B.

The first step that the neural network does is process the data from the input file ANNIN.DAT through the program UNSUPER.FOR [3]. This step is identical to the first step performed by the original neural network. Clusters are generated that are dependent on the patterns position in space where all patterns are contained within a radius RHO. All class membership information is disregarded in this step.

The second step that the neural network does is process the data from UNSUPER.FOR through the program SUPER.FOR. This information includes the dimension, the number of patterns, the normalized pattern, the class membership of the pattern, and the cluster number that the pattern belongs to. SUPER.FOR places the data from UNSUPER.FOR into the output file CLUSTER.DAT. The steps performed along with their corresponding FORTRAN programs are shown in Figure 2.

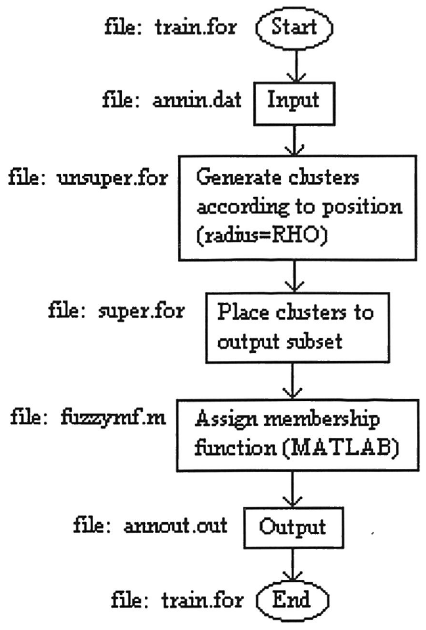

Figure 2: Revised Neural Network Algorithm

3. Neural Network Output

The clusters generated through the training procedure are outputted to a file that gives the number of clusters, the dimension, the cluster radius, and the cluster center. These values are given in the output file CLUSTER.DAT which is shown in Section VII.

4. Fuzzy Membership Function

The information from CLUSTER.DAT is used by the MATLAB file FUZZYMF.M to create the fuzzy membership function matrix, U [7]. The process used to create the matrix follows the fuzzy c-means algorithm for membership function generation.

IV. JUSTIFICATION

The reason for incorporating fuzzy logic in the neural network's classification system was to model events that do not have an absolute value. By using fuzzy theory, the degree to which an event occurs can be measured. Clustering uses information to organize data into categories so that patterns within a cluster are more similar to each other than to patterns of another cluster [9]. Fuzzy clustering allows a pattern to belong to several clusters with varying degrees depending on the membership function [2].

By using the fuzzy logic classifier in a mode that uses unsupervised learning, the class membership of the patterns can be disregarded and natural clusters can be generated. The fuzziness that is used in the clustering classification is important when modeling real events that can have many dynamic parameters involved with them. Data that holds traits to more than one class can be effectively represented.

VI. SUMMARY OF REVISIONS

The most drastic change made to the original neural network was removing the supervised mode used for cluster generation. Originally, the supervised stage was used for cluster verification and merging and tightening of the radius that describes the clusters [3]. Alternatively, there is no supervised stage, SUPER.FOR simply places the data from the unsupervised stage into the output subset without checking for overlapping or homogeneous clusters.

The lack of supervision affects the output file that describes the cluster's properties. The number and center of clusters will not remain the same because of the different processes used to generate them and the characteristics of its cluster radius are also affected. Originally, the radius varied depending on how many times the data was looped through the two stages. The revised process assigns each cluster the same radius value and executes the process once.

The original two-stage unsupervised and supervised approach assigns each data point an absolute class and uses these classes to group the data points together. However, the revised unsupervised approach creates a membership function matrix that describes the degree of membership that a data point has to a cluster. Therefore, a data point can belong to more than one cluster.

VII. STUDIES AND RESULTS

A. Input Patterns

The information given in the input file ANNIN.DAT includes the dimension of the pattern, the total number of patterns, the pattern class, and the pattern position in space [3]. The input file and a plot of the normalized patterns is shown below.

Input file: ANNIN.DAT

Line	Variable	Record	Format	File
1	N	No. of dimensions	Integer*4	2
2	P	No. of patterns	Integer*4	11
3	X1	Data point 1	Real*8	1.0
4	Y1	Data point 1	Real*8	1.0
5	C1	Class membership	Character*25	a
6	X2	Data point 2	Real*8	2.0
7	Y2	Data point 2	Real*8	3.0
8	C2	Class membership	Character*25	b
9	X3	Data point 3	Real*8	2.0
10	Y3	Data point 3	Real*8	7.0
11	C3	Class membership	Character*25	a
12	X4	Data point 4	Real*8	1.0
13	Y4	Data point 4	Real*8	7.0
14	C4	Class membership	Character*25	b
15	X5	Data point 5	Real*8	1.5
16	Y5	Data point 5	Real*8	9.0
17	C5	Class membership	Character*25	b
18	X6	Data point 6	Real*8	2.0
19	Y6	Data point 6	Real*8	10.0
20	C6	Class membership	Character*25	c
21	X7	Data point 7	Real*8	1.0
22	Y7	Data point 7	Real*8	10.0
23	C7	Class membership	Character*25	a
24	X8	Data point 8	Real*8	7.0
25	Y8	Data point 8	Real*8	2.0
26	C8	Class membership	Character*25	b
27	X9	Data point 9	Real*8	6.5
28	Y9	Data point 9	Real*8	4.0
29	C9	Class membership	Character*25	c
30	X10	Data point 10	Real*8	7.5
31	Y10	Data point 10	Real*8	4.0
32	C10	Class membership	Character*25	c
33	X11	Data point 11	Real*8	7.0
34	Y11	Data point 11	Real*8	8.0
35	C11	Class membership	Character*25	a

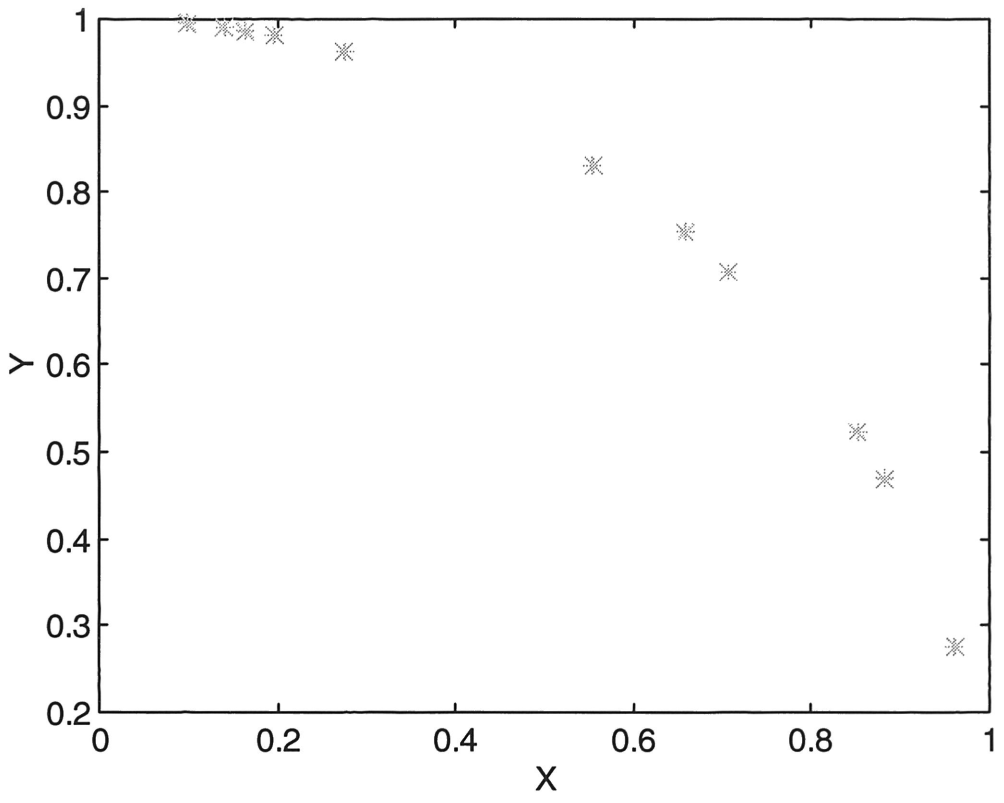

Figure 3: Normalized Input Patterns

B. Original Neural Network Clusters

The clusters generated through the training procedure for the original neural network are output to a file that gives the number of clusters, the dimension, the cluster number, the radius, the number of patterns in the cluster, the cluster class, and the cluster center to the output file ANNOUT.OUT which is shown below [3].

Output file: ANNOUT.OUT

```
Variable                                File
                                        number of clusters
CNUM                                            8
                                         size of centroid vector
N                                               2
                                        CLUS#   RHO          NUM   MEMBER CLASS
CLUS1, RHO1, NUM1, CLASS1                  1   0.20095157473000    2   c
X1                                         0.867418335624768
Y1                                         0.497579572552930
```

CLUS2, RHO2, NUM2, CLASS2	2	0.20000000000000	1	b
X2	0.961523947640823			
Y2	0.274721127897378			
CLUS3, RHO3, NUM3, CLASS3	3	0.20000000000000	1	b
X3	0.554700196225229			
Y3	0.832050294337844			
CLUS4, RHO4, NUM4, CLASS4	4	0.12642232844643	2	a
X4	0.683184085542186			
Y4	0.730246194965703			
CLUS5, RHO5, NUM5, CLASS5	5	0.20000000000000	1	a
X5	0.274721127897378			
Y5	0.961523947640823			
CLUS6, RHO6, NUM6, CLASS6	6	0.20013515340158	2	b
X6	0.152910171771333			
Y6	0.988171708746655			
CLUS7, RHO7, NUM7, CLASS7	7	0.20000000000000	1	c
X7	0.196116135138184			
Y7	0.980580675690920			
CLUS8, RHO8, NUM8, CLASS8	8	0.20000000000000	1	a
X8	9.950371902099892E-002			
Y8	0.995037190209989			

C. Revised Neural Network Clusters

The clusters generated through the training procedure of the revised neural network are output to a file CLUSTER.DAT that gives the number of clusters, the dimension, the cluster radius, and the cluster center. The output file and the cluster centers plotted against the normalized patterns are shown below.

Output file: CLUSTER.DAT

Variable	Record	File
CNUM	No. of clusters	2
RHO	Cluster radius	0.719540875323049
X1	Cluster 1 center	0.256402173381587
Y1	Cluster 1 center	0.966570186528220
X2	Cluster 2 center	0.830732249398834
Y2	Cluster 2 center	0.556672192415566

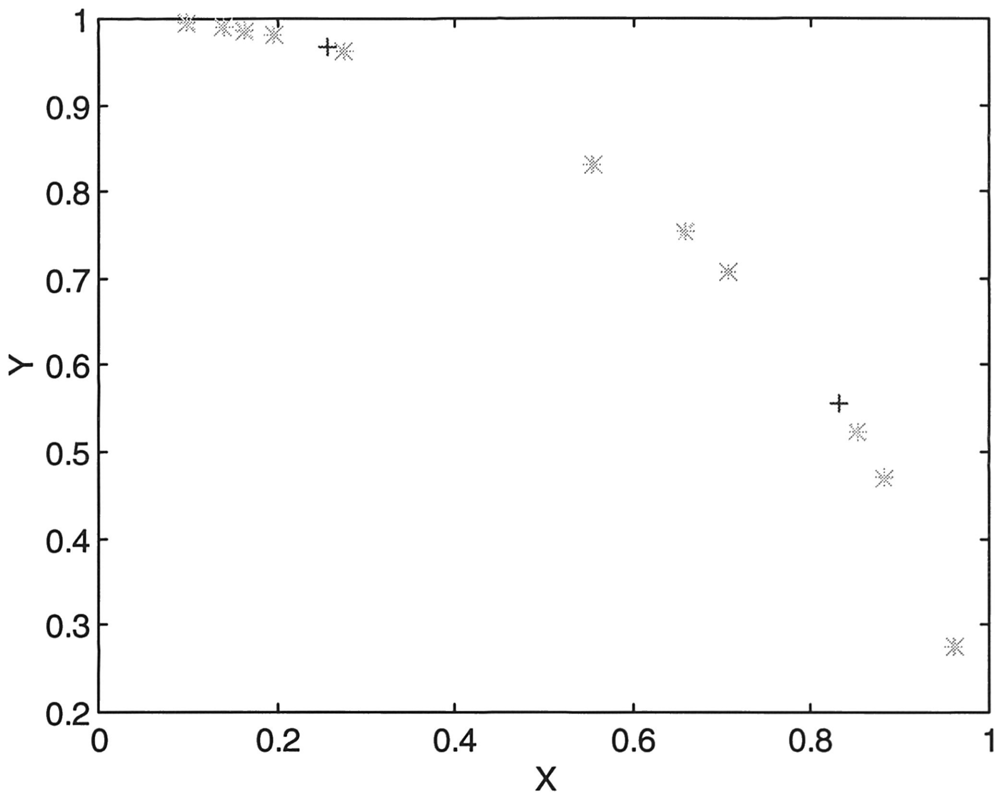

Figure 4: Cluster Centers in Normalized Pattern Space

D. Revised Neural Network Membership Function Matrix

The membership matrix formed using fuzzy logic describes how much a point belongs to a cluster according to its degree of membership. These values can be seen by referring to the membership function matrix below. The membership function plotted in the normalized pattern space is also shown for both clusters in Figure 5 and Figure 6.

The two rows of the matrix represent the two clusters that were generated from the revised neural network. The eleven columns of the matrix represent the eleven input patterns that were used to create the clusters. The element that is in the ith column of the jth row is the degree of membership that the ith pattern has for the jth cluster. For example, μ_{11}=0.1229 and μ_{12}=0.8771 means that pattern 1 holds 12.29% membership for cluster 1 and 87.71% membership for cluster 2.

Membership Function Matrix

U=

0.1229	0.5867	0.9992	0.9797	0.9861	0.9935	0.9662	0.0901	0.0027	0.0156	0.2470
0.8771	0.4133	0.0008	0.0203	0.0139	0.0065	0.0338	0.9099	0.9973	0.9844	0.7530

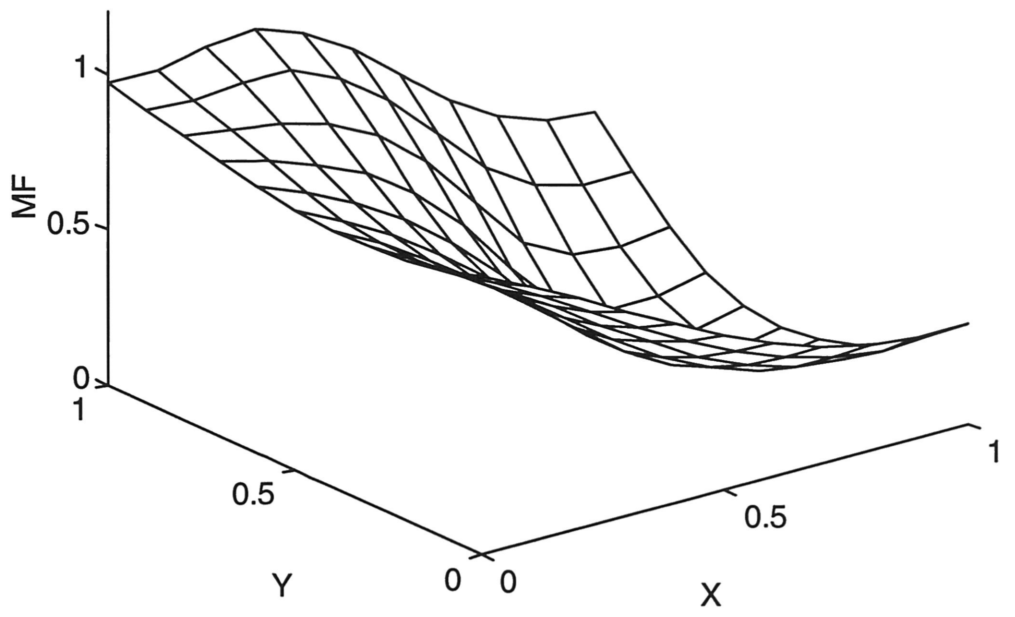

Figure 5: Membership Function Plot for Cluster 1

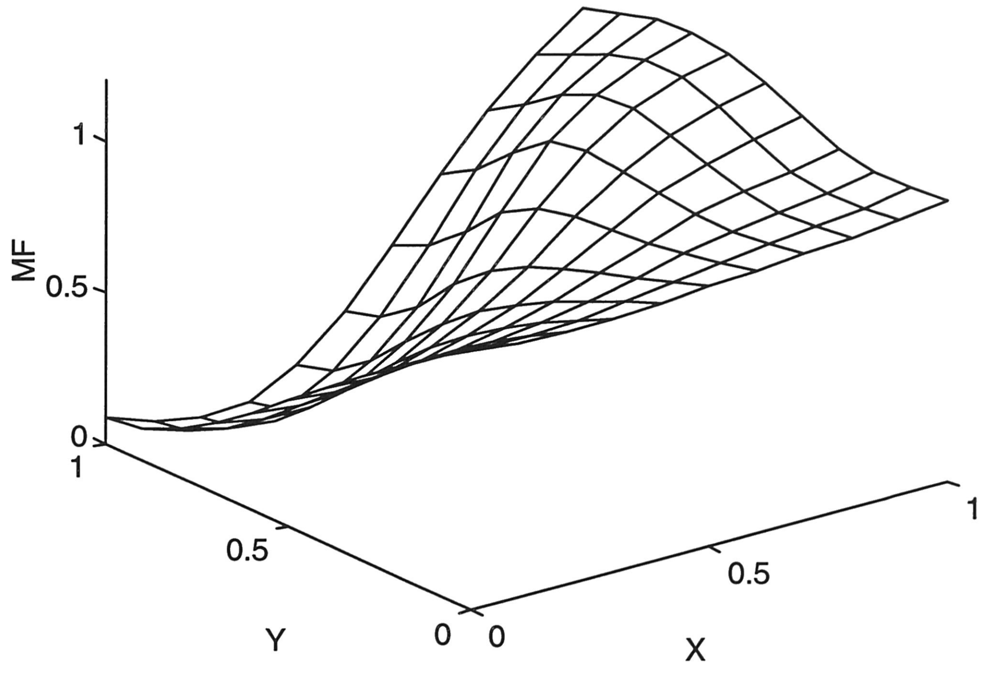

Figure 6: Membership Function Plot for Cluster 2

VIII. SUMMARY AND CONCLUSION

Fuzzy classification is an important tool that describes events which lie between absolute values. The characteristics of real life data sets can be represented with fuzzy logic and processed accordingly using neural networks in order to model actual events. A system that incorporates both fuzzy logic and neural networks is capable of processing commonsense knowledge with the capacity to expand its base of information.

The goal of this research was to implement a fuzzy logic classifier within a clustering neural network. This was accomplished by using the classifier in the unsupervised stage of the network. The major steps involved in accomplishing this project included designing the fuzzy logic classifier, creating the membership function algorithm, and altering the neural network so that only the unsupervised stage generates cluster. This area shows great promise in the development of predictive and estimative systems that model real systems dealing with inexact parameters in inexact environments. Further research will investigate ways to implement fuzzy logic classification in the supervised stage of the neural network.

REFERENCES

1. James C. Bezdek, "A Physical Interpretation of Fuzzy ISODATA", *IEEE Transactions on Systems, Man, and Cybernetics*, May 1976, pp. 387-389.
2. James C. Bezdek, "A Convergence Theorem for the Fuzzy ISODATA Clustering Algorithms", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-2, No. 1, January 1988, pp. 1-8.
3. Karen L. Butler, *Arcing Distribution Fault Diagnosis System Users Manual (Version 1.0)*, Center for Energy Systems and Controls, Howard University, May 1994.
4. Didier Dubois and Henri Prade, *Fuzzy Sets and Systems: Theory and Application*, Academic Press, Inc., 1975.
5. Richard P. Lippmann, "An Introduction to Computing with Neural Nets", *IEEE Acoustics, Speech, and Signal Processing Magazine*, April 1987, pp. 4-22.
6. Sushmita Mitra and Sankar K. Pal, "Logical Operation Based Fuzzy MLP for Classification and Rule Generation", *Neural Networks*, Vol. 7, No. 2, pp. 353-373.
7. F. Martin McNeill and Ellen Thro, *Fuzzy Logic A Practical Approach*, Academic Press, Inc., 1994.
8. Patrick K. Simpson, "Fuzzy Min-Max Neural Networks-Part 1: Classification", *IEEE Transactions on Neural Networks*, Vol. 3, No. 5, September 1992, pp. 776-786.9.
9. Patrick K. Simpson, "Fuzzy Min-Max Neural Networks-Part 2: Clustering", *IEEE Transactions on Fuzzy Systems*, Vol. 1, No. 1, February 1993, pp. 32-45.
10. Lofti A. Zadeh, "Fuzzy Logic", *IEEE Computer Magazine*, April 1988, pp. 83-93.

Appendix A: Original Neural Network FORTRAN Programs

A.1 TRAIN.COM

TRAIN.COM is a command file that compiles and links the FORTRAN programs TRAIN.FOR, UNSUPER.FOR, and SUPER.FOR for training the neural network.

```
$!This command routine calls the files to train the fault diagnosis sytem
$fortran unsuper.for
$fortran super.for
$fortran train.for
$link train,super,unsuper
$del *.obj;*
$!run train
```

A.2 TRAIN.FOR

TRAIN.FOR is a FORTRAN program that trains the neural network using the input file ANNIN.DAT and the FORTRAN programs UNSUPER.FOR and SUPER.FOR.

```
      PROGRAM TRAIN
*********************************************************************
**  THIS PROGRAM TRAINS THE CLUSTERING ANN
**  (1) EACH TRAINING PATTERN AND ITS RESPECTIVE  MEMBERSHIP CLASS
**  IS READ
**  (2) THE ANN IS TRAINED WITH THE DATA
**  (3) THE GENERATED CLUSTERS ARE WRITTEN TO AN OUTPUT FILE
**
*********************************************************************
      IMPLICIT NONE
      INCLUDE 'PARAMS.INC'

      INTEGER*4   I,J,K,M       !Indexes for DO loops
      INTEGER*4   L,R           !Indexes
      INTEGER*4   FIN_CNUM      !Number of Final Clusters
      INTEGER*4   P             !Number of Patterns
      INTEGER*4   N             !Size of Pattern Vector

      INTEGER*4  ICNTR,CNTR,PTRAIN

         INTEGER*4    NUM_PATTS        !Number of training patterns

      STRUCTURE  /XPATTERN/
      REAL*8     XP(NMAX)  !Normalized Input Pattern:
      INTEGER*4   XELEM    !# of Cluster pattern belongs to
      CHARACTER*25  CLASS !The class membership of the pattern
      END STRUCTURE

      STRUCTURE /FIN_CLUSTER/
      REAL*8     B(NMAX)    !Centroid Vector
      REAL*8     CRHO      !Square of Sphere Radius
      INTEGER*4  NUM        !Number of Patterns in the Cluster
      CHARACTER*25  CLASS  !The class membership of the pattern
      REAL*8    PROB       !Probability of cluster
      END STRUCTURE

      RECORD /XPATTERN/ X(PP),XI(PP)
      RECORD /XPATTERN/ FINAL_X
      RECORD /FIN_CLUSTER/ FINAL_C

      COMMON /BLK5/FINAL_X(PP)
      COMMON /BLK6/FINAL_C(PP)
      COMMON /BLK7/FIN_CNUM


      OPEN(UNIT=5,STATUS='OLD',ERR=900,FILE='ANNIN.DAT')
      OPEN(UNIT=6,STATUS='NEW',ERR=900,FILE='ANNOUT.OUT')

300     FORMAT(I4,2X,F18.14,3X,I4,3X,A25)
360     FORMAT('CLUS#',4X,'RHO',13X,'NUM',4X,'MEMBER CLASS')

100     FORMAT(A5)
150     FORMAT(1X,F17.14,1X,F17.14,1X,F17.14,1X,F17.14)
160     FORMAT(1X,F17.14,2X,F17.14)
200     FORMAT(A5)
250     FORMAT(2X,F13.9,5X,F13.9)
350     FORMAT(2X,F15.9,2X,F15.9,2X,F15.9,2X,F15.9)
355       FORMAT(T1,A25)

          rEAD (5,*) N
          rEAD (5,*) NUM_PATTS

!       READ(5,*)  !Skip blank line

      DO J = 1,NUM_PATTS

! Read pattern
        DO M = 1,N
          READ(5,*) XI(J).XP(M)
        END DO !(K=1,N)
! Read Membership class
            READ(5,355) XI(J).CLASS

      END DO !(J=1,num_patts)


          PTRAIN = NUM_PATTS
      DO I = 1,PTRAIN
      DO J = 1,N
       X(I).XP(J) = XI(I).XP(J)
      END DO
      X(I).CLASS = XI(I).CLASS
     END DO

     CALL SUPERVISED(N,PTRAIN,X)

         write(6,*) 'number of clusters'
         write(6,*) fin_cnum

         write(6,*) 'size of centroid vector'
         write(6,*) n

     WRITE(6,360)
     DO I = 1,FIN_CNUM
      WRITE(6,300) I,FINAL_C(I).CRHO,FINAL_C(I).NUM,FINAL_C(I).CLASS
          do j = 1,n
           write(6,*) final_c(i).b(j)
          end do
     END DO !(I=1,FIN_CNUM)
     WRITE(6,*) ' '


     GO TO 999 !End of Program

****************************************************************
*****    ERROR PROCESSING          **********
*****                   **********
****************************************************************
!800    WRITE(6,*) 'ERROR OPENING THE INPUT FILE'
!      WRITE(6,*) 'CHECK FILE AND TRY AGAIN'
!      GO TO 999  !Exit Program

900    WRITE(6,*) 'ERROR OPENING THE OUTPUT FILE'
     WRITE(6,*) 'CHECK FILE AND TRY AGAIN'
     GO TO 999   !Exit Program

999    END   ! (* program *)
```

A.3 PARAMS.INC

PARAMS.INC is an include file which sets limits on the arrays that are generated by the SUPER.FOR and UNSUPER.FOR programs.

```
! Maximum number of cycles allocated
     INTEGER*4      CP
!     PARAMETER      (CP=40)
         parameter     (cp=1)
```

```
! Maximum number of points allocated for the input vector
! before feature extraction
      INTEGER*4      NP
!      PARAMETER      (NP=6000)
           PARAMETER    (NP=1) !ISAP
! Maximum number of channels allocated
      INTEGER*4      LP
!      PARAMETER      (LP=4)
       PARAMETER        (LP=1) !ISAP
! Maximum number of points allocated for a vector
! after feature extraction
      INTEGER*4      NMAX
!      PARAMETER      (NMAX=1500)
!          Parameter           (nmax=160) !reg ANN
!      parameter     (nmax=240) !Damage Diagnosis--Scenerio 1
       parameter   (nmax=2)

! Maximum number of final clusters
           INTEGER*4      FCMAX
!          PARAMETER    (FCMAX = 80)  !Scenario 1
      parameter      (fcmax = 100)   !Scenario 2

! Maximum number of reflection coefficients to be computed
      INTEGER*4      MP
!      PARAMETER      (MP=8)
       PARAMETER        (MP=1) !ISAP
! Maximum number of patterns allocated
      INTEGER*4      PP
!           PARAMETER   (PP=5000)
!           PARAMETER   (PP=80)  !Scenario 1
       parameter   (pp=100)

           INTEGER*4      KMAX
           PARAMETER    (KMAX=20)

!5/24/95 --- Not needed when not running with feature extraction module
c      COMPLEX*16     FPhaN(NP)
!      COMPLEX*16     FPhaA(NP)
!      COMPLEX*16     FPhaB(NP)
!      COMPLEX*16     FPhaC(NP)

c      COMMON        /BLK11/FPhaN
!      COMMON        /BLK12/FPhaA
!      COMMON        /BLK13/FPhaB
!      COMMON        /BLK14/FPhaC
```

A.2 UNSUPER.FOR

UNSUPER.FOR is a FORTRAN program that creates a set of clusters using unsupervised learning so that a group of patterns lie within a radius RHO.

```
            SUBROUTINE UNSUPERVISED
*******************************************************************************
**     THIS SUBROUTINE IMPLEMENTS AN UNSUPERVISED Learning ALGORITHM
**     The Algorithm is a Clustering type
**     We are given a set of P patterns, X
**     We want to find a family of clusters (hyper)spherical clusters, Ck
**     so that all patterns in a cluster are within some sphere radius,Rho
**
*******************************************************************************
      IMPLICIT NONE
      INCLUDE 'PARAMS.INC'

      INTEGER*4   I,J,K,M        !Indexes for DO loops
      INTEGER*4   R            !Indexes
      INTEGER*4   CNUM             !Counter of clusters
      INTEGER*4   CLUS
      INTEGER*4   PATT           !
      INTEGER*4   CHAN_CTR   !Counts num of passes w/ no change in stab run
         LOGICAL*2   DONE       !Two passes through stab run w/ no changes
      LOGICAL*2   FINISHED        !Loop Control
      INTEGER*4   P             !Number of Patterns
      INTEGER*4   N             !Size of Pattern Vector
      REAL*8      RHO          !Sphere Radius
      REAL*8      RHOTWO        !Square of RHO
      REAL*8      RAD      !Distance bet pattern and cluster centroid
      REAL*8      INTSUM    !Intermediate Sum Term
      REAL*8      BP(NMAX)   !Centroid of Data Set
      REAL*8      DT(PP)     !Distance between each pattern & the center
         REAL*8     D(FCMAX)    !Distance between each pattern and all clusters
      REAL*8      DIS,MIN
      LOGICAL     FIRST      !Signifies first iteration through program
         integer*4          epoch   !counts # of times patts are presented to ANN
         INTEGER*4        II,JJ
         LOGICAL            NEAR
         LOGICAL            BAD_SET  !Signifies patts couldn't be placed in 1 cluster
         REAL*8     JTOTAL,JT(FCMAX)   !Sum of all Jk ; Jk
         REAL*8     OLD     !Intermediate variable

      STRUCTURE  /XPATTERN/
      REAL*8   XP(NMAX)  !Normalized Input Pattern: index reps vector pos'n
      INTEGER*4 XELEM    !# of Cluster pattern belongs to
      CHARACTER*25  CLASS           !The class membership of the pattern
      END STRUCTURE

      RECORD   /XPATTERN/ X

      STRUCTURE /CLUSTER/
      REAL*8   B(NMAX)    !Centroid Vector
      REAL*8   CRHO          !Sphere Radius
      INTEGER*4 NUM            !Number of Patterns in the Cluster
      INTEGER*4 PATT_NUM (PP) !Identifies patterns that belong to Cluster
      END STRUCTURE

      RECORD   /CLUSTER/ C

      COMMON /BLK1/X(PP)
      COMMON /BLK2/C(PP)
          common /blk3/rho,rhotwo,n,p,cnum
!3/19/95       COMMON /BLK3/FIRST,RHO,RHOTWO,N,P,CNUM,BAD_SET
          common /blk4/first,bad_set

!2/3/95
     open(unit=11,file='jtotal.out',status='new')
     open(unit=12,file='distance.out',status='new')
!2/3/95

**************************************************************************
****        BEGIN INITIALIZATION PHASE           *******************
****        EXECUTED ONE TIME!               *******************
**************************************************************************

!   Find the center of the entire data set
      DO J = 1,N
       INTSUM = 0.0
       DO I = 1,P
        INTSUM = INTSUM + X(I).XP(J)
       END DO !(I)
       BP(J) = INTSUM/(FLOAT(P))
      END DO !(J)
      CALL NORM(N,BP)       !NORMALIZE BP
!   Compute distances between each pattern and Bj
      DO I = 1,P
        CALL DISTANCE(DIS,X(I).XP,BP,N)
        DT(I) = DIS
      END DO
!    Reorder the patterns in increasing order based on distance
      CALL SORT(P,DT,N,X)
!    Select the RHO as the value of the largest distance to ensure
!    that all patterns fall in one cluster


      IF ((FIRST) .AND. (CNUM .GT. 1)) THEN
       RHOTWO = RHOTWO+(0.05*rhotwo)
           if (rhotwo .gt. 1.0000) then
! patterns are more than rho=1.0 distance away from the center
                 bad_set = .true.
                 go to 1000
           end if
       RHO = SQRT(RHOTWO)
      ELSEIF (FIRST) THEN
           rhotwo =  1.05*dt(p) !3/10
           rho = sqrt(rhotwo) !3/19/95
!         if (rhotwo .gt. 1.0000) then
           if (rho .gt. 1.00000) then
            rhotwo = 1.0
```

```
              rho = 1.0 !3/19/95
            end if
!3/19/95        RHO = SQRT(RHOTWO)
      END IF

!3/19/95 want to see what the values of the distances are
!          if (first)  then
!        write(11,*) 'dt(i)'
!            do i = 1, p
!              write(11,*) dt(i)
!            end do
!            write(11,*) 'end'
!          end if
!3/19/95


!   Form Cluster 1 with pattern #1
      DO I = 1,N
       C(1).B(I) = X(1).XP(I)
      END DO
      C(1).CRHO = RHO
      C(1).NUM = 1
      C(1).PATT_NUM(1) = 1
      X(1).XELEM = 1
!   For remaining patterns,
!   Check to see if pattern belongs to existing cluster or if a
!   new cluster should be created
      CNUM = 1
      DO J = 2,P
       FINISHED = .FALSE.
       M = 1
        DO WHILE (.NOT. FINISHED)
!   Compute distance between pattern and center of cluster M
          CALL DISTANCE(DIS,X(J).XP,C(M).B,N)
          RAD = DIS
          IF (RAD .LE. RHOTWO) THEN
!   If distance is <= RHO, pattern in placed in cluster M
            DO  K = 1,N
              C(M).B(K) = (C(M).B(K) + (1.0/(C(M).NUM+1.0))*
     *                   (X(J).XP(K)- C(M).B(K)))
            END DO !(K)
            CALL NORM(N,C(M).B)
            C(M).NUM = C(M).NUM + 1
            C(M).PATT_NUM(C(M).NUM) = J
            X(J).XELEM = M
            FINISHED = .TRUE.
           ELSE
!   If distance is > RHO
             M = M+1
!    Have all clusters been checked
!    Else check next cluster
            IF (M .GT. CNUM) THEN  !All clusters checked?
             CNUM = CNUM + 1      !Form new cluster
             DO I = 1,N          !Update cluster params
              C(CNUM).B(I) = X(J).XP(I)
              END DO
!             CALL NORM(N,C(CNUM).B)
              C(CNUM).NUM = 1
              C(CNUM).PATT_NUM(1) = J
              C(CNUM).CRHO = RHO
              X(J).XELEM = CNUM    !Update pattern ptr to cluster
              FINISHED = .TRUE.
            END IF !(M>=CNUM)
          END IF !(RAD<RHOTWO)
        END DO !(DO WHILE)
      END DO !(J)


!2/3/95
      write(11,*) 'first =   ',first
      write(11,*) 'initialization phase'
           write(11,*) ' '
!2/3/95

!3/27/95
!Calculate the original value of Jt(I) for each cluster to be minimized
           if ((rho .lt. 0.430) .and. (rho .gt. 0.410)) then
             write(11,*) 'JT(I)'
             write(11,*) ' '
             DO I = 1,CNUM
              JT(I) = 0.0
              DO J = 1,C(I).NUM
                CALL DISTANCE(DIS,X(C(I).PATT_NUM(J)).XP,C(I).B,N)
                JT(I) = JT(I) + DIS
              END DO
              write(11,*) JT(I)
             END DO
             write(11,*) ' '
           end if
!3/27/95

****************************************************************
*****        END OF INITIALIZATION PHASE          **********
*****                                   **********
****************************************************************

**********************************************************************
*****        BEGIN STABILIZATION PHASE          ****************
*****                                   ****************
**********************************************************************

           write(11,*) 'stabilization phase'
!3/27/95
!Compute JTotal for intital partition
           JTOTAL = 0.0
           DO I = 1,CNUM
            DO J = 1,C(I).NUM
              CALL DISTANCE(DIS,X(C(I).PATT_NUM(J)).XP,C(I).B,N)
              JTOTAL = JTOTAL + DIS
            END DO
          END DO
          write(11,*) 'JTOTAL   ',JTOTAL
!3/27/95

!Initialize Counter that records the num of patterns that changed clusters
       CHAN_CTR = 0          ! # of changes
       DONE = .FALSE.

       IF (CNUM .NE. 1) THEN  !if more than one cluster
       epoch = 0 !1-25-94
       DO WHILE (.NOT. DONE)
            epoch = epoch + 1 !1-25-94
! Calc shortest distance between each pattern and each cluster's centroid
       DO K = 1,P

         DO M = 1,CNUM
           CALL DISTANCE(DIS,X(K).XP,C(M).B,N)
           D(M) = DIS
         END DO
!  Sort the distances in increasing order, the min distance is in MIN
!  The num of the cluster whose centroid rep the min distance is in PATT
         CALL MIN_DIS(CNUM,D,MIN,PATT)

!  If the min_dis is the same for the 1st and 2nd nearest cluster, and
!  the pattern already resides in one of these clusters.  Leave it
!  in the cluster in which it resides
            NEAR = .FALSE.
            DO II = 1,CNUM
          IF ((ABS(MIN-D(II)) .LT. 1e-15) .AND.
     *                 (II .NE. PATT)) THEN
                NEAR = .TRUE.
                JJ = II
              END IF
            END DO
            IF (NEAR) THEN
          IF (JJ .EQ. X(K).XELEM) THEN
                   PATT = X(K).XELEM
              END IF
            END IF

         CLUS = X(K).XELEM          !old cluster number
         IF (MIN .LE. RHOTWO) THEN

          IF (CLUS .NE. PATT) THEN
!  Remove the pattern from old cluster, CLUS
!  Move the pattern new cluster, PATT
!  Adapt the params for CLUS & PATT

           DO  R = 1,N      !add pattern to other cluster
            C(PATT).B(R) = (C(PATT).B(R) +
     *     (1.0/(C(PATT).NUM+1.0))*(X(K).XP(R)- C(PATT).B(R)))
           END DO !(R)
           CALL NORM(N,C(PATT).B)
           C(PATT).NUM = C(PATT).NUM + 1
           C(PATT).PATT_NUM(C(PATT).NUM) = K
          X(K).XELEM = PATT

!Before deleting pattern from cluster, check if last pattern in cluster
          IF ((C(CLUS).NUM-1) .EQ. 0) THEN      !last patten in CLUS
!delete cluster, CLUS
!switch last cluster, C(CNUM), w/ cluster being deleted
           IF (CLUS .NE. CNUM) THEN     !switch clusters
            DO R = 1,N
             C(CLUS).B(R) = C(CNUM).B(R)
            END DO !(R=1,N)
            C(CLUS).CRHO = C(CNUM).CRHO
            C(CLUS).NUM = C(CNUM).NUM
            DO R = 1,C(CLUS).NUM   !switch pattern nums & clus ptr
             X(C(CNUM).PATT_NUM(R)).XELEM = CLUS
             C(CLUS).PATT_NUM(R) = C(CNUM).PATT_NUM(R)
            END DO !(R=1,C(CLUS).NUM)
           END IF !(CLUS .NE. CNUM) -- else do nothing
           CNUM = CNUM - 1

!Remove pattern from cluster
          ELSE  !(not last pattern in cluster)
           R = 1
           DO WHILE (C(CLUS).PATT_NUM(R) .NE. K)
            R = R + 1
           END DO
!pattern not in last position, switch last pattern into place of deleted pattern
           IF (R .NE. C(CLUS).NUM) THEN  !switch pattern
            C(CLUS).PATT_NUM(R) = C(CLUS).PATT_NUM(C(CLUS).NUM)
           END IF
           DO R = 1,N                !adjust centroid
            C(CLUS).B(R) = (C(CLUS).B(R) -
  *          (1.0/(C(CLUS).NUM-1.0))*(X(K).XP(R)- C(CLUS).B(R)))
           END DO
           CALL NORM(N,C(CLUS).B)
           C(CLUS).NUM = C(CLUS).NUM - 1

          END IF !(c(clus).num-1 = 0)

         END IF !(clus .ne. patt)

        ELSE  !remove pattern from old cluster and form a new cluster

!Before deleting pattern from cluster,CLUS, check if last pattern in cluster
          IF ((C(CLUS).NUM-1) .EQ. 0) THEN      !last patten in CLUS
!delete cluster, CLUS
!switch last cluster, C(CNUM), w/ cluster being deleted
           IF (CLUS .NE. CNUM) THEN     !switch clusters
            DO R = 1,N
             C(CLUS).B(R) = C(CNUM).B(R)
            END DO !(R=1,N)
            C(CLUS).CRHO = C(CNUM).CRHO
            C(CLUS).NUM = C(CNUM).NUM
            DO R = 1,C(CLUS).NUM   !switch pattern nums & clus ptr
             X(C(CNUM).PATT_NUM(R)).XELEM = CLUS
             C(CLUS).PATT_NUM(R) = C(CNUM).PATT_NUM(R)
```

```
                  END DO !(R=1,C(CLUS).NUM)
                END IF !(CLUS .NE. CNUM) -- else do nothing
                CNUM = CNUM - 1

!Remove pattern from cluster
              ELSE  !(not last pattern in cluster)
                R = 1
                DO WHILE (C(CLUS).PATT_NUM(R) .NE. K)
                 R = R + 1
                END DO
!pattern not in last position, switch last pattern into place of deleted pattern
                IF (R .NE. C(CLUS).NUM) THEN  !switch patterns
                 C(CLUS).PATT_NUM(R) = C(CLUS).PATT_NUM(C(CLUS).NUM)
                END IF
                DO R = 1,N                !adjust centroid
                 C(CLUS).B(R) = (C(CLUS).B(R) -
   *              (1.0/(C(CLUS).NUM-1.0))*(X(K).XP(R)- C(CLUS).B(R)))
                END DO
                CALL NORM(N,C(CLUS).B)
                C(CLUS).NUM = C(CLUS).NUM - 1

            END IF !(c(clus).num-1 = 0)

            CNUM = CNUM + 1    !Form new cluster
            DO  R = 1,N        !Update cluster params
              C(CNUM).B(R) = X(K).XP(R)
            END DO !(R)
!           CALL NORM(N,C(CNUM).B)
            C(CNUM).NUM = 1
            C(CNUM).PATT_NUM(1) = K
            C(CNUM).CRHO = RHO
            X(K).XELEM = CNUM  !Update pattern ptr to cluster

           END IF !(RK .NE. RHOTWO)

!3/27/95
!Calculate the value of Jt(I) for each cluster
              if ((rho .lt. 0.430) .and. (rho .gt. 0.410)) then
               DO I = 1,CNUM
                old = jt(i)
                JT(I) = 0.0
                DO J = 1,C(I).NUM
                 CALL DISTANCE(DIS,X(C(I).PATT_NUM(J)).XP,C(I).B,N)
                 JT(I) = JT(I) + DIS
                END DO
                if (jt(i) .ne. old) then
                  write(11,*) k,i,c(i).num,JT(I)
                end if
               END DO
               write(11,*) ' '
              end if
!3/27/95

          END DO  !(K)
```

```
!Calculate the value of objective function to be minimized
          OLD = JTOTAL
          JTOTAL = 0.0
          DO I = 1,CNUM
           DO J = 1,C(I).NUM
            CALL DISTANCE(DIS,X(C(I).PATT_NUM(J)).XP,C(I).B,N)
            JTOTAL = JTOTAL + DIS
           END DO
          END DO
! We want to continue looping if the JTotal value is changed
! If it doesn't change values for two consecutive iterations, stop
! If the new value of JTOTAL changed, init chan_ctr =0
! Else incr chan_ctr (no change)
          IF (JTOTAL .EQ. OLD) THEN
           CHAN_CTR = CHAN_CTR + 1
          ELSE
           CHAN_CTR = 1
          END IF
          IF (CHAN_CTR .GE. 2) THEN
           DONE = .TRUE.
          END IF


      END DO  !(DO WHILE)
     END IF !(CNUM .NE. 1)

**************************************************************
*****        END  STABILIZATION PHASE           **********
*****                                **********
**************************************************************


!2/3/95
         write(11,*) 'after stabilization'
     write(11,*) 'epoch number  ',epocH
         write(11,*) ' '


1000     RETURN
     END   ! (* program *)


      SUBROUTINE SORT(NN,RA,P,XX)
**************************************************************
***** From "Numerical Recipes" (p. 231)
***** Sorts an array RA of length NN into ascending numerical order
***** using the Heapsort algorithm, while making the corresponding
***** rearrangement of the array XX.  NN is input; RA is replaced on
***** output by its sorted rearrangement
***** The Heapsort Algorithm is a NN(log2)NN process
**************************************************************
     IMPLICIT NONE
     INCLUDE 'PARAMS.INC'

     INTEGER*4      NN     !Array Size
```

```
      INTEGER*4      P      !Vector Size
      REAL*8         RA(PP)
      REAL*8         RRA
      REAL*8         RRB(PP)
      CHARACTER*25   RRC
      INTEGER*4      L,IR,K,I,J

      STRUCTURE  /XPATTERN/
      REAL*8           XP(NMAX)  !Normalized Input Pattern: index reps vector pos'n
      INTEGER*4 XELEM    !# of the Cluster that pattern belongs to
      CHARACTER*25 CLASS  !Class membership label
      END STRUCTURE
      RECORD   /XPATTERN/ XX(PP)

      L = (NN/2) + 1
      IR = NN
*****  The index L will be decremented from its initial value down to
*****  1 during the "hiring" (heap creation) phase.  Once it reaches 1
*****  the index IR will be decremented from its initial value down
*****  to 1 during the "retirement-and-promotion" (heap selection)
******  phase.
10     CONTINUE
       IF (L.GT.1) THEN              !Still in hiring phase
        L = L-1
        RRA = RA(L)
        DO K = 1,P
         RRB(K) = XX(L).XP(K)
        END DO
        RRC = XX(L).CLASS
       ELSE           !In retirement-and-promotion phase
        RRA = RA(IR)                !Clear a space at end of array
        DO K = 1,P
         RRB(K) = XX(IR).XP(K)
        END DO
        RRC = XX(IR).CLASS
        RA(IR) = RA(1)              !Retire the top of the heap into it
        DO K = 1,P
         XX(IR).XP(K) = XX(1).XP(K)
        END DO
        XX(IR).CLASS = XX(1).CLASS
        IR = IR -1                !Decrease the size of the corp.
        IF (IR.EQ.1) THEN           !Done with the last promotion
         RA(1) = RRA              !The lease competent worker of all!
         DO K = 1,P
          XX(1).XP(K) = RRB(K)
         END DO
         XX(1).CLASS = RRC
         RETURN
        END IF
       END IF
      I = L          !Set up to sift down element RRA
      J = L+L         !to its proper level
20     IF (J.LE.IR) THEN              !"Do while J.LE.IR"
        IF (J.LT.IR) THEN
         IF (RA(J).LT.RA(J+1)) THEN   !Compare to the better underling
      J = J + 1
     END IF
    END IF

    IF (RRA.LT.RA(J)) THEN         !Demote RRA
     RA(I) = RA(J)
     DO K = 1,P
      XX(I).XP(K) = XX(J).XP(K)
     END DO
     XX(I).CLASS = XX(J).CLASS
     I = J
     J = J + J
    ELSE        !This is RRA's level.
     J = IR + 1  !Set J to terminate the sift-down
    END IF
   GO TO 20
   END IF
   RA(I) = RRA      !Put RRA into its slot
   DO K = 1,P
    XX(I).XP(K) = RRB(K)
   END DO
   XX(I).CLASS = RRC
   GO TO 10
   END

     SUBROUTINE MIN_DIS(NUM,D,MIN,CLUST)
****************************************************************
****   This subroutine finds the minimun distance between the
****   test pattern and each cluster.  MIN represents the distance
****   and CLUST represents the number of the cluster that is the
****   minimum distance from the test pattern
****************************************************************
    IMPLICIT NONE
    INCLUDE 'PARAMS.INC'

    INTEGER*4      I,NUM,CLUST
    REAL*8         D(FCMAX),MIN

    MIN = 100000000
    DO I = 1,NUM
     IF (D(I) .LT. MIN) THEN
       MIN = D(I)
       CLUST = I
     END IF
    END DO  !(I)
   RETURN
   END

   SUBROUTINE DISTANCE(DIS,X,B,NN)
************************************************************
***   This subroutine computes the distance between
***   the test pattern and the center
****************************************************************
    IMPLICIT NONE
    INCLUDE 'PARAMS.INC'
```

```
      REAL*8        X(NMAX)
      REAL*8        B(NMAX)
      REAL*8        DIS,INTSUM
      INTEGER*4     J,NN

      INTSUM = 0.0
      DO J = 1,NN
       INTSUM = INTSUM + ((X(J) - B(J))**2)
      END DO
      DIS = INTSUM
      RETURN
      END


     SUBROUTINE NORM(N,X)

     IMPLICIT NONE
     INCLUDE 'PARAMS.INC'

     INTEGER*4     N,J
     REAL*8        XNORM,X(NMAX)

     XNORM = 0.0
     DO J = 1,N
      XNORM = XNORM + (X(J)**2)
     END DO !(J=1,N)
     XNORM = DSQRT(XNORM)
     IF (XNORM .EQ. 0.0) THEN
      XNORM = 0.0000001
     END IF
     DO J = 1,N
       X(J) = X(J)/XNORM
     END DO !(J=1,N)

     RETURN
END
```

A.5 SUPER.FOR

SUPER.FOR is a FORTRAN program that uses supervised learning to determine if the clusters generated from UNSUPER.FOR are homogeneous.

```
      SUBROUTINE SUPERVISED(NTRAIN,PTRAIN,XTRAIN)
*****************************************************************
**  THIS PROGRAM IMPLEMENTS A SUPERVISED Learning ALGORITHM
**  The Algorithm is a Clustering type
**  We are given a set of PTRAIN patterns, XTRAIN
**  We want to find a family of clusters (hyper)spherical clusters, Ck
**  so that all patterns in a cluster are within some sphere radius,Rho
**  The supervised learning calls the unsupervised learning algorithm
**  which generates a set of natural clusters.  Then the supervised
**  learning removes those clusters which are composed of homogeneous
**  elements.
**
**
**  This supervised algorithm has been enhanced to include:
**    (1) merging of clusters, (2) tightening of clusters,
**    (3) adding probabilities to clusters
**
**    Inputs:
**          NTRAIN -- Vector size of training patterns
**          PTRAIN -- Number of training patterns
**          XTRAIN --  PTRAIN Structures of Training patterns
**
**    Common:
**          Final_C -- contains the clusters generated by the routine
**          Fin_CNUM -- final number of clusters
**
***************************************************************************
      IMPLICIT NONE
      INCLUDE 'PARAMS.INC'

           INTEGER*4      NTRAIN,PTRAIN
      INTEGER*4     I,J,K,M      !Indexes for DO loops
      INTEGER*4     L,R          !Indexes
      INTEGER*4     CNUM          !Number of ActiveClusters
      INTEGER*4     FIN_CNUM     !Number of Final Clusters
      INTEGER*4     P            !Number of Patterns
      INTEGER*4     N            !Size of Pattern Vector
      REAL*8        RHO          !Sphere Radius
      REAL*8        RHOTWO          !Square of RHO
      REAL*8        NORM         !Normalized Value of Input Pattern
      REAL*8        D(FCMAX)         !Distance between each pattern & the center
      REAL*8        DECR         !Interval by which RHO will be decremented
      LOGICAL       STOP         !Parameter that stops DO loop
      INTEGER*4     FX_CTR          !Number of patterns in final set
      INTEGER*4     P_CTR         !Total Number of patterns
      LOGICAL       FIRST        !Signifies first iteration through program
      INTEGER*4     MATCH      !Counts num of patterns in cluster that match
      REAL*8        DIS       !Distance between two vectors
      REAL*8        NEW_DIS !Used to hold distance value until an update
      INTEGER*4     NEW_CLUS  !Used to hold value of cluster until an update
      REAL*8        DEPSILON  !Min distance between two cluster
      REAL*8        PEPSILON
!Distance to be added to fartherest pattern in cluster to adjust radius
!     PARAMETER     (PEPSILON = 0.15)
           parameter        (pepsilon = 0.20)
          LOGICAL                   BAD_SET  !Signifies patts couldn't be place in 1 cluster

      STRUCTURE /XPATTERN/
      REAL*8     XP(NMAX)  !Normalized Input Pattern:
      INTEGER*4   XELEM    !# of Cluster pattern belongs to
      CHARACTER*25  CLASS !The class membership of the pattern
      END STRUCTURE

      STRUCTURE /CLUSTER/
      REAL*8      B(NMAX)   !Centroid Vector
      REAL*8    CRHO        !Square of Sphere Radius
      INTEGER*4 NUM        !Number of Patterns in the Cluster
      INTEGER*4 PATT_NUM(PP) !Identifies patterns that belong to Cluster
      END STRUCTURE

      STRUCTURE /TEMP_XPATTERN/
      REAL*8      XP(NMAX)  !Normalized Input Pattern: index reps vector pos'n
      CHARACTER*25  CLASS         !The class membership of the pattern
      END STRUCTURE

      STRUCTURE /FIN_CLUSTER/
      REAL*8     B(NMAX)    !Centroid Vector
      REAL*8     CRHO     !Square of Sphere Radius
      INTEGER*4  NUM        !Number of Patterns in the Cluster
      CHARACTER*25  CLASS  !The class membership of the pattern
      REAL*8     PROB     !Probability of this cluster
      END STRUCTURE

      RECORD   /XPATTERN/ X,XTRAIN(PP)
      RECORD   /CLUSTER/  C
      RECORD /XPATTERN/ FINAL_X
      RECORD /FIN_CLUSTER/ FINAL_C
      RECORD   /TEMP_XPATTERN/TEMP(PP)


      COMMON /BLK1/X(PP)
      COMMON /BLK2/C(PP)
!3/19/95        COMMON /BLK3/FIRST,RHO,RHOTWO,N,P,CNUM,BAD_SET
         common /blk3/rho,rhotwo,n,p,cnum
         common /blk4/first,bad_set
      COMMON /BLK5/FINAL_X(PP)
      COMMON /BLK6/FINAL_C(PP)
      COMMON /BLK7/FIN_CNUM

C      OPEN(UNIT=5,STATUS='OLD',ERR=100)
c     OPEN(UNIT=6,STATUS='NEW',ERR=200,)


305   FORMAT(I5)
310   FORMAT(F6.4)
315   FORMAT(10F7.4)
316   FORMAT(10F7.3)
350   FORMAT(A25)
351   FORMAT(2X,A25)
205   FORMAT(' VECTOR SIZE:',3X,I5,/' NUMBER OF PATTERNS:',3X,I5/)
210   FORMAT(' NUMBER OF PATTERNS IN CLUSTER:',3X,I5/)
215   FORMAT(' CLUSTER NUMBER:',3X,I5/)
220   FORMAT(10F10.6)
230   FORMAT(' CLASS MEMBERSHIP OF CLUSTER:',3X,'"',A25,'"',//)
240   FORMAT(' RHO OF CLUSTER:',3X,F8.6/)
```

```
      P_CTR = PTRAIN
      P = PTRAIN
      N = NTRAIN
      DO I = 1,P
       DO J = 1,N
        X(I).XP(J) = XTRAIN(I).XP(J)
       END DO
       X(I).CLASS = XTRAIN(I).CLASS
      END DO

!   Normalize each pattern, using ||Xi|| = 1
      DO J = 1,P
       NORM = 0
       DO K = 1,N
        NORM = NORM + (X(J).XP(K)**2)
       END DO !(K)
       NORM = SQRT(NORM)
       IF (NORM .EQ. 0.0) THEN
        NORM = .0000001
       END IF
       DO K = 1,N
        X(J).XP(K) = (X(J).XP(K)/NORM)
       END DO !(K)
      END DO ! (J)


!**************************************************************************
!****  Set up One Cluster that Contains all Patterns                  ****
!****                                                    ***
!**************************************************************************
      FIRST = .TRUE.
          BAD_SET = .FALSE.
      CNUM = 0
      CALL UNSUPERVISED
!Check to see if there is more than one cluster
      DO WHILE (CNUM .GT. 1)
       CALL UNSUPERVISED
           IF (BAD_SET) THEN
             rhotwo = 1.0 + (1.0/75.0)
             cnum = 1
           END IF
      END DO !(DO WHILE)   !Set up interval to decrement
       DECR = RHOTWO/75.0
       RHOTWO = RHOTWO - DECR
       RHO = SQRT(RHOTWO)

!Initialize Main Loop Variables
       FIN_CNUM = 0
       FX_CTR = 0
       STOP = .FALSE.
       FIRST = .FALSE.

**************************************************************************
```

```
****           Main Loop of Program        ***
****                     ***
****************************************************************************
      DO WHILE ((RHO .GT. 0.0) .AND. (.NOT. STOP))
       CALL UNSUPERVISED  !   Check each Cluster

       IF ((CNUM .NE. 1) .OR.
     *     ((CNUM .EQ. 1) .AND. (FX_CTR .GT. 0))) THEN

       DO J = 1,CNUM

       IF (C(J).NUM .EQ. 1) THEN  !(one pattern in cluster)
        FIN_CNUM = FIN_CNUM + 1  !insert cluster in final set
        FINAL_C(FIN_CNUM).CRHO = C(J).CRHO
        FINAL_C(FIN_CNUM).NUM = 1
        DO M = 1,N
         FINAL_C(FIN_CNUM).B(M) = C(J).B(M)
        END DO !(M)
        FINAL_C(FIN_CNUM).CLASS = X(C(J).PATT_NUM(1)).CLASS
! Insert pattern in final set
        FX_CTR = FX_CTR + 1
        DO L = 1,N
         FINAL_X(FX_CTR).XP(L) = X(C(J).PATT_NUM(1)).XP(L)
        END DO
        FINAL_X(FX_CTR).XELEM = FIN_CNUM
        FINAL_X(FX_CTR).CLASS = X(C(J).PATT_NUM(1)).CLASS
        CNUM = CNUM - 1  !  remove cluster from active set
        X(C(J).PATT_NUM(1)).XELEM = 0 !remove pattern from active set
! If there is more than one pattern in cluster
! Determine if all patterns in the cluster belong to the same class
       ELSE !(C(J).NUM > 1)

        MATCH = 0
        DO K = 2,C(J).NUM
         IF (X(C(J).PATT_NUM(K)).CLASS .EQ.
     *       X(C(J).PATT_NUM(1)).CLASS) THEN
          MATCH = MATCH + 1
         END IF
        END DO !(K)
! If they belong to the same class, remove the cluster from consideration
        IF (MATCH .EQ. (C(J).NUM - 1)) THEN  !  Insert cluster in final set
         FIN_CNUM = FIN_CNUM + 1
         FINAL_C(FIN_CNUM).CRHO = C(J).CRHO
         FINAL_C(FIN_CNUM).NUM = C(J).NUM
         DO M = 1,N
          FINAL_C(FIN_CNUM).B(M) = C(J).B(M)
         END DO !(M)
         FINAL_C(FIN_CNUM).CLASS = X(C(J).PATT_NUM(1)).CLASS
! Insert patterns in final set
         DO M = 1,C(J).NUM
          FX_CTR = FX_CTR + 1
          DO L = 1,N
           FINAL_X(FX_CTR).XP(L) = X(C(J).PATT_NUM(M)).XP(L)
          END DO !(l=1,N)
          FINAL_X(FX_CTR).XELEM = FIN_CNUM
```

```
            FINAL_X(FX_CTR).CLASS = X(C(J).PATT_NUM(M)).CLASS
           END DO  !(M)  !  Remove cluster from active set
           CNUM = CNUM - 1  !  Remove patterns from active set
           DO L = 1,C(J).NUM
            X(C(J).PATT_NUM(L)).XELEM = 0
           END DO  !(L)
          END IF  ! (Match = C(J).NUM)
         END IF !(C(J).NUM > 1)

        END DO  !(J)
        END IF !(FX_CTR NE 0  AND CNUM NE 1)

!  If all patterns have not been placed in final set,
!  Assemble set of patterns remaining in active set
!  Place remaining patterns in a temporary holder, TEMP
        IF ((FX_CTR .GT. 0) .AND. (FX_CTR .LT. P_CTR)) THEN
         M = 0
         DO L = 1,P
          IF (X(L).XELEM .NE. 0) THEN
          M = M + 1
          DO R = 1,N
           TEMP(M).XP(R) = X(L).XP(R)
          END DO !(R)
          TEMP(M).CLASS = X(L).CLASS
          END IF !(X(L).XELEM .NE. 0)
         END DO !(L)
         P = M

!  Take patterns from temporary holder and place back in
!  original structure, X
        DO L = 1,M
         DO R = 1,N
          X(L).XP(R) = TEMP(L).XP(R)
         END DO !(R)
         X(L).CLASS = TEMP(L).CLASS
        END DO !(L)
       ELSEIF (FX_CTR .EQ. P_CTR) THEN
        STOP = .TRUE.
       ENDIF !((FX_CTR > 0) and (FX_CTR < P_CTR))

!  Update Rho
         IF (.NOT. STOP) THEN
          RHOTWO = RHOTWO - DECR
             IF (RHOTWO .LT. 0.0) THEN
              RHOTWO = 0.0
             END IF
          RHO = SQRT(RHOTWO)
         END IF !(NOT STOP)

        END DO  !(DO WHILE)

        IF (RHO .LE. 0.0) THEN
!  Insert remaining patterns in final set
         DO J = 1,P
          FIN_CNUM = FIN_CNUM+1
        FINAL_C(FIN_CNUM).NUM = 1
        DO M = 1,N
         FINAL_C(FIN_CNUM).B(M) = X(J).XP(M)
        END DO !(M)
        FINAL_C(FIN_CNUM).CLASS = X(J).CLASS
            FINAL_C(FIN_CNUM).CRHO = PEPSILON
        FX_CTR = FX_CTR + 1
        DO L = 1,N
         FINAL_X(FX_CTR).XP(L) = X(J).XP(L)
        END DO
        FINAL_X(FX_CTR).XELEM = FIN_CNUM
        FINAL_X(FX_CTR).CLASS = X(J).CLASS
        X(J).XELEM = 0 !remove pattern from active set

       END DO !(J)
      END IF !(RHO <= 0)



*******************************************************************************
****  End of Main Loop                    ***
****                              ***
*******************************************************************************

*****************************************************************************
****  Enhancements to Clusters
****
*****************************************************************************

! Tighten the radius of the clusters
! Find the pattern in cluster that is the fartherest distance from the center
! Change RHO to that distance plus PEPSILON
      DO I = 1,FIN_CNUM
       IF (FINAL_C(I).NUM .EQ. 1) THEN
         FINAL_C(I).CRHO = PEPSILON
       ELSEIF (FINAL_C(I).NUM .GT. 1) THEN
        K = 0
!1-1-94          DO J = 1,P_CTR
            DO J = 1,FX_CTR
         IF (FINAL_X(J).XELEM .EQ. I) THEN
          CALL DISTANCE(DIS,FINAL_X(J).XP,FINAL_C(I).B,N)
          K = K + 1
          D(K) = DIS
         END IF !(FINAL_X(J).XELEM = I)
        END DO !(J=1,P_CTR)
        NEW_DIS = 0.0
        DO J = 1,K
         IF (D(J) .GT. NEW_DIS) THEN
          NEW_DIS = D(J)
         END IF !(D(J) > NEW_DIS)
        END DO !(J=1,K)
        NEW_DIS = NEW_DIS + PEPSILON
! The distance between the fartherest pattern plus PEPSILON is less than
! the radius of Cluster I, then change it to that value
        IF (NEW_DIS .LT. FINAL_C(I).CRHO) THEN
        FINAL_C(I).CRHO = NEW_DIS
       END IF !(NEW_DIS < FINAL_C(I).CRHO)
      END IF !(FINAL_C(I).NUM = 1)
     END DO !(I=1,FIN_CNUM)


! Merge Clusters that are close to each other
! Compare Cluster I with all clusters to see if there is a cluster of
! the same class that is closer than DEPSILON
     DO I = 1,FIN_CNUM
      IF (FINAL_C(I).NUM .NE. 0) THEN
       NEW_DIS = 5.0
       DO J = 1,FIN_CNUM
        IF ((I .NE. J) .AND. (final_c(j).num .ne. 0) .and.
   *          (FINAL_C(I).CLASS .EQ. FINAL_C(J).CLASS)) THEN
         CALL DISTANCE(DIS,FINAL_C(I).B,FINAL_C(J).B,N)
!Cluster I is closer to Cluster J than to other clusters
         IF (DIS .LE. NEW_DIS) THEN
          NEW_DIS = DIS
          NEW_CLUS = J
         END IF !(DIS <= NEW_DIS)
        END IF !(I .NE. J)
       END DO !(J=1,FIN_CNUM)
!Cluster I is closer than sum of the two cluster radii
! and the two clusters are of the same class
!Merge Cluster I and Cluster NEW_CLUS
       IF (NEW_DIS .LE. 0.05*(FINAL_C(I).CRHO +
   *                FINAL_C(NEW_CLUS).CRHO)) THEN
        FINAL_C(I).NUM = 0
           DO J = 1,P_CTR
         IF (FINAL_X(J).XELEM .EQ. I) THEN !Pattern belongs to Cluster I
          FINAL_X(J).XELEM = NEW_CLUS
          DO K = 1,N !Add pattern to Cluster NEW_CLUS
           FINAL_C(NEW_CLUS).B(K) = (FINAL_C(NEW_CLUS).B(K) +
   *            (1.0/(FINAL_C(NEW_CLUS).NUM + 1.0)) *
   *            (FINAL_X(J).XP(K)-FINAL_C(NEW_CLUS).B(K)))
          END DO !(K=1,N)
          FINAL_C(NEW_CLUS).NUM = FINAL_C(NEW_CLUS).NUM + 1
         END IF !(FINAL_X(J).XELEM = I)
        END DO !(J=1,P_CTR)
!Set radius equal to distance of fartherest pattern plus PEPSILON
        DEPSILON = 0.0
        DO J = 1,P_CTR
         IF (FINAL_X(J).XELEM .EQ. NEW_CLUS) THEN
          CALL DISTANCE(DIS,FINAL_X(J).XP,FINAL_C(NEW_CLUS).B,N)
          IF (DIS .GT. DEPSILON) THEN
            DEPSILON = DIS
          END IF !(DIS > DEPSILON)
         END IF !(FINAL_X(J).XELEN = NEW_CLUS)
        END DO !(J=1,P_CTR)
        FINAL_C(NEW_CLUS).CRHO = DEPSILON + PEPSILON
       END IF !(NEW_DIS <= sum of the radii of the two clusters)
      END IF !(FINAL_C(I).NUM .NE. 0)
     END DO !(I=1,FIN_CNUM)
```

```
! Remove the empty clusters from the list after merging complete
      J = 0
      DO I = 1,FIN_CNUM
        IF (FINAL_C(I).NUM .NE. 0) THEN
          J = J + 1
          DO K = 1,N
            FINAL_C(J).B(K) = FINAL_C(I).B(K)
          END DO
          FINAL_C(J).CRHO = FINAL_C(I).CRHO
          FINAL_C(J).NUM = FINAL_C(I).NUM
          FINAL_C(J).CLASS = FINAL_C(I).CLASS
        END IF !(FINAL_C(I).NUM .NE. 0)
      END DO
      FIN_CNUM = J


! Add a probability value to the final clusters
! Probability equals the number of patterns in the cluster divided by
! the total number of training patterns
      DO I = 1,FIN_CNUM
         FINAL_C(I).PROB = (DFLOAT(FINAL_C(I).NUM)/DFLOAT(PTRAIN))
      END DO !(I=1,FIN_CNUM)


****************************************************************************
****  Enhancements to Clusters
****
****************************************************************************
      GO TO 999 !End of Program
*******************************************************************
*****    ERROR PROCESSING          **********
*****               **********
*******************************************************************
C100   WRITE(6,*) 'ERROR OPENING THE INPUT FILE'
C      WRITE(6,*) 'CHECK FILE AND TRY AGAIN'
C      GO TO 999  !Exit Program

C200   WRITE(6,*) 'ERROR OPENING THE OUTPUT FILE'
C      WRITE(6,*) 'CHECK FILE AND TRY AGAIN'
C      GO TO 999   !Exit Program

c1000   WRITE(6,*) 'COULD NOT GENERATE CLUSTERS FOR PATTERNS'
        GO TO 999

999    END   ! (* program *)
```

Appendix B: Revised Neural Network FORTRAN Programs

B.1 TRAIN.FOR

TRAIN.FOR is a FORTRAN program used by the revised neural network for training in conjunction with SUPER.FOR and UNSUPER.FOR.

```
      PROGRAM TRAIN
******************************************************************
**   THIS PROGRAM TRAINS THE CLUSTERING ANN
**   (1) EACH TRAINING PATTERN AND ITS RESPECTIVE  MEMBERSHIP CLASS
**   IS READ
**   (2) THE ANN IS TRAINED WITH THE DATA
**   (3) THE GENERATED CLUSTERS ARE WRITTEN TO AN OUTPUT FILE
**
*********************************************************************
      IMPLICIT NONE
      INCLUDE 'PARAMS.INC'

      INTEGER*4   I,J,K,M       !Indexes for DO loops
      INTEGER*4   L,R          !Indexes
      INTEGER*4   FIN_CNUM      !Number of Final Clusters
      INTEGER*4   P            !Number of Patterns
      INTEGER*4   N            !Size of Pattern Vector

      INTEGER*4  ICNTR,CNTR,PTRAIN

         INTEGER*4    NUM_PATTS        !Number of training patterns

      STRUCTURE  /XPATTERN/
      REAL*8      XP(NMAX)  !Normalized Input Pattern:
      INTEGER*4   XELEM    !# of Cluster pattern belongs to
      CHARACTER*25  CLASS !The class membership of the pattern
      END STRUCTURE

      STRUCTURE /FIN_CLUSTER/
      REAL*8     B(NMAX)    !Centroid Vector
      REAL*8     CRHO    !Square of Sphere Radius
      INTEGER*4  NUM       !Number of Patterns in the Cluster
      CHARACTER*25  CLASS  !The class membership of the pattern
      REAL*8    PROB      !Probability of cluster
      END STRUCTURE

      RECORD /XPATTERN/ X(PP),XI(PP)
      RECORD /XPATTERN/ FINAL_X
      RECORD /FIN_CLUSTER/ FINAL_C

      COMMON /BLK5/FINAL_X(PP)
      COMMON /BLK6/FINAL_C(PP)
      COMMON /BLK7/FIN_CNUM


      OPEN(UNIT=5,STATUS='OLD',ERR=900,FILE='ANNIN.DAT')
!     OPEN(UNIT=6,STATUS='NEW',ERR=900,FILE='ANNOUT.OUT')

300   FORMAT(I4,2X,F18.14,3X,I4,3X,A25)
360   FORMAT('CLUS#',4X,'RHO',13X,'NUM',4X,'MEMBER CLASS')

100   FORMAT(A5)
150   FORMAT(1X,F17.14,1X,F17.14,1X,F17.14,1X,F17.14)
160   FORMAT(1X,F17.14,2X,F17.14)
200   FORMAT(A5)
250   FORMAT(2X,F13.9,5X,F13.9)
350   FORMAT(2X,F15.9,2X,F15.9,2X,F15.9,2X,F15.9)
355     FORMAT(T1,A25)

        rEAD (5,*) N
        rEAD (5,*) NUM_PATTS

!      READ(5,*)  !Skip blank line

      DO J = 1,NUM_PATTS

! Read pattern
        DO M = 1,N
          READ(5,*) XI(J).XP(M)
        END DO !(K=1,N)
! Read Membership class
            READ(5,355) XI(J).CLASS

      END DO !(J=1,num_patts)


        PTRAIN = NUM_PATTS
      DO I = 1,PTRAIN
        DO J = 1,N
          X(I).XP(J) = XI(I).XP(J)
        END DO
        X(I).CLASS = XI(I).CLASS
      END DO

      CALL SUPERVISED(N,PTRAIN,X)

        write(6,*) fin_cnum

        do I=1,fin_cnum
                do j=1,n
                        write(6,*)final_c(i).b(j)
                end do
        end do

     GO TO 999 !End of Program
```

```
****************************************************************
*****    ERROR PROCESSING            **********
*****                  **********
****************************************************************
!800    WRITE(6,*) 'ERROR OPENING THE INPUT FILE'
!      WRITE(6,*) 'CHECK FILE AND TRY AGAIN'
!      GO TO 999  !Exit Program

900    WRITE(6,*) 'ERROR OPENING THE OUTPUT FILE'
      WRITE(6,*) 'CHECK FILE AND TRY AGAIN'
      GO TO 999   !Exit Program

999    END   ! (* program *)
```

B.2 SUPER.FOR

SUPER.FOR is a FORTRAN program used by the revised neural network that disengages the supervised learning used in the old neural network and instead places the clusters generated by UNSUPER.FOR directly into the output subset CLUSTER.DAT.

```
       SUBROUTINE SUPERVISED(NTRAIN,PTRAIN,XTRAIN)
******************************************************************
**   THIS PROGRAM IMPLEMENTS A SUPERVISED Learning ALGORITHM
**   The Algorithm is a Clustering type
**   We are given a set of PTRAIN patterns, XTRAIN
**   We want to find a family of clusters (hyper)spherical clusters, Ck
**   so that all patterns in a cluster are within some sphere radius,Rho
**   The supervised learning calls the unsupervised learning algorithm
**   which generates a set of natural clusters.  Then the supervised
**   learning removes those clusters which are composed of homogeneous
**   elements.
**
**
**   This supervised algorithm has been enhanced to include:
**     (1) merging of clusters, (2) tightening of clusters,
**     (3) adding probabilities to clusters
**
**     Inputs:
**           NTRAIN -- Vector size of training patterns
**           PTRAIN -- Number of training patterns
**           XTRAIN --  PTRAIN Structures of Training patterns
**
**     Common:
**           Final_C -- contains the clusters generated by the routine
**           Fin_CNUM -- final number of clusters
**
*********************************************************************
      IMPLICIT NONE
```

```
        INCLUDE 'PARAMS.INC'

           INTEGER*4     NTRAIN,PTRAIN
        INTEGER*4     I,J,K,M      !Indexes for DO loops
        INTEGER*4     L,R         !Indexes
        INTEGER*4     CNUM        !Number of ActiveClusters
        INTEGER*4     FIN_CNUM    !Number of Final Clusters
        INTEGER*4     P           !Number of Patterns
        INTEGER*4     N           !Size of Pattern Vector
        REAL*8       RHO          !Sphere Radius
        REAL*8       RHOTWO        !Square of RHO
        REAL*8       NORM          !Normalized Value of Input Pattern
        REAL*8       D(FCMAX)        !Distance between each pattern & the center
        REAL*8       DECR          !Interval by which RHO will be decremented
        LOGICAL       STOP         !Parameter that stops DO loop
        INTEGER*4     FX_CTR        !Number of patterns in final set
        INTEGER*4     P_CTR        !Total Number of patterns
        LOGICAL       FIRST        !Signifies first iteration through program
        INTEGER*4     MATCH      !Counts num of patterns in cluster that match
        REAL*8       DIS       !Distance between two vectors
        REAL*8       NEW_DIS !Used to hold distance value until an update
        INTEGER*4     NEW_CLUS  !Used to hold value of cluster until an update
        REAL*8       DEPSILON  !Min distance between two cluster
        REAL*8       PEPSILON
    !Distance to be added to fartherest pattern in cluster to adjust radius
    !      PARAMETER     (PEPSILON = 0.15)
              parameter         (pepsilon = 0.20)
           LOGICAL                    BAD_SET  !Signifies patts couldn't be place in 1 cluster

        STRUCTURE  /XPATTERN/
        REAL*8     XP(NMAX)  !Normalized Input Pattern:
        INTEGER*4   XELEM    !# of Cluster pattern belongs to
        CHARACTER*25  CLASS !The class membership of the pattern
        END STRUCTURE

        STRUCTURE /CLUSTER/
        REAL*8     B(NMAX)   !Centroid Vector
        REAL*8   CRHO      !Square of Sphere Radius
        INTEGER*4 NUM      !Number of Patterns in the Cluster
        INTEGER*4 PATT_NUM(PP) !Identifies patterns that belong to Cluster
        END STRUCTURE

        STRUCTURE /TEMP_XPATTERN/
        REAL*8     XP(NMAX)  !Normalized Input Pattern: index reps vector pos'n
        CHARACTER*25  CLASS       !The class membership of the pattern
        END STRUCTURE

        STRUCTURE /FIN_CLUSTER/
        REAL*8    B(NMAX)   !Centroid Vector
        REAL*8    CRHO   !Square of Sphere Radius
        INTEGER*4  NUM     !Number of Patterns in the Cluster
        CHARACTER*25  CLASS  !The class membership of the pattern
        REAL*8    PROB   !Probability of this cluster
        END STRUCTURE
```

```
      RECORD   /XPATTERN/ X,XTRAIN(PP)
      RECORD   /CLUSTER/  C
      RECORD /XPATTERN/ FINAL_X
      RECORD /FIN_CLUSTER/ FINAL_C
      RECORD   /TEMP_XPATTERN/TEMP(PP)


      COMMON /BLK1/X(PP)
      COMMON /BLK2/C(PP)
!3/19/95      COMMON /BLK3/FIRST,RHO,RHOTWO,N,P,CNUM,BAD_SET
         common /blk3/rho,rhotwo,n,p,cnum
         common /blk4/first,bad_set
      COMMON /BLK5/FINAL_X(PP)
      COMMON /BLK6/FINAL_C(PP)
      COMMON /BLK7/FIN_CNUM

C      OPEN(UNIT=5,STATUS='OLD',ERR=100)
c     OPEN(UNIT=6,STATUS='NEW',ERR=200,)
          OPEN(UNIT=8,FILE='CLUSTER.DAT',STATUS='NEW')

305   FORMAT(I5)
310   FORMAT(F6.4)
315   FORMAT(10F7.4)
316   FORMAT(10F7.3)
350   FORMAT(A25)
351   FORMAT(2X,A25)
205   FORMAT(' VECTOR SIZE:',3X,I5,/' NUMBER OF PATTERNS:',3X,I5/)
210   FORMAT(' NUMBER OF PATTERNS IN CLUSTER:',3X,I5/)
215   FORMAT(' CLUSTER NUMBER:',3X,I5/)
220   FORMAT(10F10.6)
230   FORMAT(' CLASS MEMBERSHIP OF CLUSTER:',3X,'"',A25,'"',//)
240   FORMAT(' RHO OF CLUSTER:',3X,F8.6/)


      P_CTR = PTRAIN
      P = PTRAIN
      N = NTRAIN
      DO I = 1,P
       DO J = 1,N
        X(I).XP(J) = XTRAIN(I).XP(J)
       END DO
       X(I).CLASS = XTRAIN(I).CLASS
      END DO

!   Normalize each pattern, using ||Xi|| = 1
      DO J = 1,P
       NORM = 0
       DO K = 1,N
        NORM = NORM + (X(J).XP(K)**2)
       END DO !(K)
       NORM = SQRT(NORM)
       IF (NORM .EQ. 0.0) THEN
        NORM = .0000001
       END IF
       DO K = 1,N
```

```
        X(J).XP(K) = (X(J).XP(K)/NORM)
       END DO !(K)
      END DO ! (J)


          CNUM=0
          FIRST=.TRUE.
          BAD_SET=.FALSE.
          CALL UNSUPERVISED

          FIN_CNUM=0
          FX_CTR=0
          FIRST=.FALSE.
!         RHO=1

          CALL UNSUPERVISED

          WRITE(8,*) CNUM
          WRITE(8,*) RHO
          DO J=1,CNUM
                  DO M=1,N
                          final_c(j).b(m)=c(j).b(m)
                          WRITE(8,*) FINAL_C(J).B(M)
                  end do
          end do


      GO TO 999 !End of Program
*****************************************************************
*****     ERROR PROCESSING          **********
*****                    **********
*****************************************************************
C100   WRITE(6,*) 'ERROR OPENING THE INPUT FILE'
C     WRITE(6,*) 'CHECK FILE AND TRY AGAIN'
C     GO TO 999  !Exit Program

C200   WRITE(6,*) 'ERROR OPENING THE OUTPUT FILE'
C     WRITE(6,*) 'CHECK FILE AND TRY AGAIN'
C     GO TO 999   !Exit Program

c1000  WRITE(6,*) 'COULD NOT GENERATE CLUSTERS FOR PATTERNS'
       GO TO 999

999   END   ! (* program *)
```

Appendix C: Revised Neural Network MATLAB Programs

C.1 FUZZYMF.M

FUZZYMF.M is a MATLAB M-file that generates a fuzzy membership function matrix using the algorithm described by fuzzy c-means clustering.

```
load datax;
load datay;
data = [datax datay];
data_n = size(data,1);

load clusterx;
load clustery;
cluster = [clusterx clustery];
cluster_n = size(cluster,1);

expo = 2;
max_iter = 100;
min_impro = 1e-5;

obj_fcn = zeros(max_iter,1);

U = rand(cluster_n,data_n);
col_sum = sum(U);
U = U ./ col_sum(ones(cluster_n,1),:);

for i = 1:max_iter,
  mf = U .^ expo;
  dist = zeros(cluster_n,data_n);
  for k = 1:cluster_n,
        dist(k,:) = sqrt(sum(((data-ones(data_n,1)*cluster(k,:)).^2)'));
  end
  obj_fcn(i) = sum(sum((dist.^2) .* mf));
  tmp = dist .^ (-2/(expo-1));
  U = tmp ./ (ones(cluster_n,1)*sum(tmp));
  if i>1
        if abs(obj_fcn(i)-obj_fcn(i-1))<min_impro,break; end,
  end
end
iter_num = i;
obj_fcn(iter_num + 1:max_iter) = [];
```

C.2 MFPLOT.M

MFPLOT.M is a MATLAB M-file that generates a plot which shows the fuzzy membership function of a pattern in relation to it's position in space.

```
for num = 1:cluster_n
        figure;
        [x,y] = meshgrid(0:0.1:1,0:0.1:1);
        tmp = U';
        z = griddata(data(:,1),data(:,2),tmp(:,num),x,y);
        mesh(x,y,z);
        title=['MF Plot for Cluster ',int2str(num)];
        xlabel('X'),ylabel('Y'),zlabel('MF');
end
```