TOPICS IN PATTERN RECOGNITION USING UNSUPERVISED LEARNING

BY

PETER J. CZEZOWSKI

A Thesis
Submitted to the Faculty of Graduate Studies
in Partial Fulfillment of the Requirements
for the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba

ISBN 0-315-99039-2

Name PETER JOHN CZEZOWSKI

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

ENGINEERING - ELECTRONICS & ELECTRICAL — SUBJECT TERM

0544 — SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS
Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION
General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523
Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS
Language
- General 0679
- Ancient 0289
- Linguistics 0290
- Modern 0291

Literature
- General 0401
- Classical 0294
- Comparative 0295
- Medieval 0297
- Modern 0298
- African 0316
- American 0591
- Asian 0305
- Canadian (English) 0352
- Canadian (French) 0355
- English 0593
- Germanic 0311
- Latin American 0312
- Middle Eastern 0315
- Romance 0313
- Slavic and East European 0314

PHILOSOPHY, RELIGION AND THEOLOGY
Philosophy 0422
Religion
- General 0318
- Biblical Studies 0321
- Clergy 0319
- History of 0320
- Philosophy of 0322

Theology 0469

SOCIAL SCIENCES
American Studies 0323
Anthropology
- Archaeology 0324
- Cultural 0326
- Physical 0327

Business Administration
- General 0310
- Accounting 0272
- Banking 0770
- Management 0454
- Marketing 0338

Canadian Studies 0385
Economics
- General 0501
- Agricultural 0503
- Commerce-Business 0505
- Finance 0508
- History 0509
- Labor 0510
- Theory 0511

Folklore 0358
Geography 0366
Gerontology 0351
History
- General 0578
- Ancient 0579
- Medieval 0581
- Modern 0582
- Black 0328
- African 0331
- Asia, Australia and Oceania 0332
- Canadian 0334
- European 0335
- Latin American 0336
- Middle Eastern 0333
- United States 0337

History of Science 0585
Law 0398
Political Science
- General 0615
- International Law and Relations 0616
- Public Administration 0617

Recreation 0814
Social Work 0452
Sociology
- General 0626
- Criminology and Penology 0627
- Demography 0938
- Ethnic and Racial Studies 0631
- Individual and Family Studies 0628
- Industrial and Labor Relations 0629
- Public and Social Welfare 0630
- Social Structure and Development 0700
- Theory and Methods 0344

Transportation 0709
Urban and Regional Planning 0999
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES
Agriculture
- General 0473
- Agronomy 0285
- Animal Culture and Nutrition 0475
- Animal Pathology 0476
- Food Science and Technology 0359
- Forestry and Wildlife 0478
- Plant Culture 0479
- Plant Pathology 0480
- Plant Physiology 0817
- Range Management 0777
- Wood Technology 0746

Biology
- General 0306
- Anatomy 0287
- Biostatistics 0308
- Botany 0309
- Cell 0379
- Ecology 0329
- Entomology 0353
- Genetics 0369
- Limnology 0793
- Microbiology 0410
- Molecular 0307
- Neuroscience 0317
- Oceanography 0416
- Physiology 0433
- Radiation 0821
- Veterinary Science 0778
- Zoology 0472

Biophysics
- General 0786
- Medical 0760

EARTH SCIENCES
Biogeochemistry 0425
Geochemistry 0996
Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES
Environmental Sciences 0768
Health Sciences
- General 0566
- Audiology 0300
- Chemotherapy 0992
- Dentistry 0567
- Education 0350
- Hospital Management 0769
- Human Development 0758
- Immunology 0982
- Medicine and Surgery 0564
- Mental Health 0347
- Nursing 0569
- Nutrition 0570
- Obstetrics and Gynecology 0380
- Occupational Health and Therapy 0354
- Ophthalmology 0381
- Pathology 0571
- Pharmacology 0419
- Pharmacy 0572
- Physical Therapy 0382
- Public Health 0573
- Radiology 0574
- Recreation 0575
- Speech Pathology 0460
- Toxicology 0383

Home Economics 0386

PHYSICAL SCIENCES
Pure Sciences
Chemistry
- General 0485
- Agricultural 0749
- Analytical 0486
- Biochemistry 0487
- Inorganic 0488
- Nuclear 0738
- Organic 0490
- Pharmaceutical 0491
- Physical 0494
- Polymer 0495
- Radiation 0754

Mathematics 0405
Physics
- General 0605
- Acoustics 0986
- Astronomy and Astrophysics 0606
- Atmospheric Science 0608
- Atomic 0748
- Electronics and Electricity 0607
- Elementary Particles and High Energy 0798
- Fluid and Plasma 0759
- Molecular 0609
- Nuclear 0610
- Optics 0752
- Radiation 0756
- Solid State 0611

Statistics 0463

Applied Sciences
Applied Mechanics 0346
Computer Science 0984
Engineering
- General 0537
- Aerospace 0538
- Agricultural 0539
- Automotive 0540
- Biomedical 0541
- Chemical 0542
- Civil 0543
- Electronics and Electrical 0544
- Heat and Thermodynamics 0348
- Hydraulic 0545
- Industrial 0546
- Marine 0547
- Materials Science 0794
- Mechanical 0548
- Metallurgy 0743
- Mining 0551
- Nuclear 0552
- Packaging 0549
- Petroleum 0765
- Sanitary and Municipal 0554
- System Science 0790

Geotechnology 0428
Operations Research 0796
Plastics Technology 0795
Textile Technology 0994

PSYCHOLOGY
General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

Nom _____

Dissertation Abstracts International est organisé en catégories de sujets. Veuillez s.v.p. choisir le sujet qui décrit le mieux votre thèse et inscrivez le code numérique approprié dans l'espace réservé ci-dessous.

_____ ☐☐☐ U·M·I

SUJET — CODE DE SUJET

Catégories par sujets

HUMANITÉS ET SCIENCES SOCIALES

COMMUNICATIONS ET LES ARTS
Architecture 0729
Beaux-arts 0357
Bibliothéconomie 0399
Cinéma 0900
Communication verbale 0459
Communications 0708
Danse 0378
Histoire de l'art 0377
Journalisme 0391
Musique 0413
Sciences de l'information 0723
Théâtre 0465

ÉDUCATION
Généralités 515
Administration 0514
Art 0273
Collèges communautaires 0275
Commerce 0688
Économie domestique 0278
Éducation permanente 0516
Éducation préscolaire 0518
Éducation sanitaire 0680
Enseignement agricole 0517
Enseignement bilingue et multiculturel 0282
Enseignement industriel 0521
Enseignement primaire. 0524
Enseignement professionnel 0747
Enseignement religieux 0527
Enseignement secondaire 0533
Enseignement spécial 0529
Enseignement supérieur 0745
Évaluation 0288
Finances 0277
Formation des enseignants 0530
Histoire de l'éducation 0520
Langues et littérature 0279
Lecture 0535
Mathématiques 0280
Musique 0522
Orientation et consultation 0519
Philosophie de l'éducation 0998
Physique 0523
Programmes d'études et enseignement 0727
Psychologie 0525
Sciences 0714
Sciences sociales 0534
Sociologie de l'éducation 0340
Technologie 0710

LANGUE, LITTÉRATURE ET LINGUISTIQUE
Langues
- Généralités 0679
- Anciennes 0289
- Linguistique 0290
- Modernes 0291

Littérature
- Généralités 0401
- Anciennes 0294
- Comparée 0295
- Médiévale 0297
- Moderne 0298
- Africaine 0316
- Américaine 0591
- Anglaise 0593
- Asiatique 0305
- Canadienne (Anglaise) 0352
- Canadienne (Française) 0355
- Germanique 0311
- Latino-américaine 0312
- Moyen-orientale 0315
- Romane 0313
- Slave et est-européenne 0314

PHILOSOPHIE, RELIGION ET THÉOLOGIE
Philosophie 0422
Religion
- Généralités 0318
- Clergé 0319
- Études bibliques 0321
- Histoire des religions 0320
- Philosophie de la religion 0322

Théologie 0469

SCIENCES SOCIALES
Anthropologie
- Archéologie 0324
- Culturelle 0326
- Physique 0327

Droit 0398
Économie
- Généralités 0501
- Commerce-Affaires 0505
- Économie agricole 0503
- Économie du travail 0510
- Finances 0508
- Histoire 0509
- Théorie 0511

Études américaines 0323
Études canadiennes 0385
Études féministes 0453
Folklore 0358
Géographie 0366
Gérontologie 0351
Gestion des affaires
- Généralités 0310
- Administration 0454
- Banques 0770
- Comptabilité 0272
- Marketing 0338

Histoire
- Histoire générale 0578
- Ancienne 0579
- Médiévale 0581
- Moderne 0582
- Histoire des noirs 0328
- Africaine 0331
- Canadienne 0334
- États-Unis 0337
- Européenne 0335
- Moyen-orientale 0333
- Latino-américaine 0336
- Asie, Australie et Océanie 0332

Histoire des sciences 0585
Loisirs 0814
Planification urbaine et régionale 0999
Science politique
- Généralités 0615
- Administration publique 0617
- Droit et relations internationales 0616

Sociologie
- Généralités 0626
- Aide et bien-être social 0630
- Criminologie et établissements pénitentiaires 0627
- Démographie 0938
- Études de l' individu et de la famille 0628
- Études des relations interethniques et des relations raciales 0631
- Structure et développement social 0700
- Théorie et méthodes. 0344
- Travail et relations industrielles 0629

Transports 0709
Travail social 0452

SCIENCES ET INGÉNIERIE

SCIENCES BIOLOGIQUES
Agriculture
- Généralités 0473
- Agronomie. 0285
- Alimentation et technologie alimentaire 0359
- Culture 0479
- Élevage et alimentation 0475
- Exploitation des pâturages 0777
- Pathologie animale 0476
- Pathologie végétale 0480
- Physiologie végétale 0817
- Sylviculture et faune 0478
- Technologie du bois 0746

Biologie
- Généralités 0306
- Anatomie 0287
- Biologie (Statistiques) 0308
- Biologie moléculaire 0307
- Botanique 0309
- Cellule 0379
- Écologie 0329
- Entomologie 0353
- Génétique 0369
- Limnologie 0793
- Microbiologie 0410
- Neurologie 0317
- Océanographie 0416
- Physiologie 0433
- Radiation 0821
- Science vétérinaire 0778
- Zoologie 0472

Biophysique
- Généralités 0786
- Medicale 0760

SCIENCES DE LA TERRE
Biogéochimie 0425
Géochimie 0996
Géodésie 0370
Géographie physique 0368
Géologie 0372
Géophysique 0373
Hydrologie 0388
Minéralogie 0411
Océanographie physique 0415
Paléobotanique 0345
Paléoécologie 0426
Paléontologie 0418
Paléozoologie 0985
Palynologie 0427

SCIENCES DE LA SANTÉ ET DE L'ENVIRONNEMENT
Économie domestique 0386
Sciences de l'environnement 0768
Sciences de la santé
- Généralités 0566
- Administration des hôpitaux 0769
- Alimentation et nutrition 0570
- Audiologie 0300
- Chimiothérapie 0992
- Dentisterie 0567
- Développement humain 0758
- Enseignement 0350
- Immunologie 0982
- Loisirs 0575
- Médecine du travail et thérapie 0354
- Médecine et chirurgie 0564
- Obstétrique et gynécologie 0380
- Ophtalmologie 0381
- Orthophonie 0460
- Pathologie 0571
- Pharmacie 0572
- Pharmacologie 0419
- Physiothérapie 0382
- Radiologie 0574
- Santé mentale 0347
- Santé publique 0573
- Soins infirmiers 0569
- Toxicologie 0383

SCIENCES PHYSIQUES
Sciences Pures
Chimie
- Généralités 0485
- Biochimie 487
- Chimie agricole 0749
- Chimie analytique 0486
- Chimie minérale 0488
- Chimie nucléaire 0738
- Chimie organique 0490
- Chimie pharmaceutique 0491
- Physique 0494
- PolymÈres 0495
- Radiation 0754

Mathématiques 0405
Physique
- Généralités 0605
- Acoustique 0986
- Astronomie et astrophysique 0606
- Électronique et électricité 0607
- Fluides et plasma 0759
- Météorologie 0608
- Optique 0752
- Particules (Physique nucléaire) 0798
- Physique atomique 0748
- Physique de l'état solide 0611
- Physique moléculaire 0609
- Physique nucléaire 0610
- Radiation 0756

Statistiques 0463

Sciences Appliqués Et Technologie
Informatique 0984
Ingénierie
- Généralités 0537
- Agricole 0539
- Automobile 0540
- Biomédicale 0541
- Chaleur et thermodynamique 0348
- Conditionnement (Emballage) 0549
- Génie aérospatial 0538
- Génie chimique 0542
- Génie civil 0543
- Génie électronique et électrique 0544
- Génie industriel 0546
- Génie mécanique 0548
- Génie nucléaire 0552
- Ingénierie des systèmes 0790
- Mécanique navale 0547
- Métallurgie 0743
- Science des matériaux 0794
- Technique du pétrole 0765
- Technique minière 0551
- Techniques sanitaires et municipales 0554
- Technologie hydraulique 0545

Mécanique appliquée 0346
Géotechnologie 0428
Matières plastiques (Technologie) 0795
Recherche opérationnelle 0796
Textiles et tissus (Technologie) 0794

PSYCHOLOGIE
Généralités 0621
Personnalité 0625
Psychobiologie 0349
Psychologie clinique 0622
Psychologie du comportement 0384
Psychologie du développement 0620
Psychologie expérimentale 0623
Psychologie industrielle 0624
Psychologie physiologique 0989
Psychologie sociale 0451
Psychométrie 0632

TOPICS IN PATTERN RECOGNITION USING UNSUPERVISED LEARNING

BY

PETER J. CZEZOWSKI

A Thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Abstract

This thesis investigates the use of unsupervised learning algorithms in applications of pattern recognition and data representation. Various classical and artificial neural network algorithms were included in the investigation. Particular attention was devoted to the topics of valid clustering and input component prediction applications. Through the examination of experiments on the algorithms considered, it was determined that the *growing cell structure* was the most promising. With respect to valid clustering, an improvement is suggested for the *self-organizing map* interpretation problem, and a new process for interpreting the growing cell structure is suggested. A new technique in the application of these algorithms to problems in the prediction of missing variables is evaluated. It was concluded that these algorithms are effectively able to form compressed representations of their original data sets which are useful in classification and prediction applications.

Acknowledgements

Upon reflection on the time and energy that went into this degree, I may identify many people who have helped, advised and influenced me. Greatest thanks to my advisor, Dr. Witold Pedrycz, whom I've been with since my undergrad days. Through symbolic computing, fuzzy sets and neural networks, he was the first to introduce me to world of artificial intelligence and was very patient as I went on my 'academic digressions' in an attempt to learn a little about everything. So far, my time at this department has been spent in a wonderfully synergistic atmosphere created by all the terrific people in the VLSI systems research group. All the professors and students in this group made me feel welcome and appreciated. Their comradeship and cooperation are greatly appreciated. Finally, thanks to MICRONET for the financial support without which this work would not have be undertaken.

List of Figures

Contents

Chapter 1

Introduction

1.1 Preamble

Simply stated, the phrase *'pattern recognition'* refers to a system's ability to categorize objects into one of two or more classes. Of course, the ability to recognize or classify objects comes naturally to us humans. However, the general area of pattern recognition does not pertain solely to the classification of objects in two dimensional planes or three dimensional real world settings. There are countless applications ranging from the processing of documents (character recognition) to medical data (interpreting x-rays or electrocardiograms) which require the categorization of higher dimensional data. In the latter case, the classifiers used may be nebulous with respect to their operation and methods of construction. That is, it is not always obvious which criteria were used to make the classification decision.

Pattern recognition tasks have always been regarded as being important and requiring optimal solutions. Therefore, the area has historically received a lot of research attention, and a variety of techniques used to solve the problem have been developed. Typically, the techniques used to create the classifiers are computationally intensive, each involving numerous iterations over the data set in order to achieve a satisfactory rate of correct classifications.

Classifiers are constructed based on the attributes of their training and testing data sets. The patterns to be recognized, i.e. classified, are stored in a vector representation format and may be either labeled or unlabeled. This is where the first distinction in classifiers

becomes evident: *supervised* versus *unsupervised learning* (*training*). Those classifiers derived using labeled data are said to have performed supervised learning. This is because they have made use of the a priori knowledge of the data's classes (said to have been provided by a teacher.) Subsequently, if unlabeled data has been used to create the classifier, it is said to have performed unsupervised learning. Classically, pattern recognition in an unlabeled data environment is often referred to as *cluster analysis.*

The statistical approach is considered to be the traditional method for performing cluster analysis. However, if as is usually the case, full statistical information about the data is not known, a deterministic-nonstatistical approach is required. Recently, classifiers based on neural networks and others making use of fuzzy sets have been receiving much research attention [35,27,4]. Finally, classifiers are often used as components of systems for automated process control or the prediction and/or compression of data.

1.2 Purpose

The purpose of this thesis is to investigate unsupervised learning pattern recognition algorithms through the comparison and contrast of several systems which may be used for prediction and data compression tasks. During the course of the research, five algorithms were investigated for their properties with respect to application to the above mentioned problems. Specifically, the algorithms investigated include Kohonen's self-organizing maps (SOM) [30-32], Fritzke's growing cell structures (GCS) [14], c-means algorithms (CM) [3,4], Bezdek's fuzzy Kohonen clustering networks (FKCN) [5] and Gath and Geva's unsupervised fuzzy partition - optimal number of clusters (UFP-ONC) [17]. The nodes, or cells, in all of these algorithms essentially perform the same calculations. Each node is able to compute its similarity, in the form of a distance measure, to the inputs from the data set. The differences between the algorithms stem from their unique approaches to the modification of the positions of the nodes in order to improve the representation of the

data with each iteration. The research in this area is necessary to determine if any of the algorithms have inherent advantages over the others with respect to specific application categories.

1.3 Scope

As indicated above, the scope of this thesis shall be limited to algorithms employing unsupervised learning. However, an introduction to supervised learning algorithms shall be included in Chapter 2 for the purpose of completeness. Extra emphasis will be given to reporting a new way to use the classifiers for prediction as developed during the course of the research.

The remainder of this report is structured as follows:

- Chapter 2 - Overview of pattern recognition including both supervised and unsupervised learning algorithms.
- Chapter 3 - Presentation of the algorithms implemented for the thesis.
- Chapter 4 - Discussion of issues regarding valid clustering.
- Chapter 5 - Presentation of simulation results for applications in the prediction of the values of missing input components.
- Chapter 6 - Conclusions and recommendations for future work.

Chapter 2

Pattern Recognition Overview

The goals of this chapter of the report are the following:

- Introduce pattern recognition systems.
- Describe various supervised and unsupervised learning algorithms and their differences.
- Explain the limitations of unsupervised learning.

Since this chapter of the report is intended to provide the reader with an introduction to pattern recognition, the descriptions provided are merely cursory and many reference sources are cited for further reading.

2.1 Composition of a Pattern Recognition System

Figure 2.1 depicts the processes necessary for a pattern recognition system. From the figure, it is evident that there are several steps involved in the classification process, i.e.

- data acquisition,
- preprocessing, and
- actual classification.

This thesis is not concerned with the data acquisition process itself. The methods used for sensing and measuring the features of the input data are independent of classification techniques. The data is normally divided into two sets. The first (normally larger) set is to be used for the *training*, i.e. constructing, the classifier. The second set is retained for the purposes of *testing* the classifier to determine its classification error rate after it has been trained. Both of the data sets must be statistically relevant and have well chosen features with respect to the classes within the problem which they represent. This criterion is

necessary in order to effect the greatest potential generalization capabilities (robustness) of the classifier. If the training and testing data sets are not indicative of the general classification problem, the classifier's overall performance will suffer. Jain [28] suggests that a good training set should contain five to ten examples per input dimension for each class to be identified within the data. We are also reminded of the "curse of dimensionality" - input features which are not necessary in order to achieve classification should be avoided for they may hinder the process and lead to slow and overly complex classifiers.

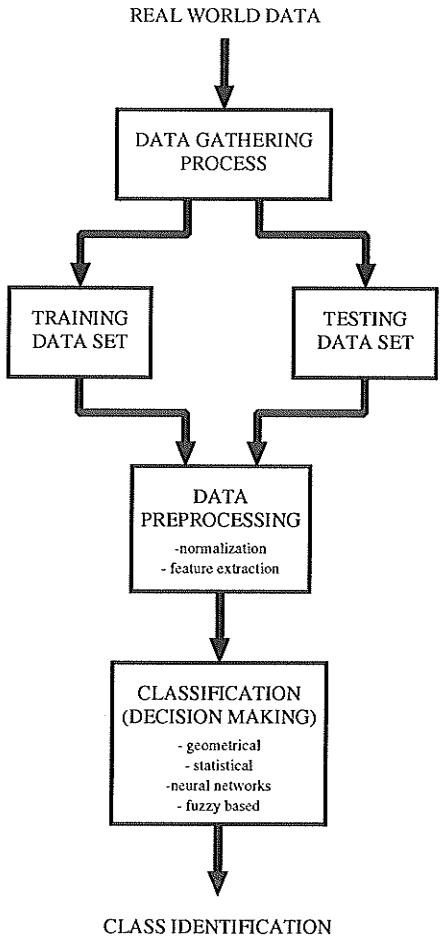


FIGURE 2.1. A General Pattern Recognition System.

After the data has been collected, any necessary preprocessing may be performed. This step may involve normalization and/or feature extraction (data reduction) processes used to make the data sets more manageable. The final step in the pattern recognition system is the actual classification itself. Here, the feature vector is passed to the trained classifier resulting in an output of the vector's class.

As mentioned in the previous chapter, the first distinct way in which the classifiers themselves are categorized is based upon whether or not the data used to train them is labeled. The supervised and unsupervised learning categories may also be subdivided according to the approaches taken to solving the classification problem. Figure 2.2 below presents a table of the taxonomy of classifiers.

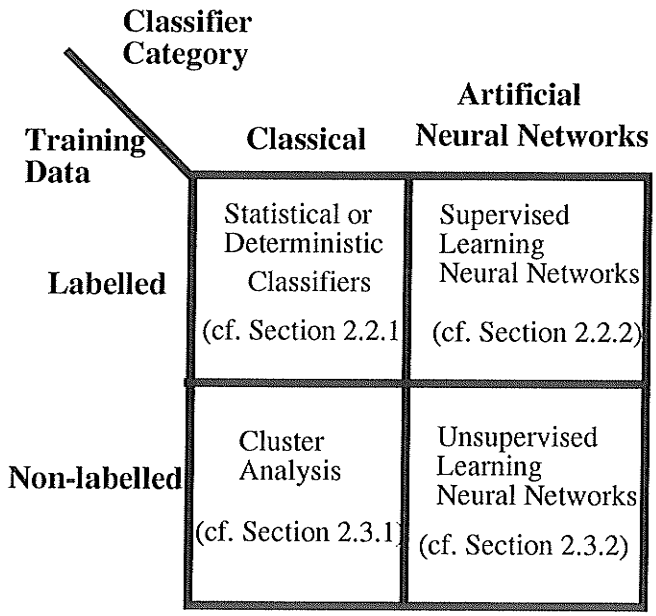


FIGURE 2.2. A Taxonomy of Pattern Classifiers.

In this thesis, classifiers are separated into two categories: classical and neural network based. Since we are concerned with unsupervised pattern recognition, the types of classifiers in the top two quadrants of the table were not included in the focus of this thesis. However, for completeness, they are briefly described in the report. For the

purposes of this report, the category of classical classifiers includes all techniques that are not neural network based.

Figure 2.3 depicts the three ways that a pattern recognizer may determine the classes within a data set. The illustrations are for a two dimensional example and indicate the intuitive methods of dividing the data points into classes. In the first case, shown in Figure 2.3(a), the classes are defined by means of decision boundaries between them. These boundaries are hyperplanes within the input space, and represent characteristics of the classes in the problem. Such a classification scheme may be employed only when the training set is labeled. Here, the training process consists of adapting the decision boundaries, i.e. learning the characteristics of the classes, until an acceptable rate of misclassification is achieved. A candidate data point is assigned full membership to a particular class based upon its position relative to the boundary lines.

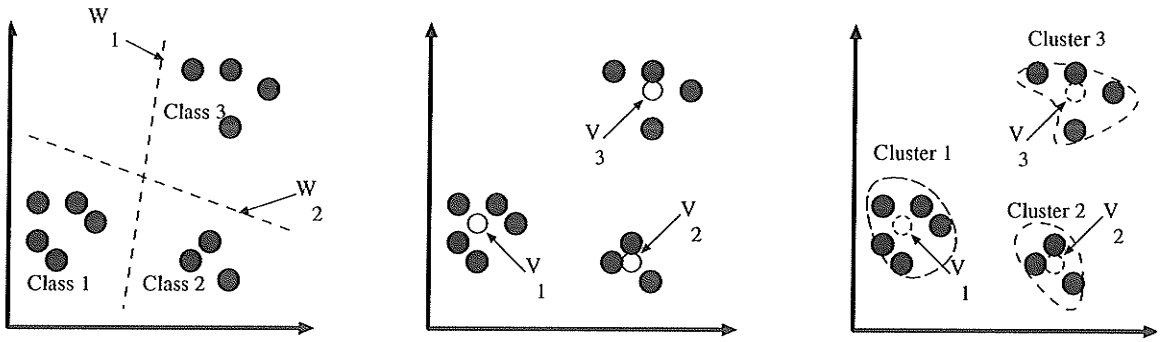


FIGURE 2.3. Classification techniques.
(a) Decision Boundaries, (b) Vector Quantization (cluster prototypes) and (c) Elastic Clustering.

The second classification technique, which is shown in Figure 2.3(b), is commonly referred to as *vector quantization*. With this approach, prototypes (centroids) for clusters within the data set are developed during the training process. For reasons explained in Section 2.2.1, vector quantization techniques for pattern recognition purposes are used

only with unlabeled data sets. Each new point to be classified is assigned a grade of membership to the various clusters based upon its relative distance to them all.

The final classification technique, depicted in Figure 2.3(c), is a hybrid of the first two. It may be used only with labeled training sets. This approach makes use of *'elastic rings'* to define the boundaries of the classes within the data set. The points to be classified are then assigned full membership to a particular class if they lie within the ring for that class, otherwise, a grade of membership with respect to relative distance to the closest points on the rings of all the classes is assigned.

In view of the above, it is evident that when constructing a classifier, having labeled data is a luxury that provides several implementation options. Whereas supervised learning classifiers may take their form from any of the three paradigms presented in Figure 2.3, all unsupervised learning algorithms are essentially consigned to performing vector quantization. The following two sections of the report present some of the supervised and unsupervised techniques currently used in pattern recognition.

2.2 Supervised Learning Algorithms

As mentioned in the previous section, the phrase *'supervised learning'* refers to the classifier's training algorithm making use of the previously labeled data to construct the classifier. In Figure 2.2, "A Taxonomy of Pattern Classifiers." on page 6, it may be seen that the classifiers in the top two quadrants belong in the supervised learning category. We shall first present the classical supervised pattern recognition techniques, followed by a selection of available neural network approaches.

The *error rate* of a classifier is estimated by experimentally determining the fraction of misclassifications of the test data set. This estimate is improved by increasing the number of samples in the test set.

2.2.1 Classical Supervised Learning Classifiers

All of the classifiers presented in this section are *one-step* or *single stage* classifiers. If the classification problem is too complicated, i.e. too large, to be handled in one-step, hierarchical classifiers may be employed. No loss of generality is incurred by describing only the one-step classifiers.

2.2.1.1 Decision Boundary Type Classifiers

In typical classification problems, complete statistical information is usually not known a priori. In such a case, an estimate of the statistical nature of the problem space must be derived from the training samples. The accuracy of the estimate depends on how well the data set represents the statistics of the classes within the problem. Such a situation gives rise to two types of classical supervised classifiers. Firstly, *statistical* pattern classifiers, which make use of the known distributions for the categories within the data and well known concepts from statistical decision theory to determine the class boundaries. Secondly, *deterministic* pattern classifiers, which are constructed by means of estimates of the distributions for the groups within the data. These distribution estimates are derived through a statistical analysis of the classes within the labeled training data set. Since both of these kinds of classifiers are used to define hard decision boundaries between the classes within the data, they are of the type shown in Figure 2.3a. A hard decision means that a sample is assigned either full membership in a class or none at all. This decision is based upon the sample's position relative to the class boundaries. In such classification schemes, there are no grades of memberships assigned to the samples.

In the case of complete knowledge of the distributions within the data, or a hypothetically infinite training data set, the Bayes decision rule produces the optimal classifier. Let a feature vector (data point) have a probability density function conditional on the pattern

class given as $p(x|\omega_i)$. The Bayes decision rule [15,16] is then described by Equation 2.1: pattern x is assigned to class ω_i if

$$p(\omega_i|x) \geq p(\omega_j|x) \quad \forall j \neq i. \tag{2.1}$$

where $p(\omega_i|x)$ is the posterior density for class ω_i defined by Equation 2.2

$$p(\omega_i|x) = \frac{p(x|\omega_i)p(\omega_i)}{\sum_{i=1}^{c} p(x|\omega_i)p(\omega_i)} \tag{2.2}$$

where $p(\omega_i)$ are the a priori class densities.

The Bayes decision rule is optimal. That is, for a given prior distribution, there does not exist a decision rule having a smaller probability of misclassification. Such a classification technique is deemed to be parametric because the class densities are known or may be well estimated. The distributions are commonly assumed to be multivariate gaussian in nature. Such an assumption leads to simple decision boundaries which are linear or quadratic. Multiclass problems result in piecewise-linear or piecewise-quadratic decision boundaries. If a parametric form for the classifier cannot be assumed due to for instance, multimodal classes within the data, a non-parametric or geometrical approach must be taken. The largest family of deterministic classifiers are the *Nearest Neighbor* classifiers described by Cover and Hart [8].

The Nearest Neighbor, or k-Nearest Neighbor (k-NN), classifiers are intuitively appealing in that, the process of classifying an unknown data sample gives the most weight to the evidence of nearby known samples. The number of nearest neighbors taken into account, k, may be greater than or equal to 1. Essentially, the k-NN classification rule assigns the feature vector x to the class which is most heavily represented within the group of k nearest training vectors.

The complexity of such classification schemes results from the need to compute a distance measure from the test vector to be classified and each vector in the training set while keeping track of the classes of the k closest neighbors. In order to reduce the complexity of this family of classifiers, editing and condensing procedures may be invoked on the training data set. The editing process consists of the removal of strangers (outliers) from the training set. The removal of such vectors improves the error rate of low k-NN classifiers which are particularly susceptible to making classification errors in such instances. The training data condensing process involves the removal from the data set of all vectors not immediately close to a decision boundary. The use of such a process to remove the non-essential training vectors may result in a great reduction in the size of the data set.

The nonparametric approaches perform better in real-world environments where the data set does not contain very many samples. This superior performance is due to the extra specific information about the class boundaries which are determined by the geometrical approaches. However, the performance boost comes at a cost of the long training times required by these techniques.

2.2.1.2 Supervised Vector Quantizers

We now examine the possibility of creating supervised classifiers of the type shown in Figure 2.3b. Vector quantizers used for pattern recognition are appealing for two reasons. First, each class is represented by a single prototype, usually the centroid. Secondly, vector quantizers allow for the possibility of the assignment of grades of membership in those cases where a good classification decision is not very likely. An example of such a case occurs if the sample lies directly on the midpoint between its two closest centroids. Here we would intuitively like to assign the sample an equal grade of membership to each cluster.

Since the decision spaces in vector quantization are defined in terms of neighborhoods around the cluster prototypes, they are very regular in their shape. The clusters found are typically hyper-spherical or hyper-elliptical in nature, depending on the type of metric employed. A Euclidean metric would result in hyper-spherical clusters, while a matrix norm induced metric would result in hyper-elliptical clusters. For a further discussion of the influence of the choice of metrics, the reader is referred to Section 2.4 of the present report and Bezdek [4].

Although easily constructed, supervised vector quantization techniques are not generally used. Since the classes within real world data sets need not be hyper-spherical or even hyper-elliptical in nature, the k-NN type classifiers would result in better decision boundaries. For this reason, supervised classifiers of this type are given no further consideration in the present report. However, such techniques are paramount to the unsupervised learning techniques described in Section 2.3.

2.2.1.3 Elastic Clustering Classifiers

The third technique for creating a supervised pattern classifier, as shown in Figure 2.3(c), is known as *Elastic Clustering*. The classes within the data set are identified by enclosing each in a closed loop. Such a procedure however is non-trivial. The method proposed by Srikanth et al [45] generates closed loops which result in groups of patterns of the same data class. Then the vectors in the test set are assigned fuzzy membership values with respect to each cluster.

The loops are considered to be elastic rings which are initially placed in the problem space. The rings are represented by equidistant points on there circumferences. The rings are slowly expanded to enclose all the training patterns belonging to the same class. This ring expansion process is analogous to fitting elastic bands around pegs placed in space. Each ring is defined to have a corresponding energy E_e, defined by Equation 2.3. Training

the classifier consists of optimizing the placement of the rings by minimizing the ring energies through a gradient descent procedure.

$$E_e = -\alpha \frac{1}{K} \sum_i ln \sum_j \Phi (|X_i - Y_j|, K) + \beta \sum_j |Y_{j+1} - Y_j| \tag{2.3}$$

where $\Phi (d, K)$ is defined as $e^{-d^2/2K^2}$, X_i is the position vector of the i^{th} pattern in the training set and Y_j is the position vector of a point on the elastic band (ring).

Each ring energy function is formed so that a change in point Y_j, given by Equation 2.4

$$\Delta Y_j = -K \frac{\partial E_e}{\partial Y_j} = \alpha \sum_i W_{ij} (X_i - Y_j) + \beta K (Y_{j+1} - 2Y_j + Y_{j-1}) \tag{2.4}$$

where $W_{ij} = \Phi (|X_i - Y_j|, K) / (\sum_k \Phi (|X_i - Y_k|, K))$, and α, β and K are constants, results in the reduction of the ring's energy.

The two parts of the energy function ensure that once the function is minimized, the band passes close to the patterns which are nearest to the band and the points on the band are pulled close to their immediate neighbors.

Once the training process has been completed, sets for the classes are immediately evident. Any test vector laying within the corresponding ring has full membership to that class. Those test vectors which do not lie within the areas enclosed by the rings are assigned graded (or fuzzy) membership to each class. Equation 2.5 describes the membership of a test pattern x to the particular cluster having its centroid at point C:

$$membership (x) = \begin{cases} 1 & if\ D \le R \\ e^{-((D-R)^2/(\sigma \cdot R))} & otherwise \end{cases} \tag{2.5}$$

where σ is a parameter which determines the steepness of the membership function, D is the distance between the cluster center C and the test pattern x and R is the distance between P (the closest point on the ring with respect to the test pattern x) and the cluster center. Figure 2.4 illustrates typical results of the graded membership assignment procedure for the Fuzzy Elastic Clustering process.

Clearly, such a classification scheme is superior to a conventional vector quantization approach due to its ability to identify the irregularly shaped boundaries of classes in real world problems. It was mentioned above in the report that by design, the k-NN algorithms perform better than the vector quantizers in such cases. Other researchers report classification results from the fuzzy elastic clustering method to be comparable to those obtained by the Fuzzy k-NN algorithm [45].

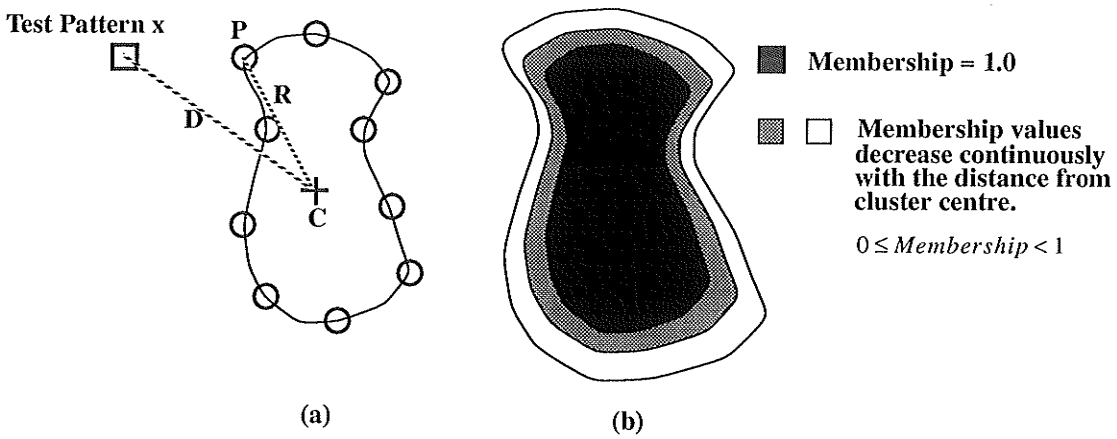


FIGURE 2.4. Fuzzy Elastic Clustering.
(a)Determining a sample's membership value in Fuzzy Elastic Clustering, (b) Membership values. Source: Adapted from [45].

In summary, this section has introduced the three main methods for which supervised pattern recognition has been performed in the classical approach. An overview of the

Bayesian, k-NN and Elastic Clustering algorithms was presented with a short discussion on the main points of each technique.

2.2.2 Supervised Learning Neural Networks

It has been said that mankind has been very successful in creating computers to effectively perform tasks which we could not easily do ourselves. With that accomplished, designers directed their efforts towards creating computers for the performance of the simple tasks which we ourselves managed with ease. Not only did the researchers achieve only very limited success, but they also realized how little is understood about the manner in which these tasks were performed in the first place. The human brain or biological neural networks in general are huge massively parallel pattern recognition systems. A large portion of pattern recognition research in recent years has been focussed on artificial neural networks (ANN) [35].

ANNs are structured as small scale conceptualizations of their biological counterparts. Neural networks are very well suited for pattern recognition problems because they perform vector-to-vector mappings. Due to their aptitude towards pattern recognition, ANNs are also suitable for variable prediction problems and controller applications.

ANNs consist of highly interconnected simple computational units referred to as neurons. The weighted interconnects between the neurons represent the synapses of the biological neural nets. Figure 2.5 shows a model for the basic structure of the neurons used in such systems. Each neuron unit is described as simply producing an output value, y, described by the Equation 2.6:

$$y = g(\sum_{i=1}^{N} x_i w_i - \theta) \tag{2.6}$$

where x_i represent the values of the inputs to the neuron, w_i represent the connection strengths (weights) of the separate inputs, and θ represents the threshold or bias value of the unit.

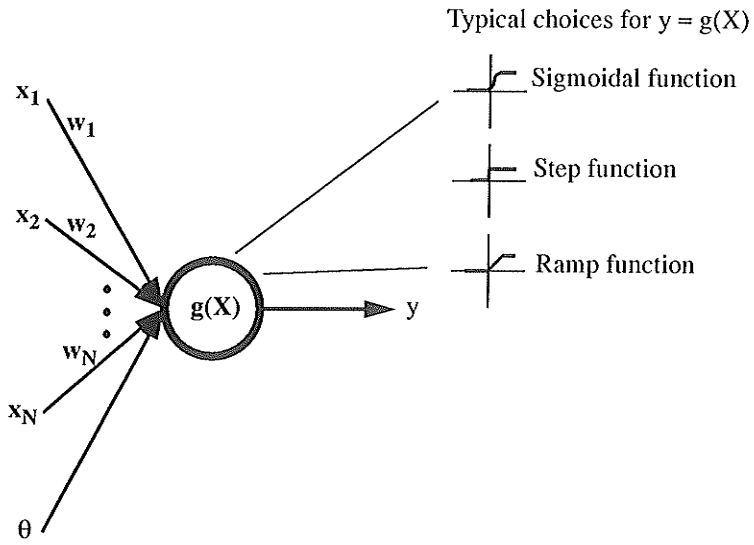


FIGURE 2.5. A simple neuron model.

Figure 2.5 also shows several typical choices for the shapes of the neurons' transfer functions. It is here noted that said transfer functions are typically non-linear and output band-limited to ranges of $[0, 1]$ or $[-1, 1]$. One of the most common choices for transfer functions belongs to the *sigmoidal* class and is known as the *logistic function*. The logistic function is defined by Equation 2.7 below:

$$g(x) = \frac{1}{1 + e^{-x}} \tag{2.7}$$

ANNs have such great computational abilities due to the non-linear nature of their neurons and the fact that they are systems of many highly interconnected such units. Such a computational scheme is known as *parallel distributed processing*.

2.2.2.1 Feedforward ANNs

Two main classes of supervised learning ANNs are obtained from the different ways in which these neurons are interconnected. These different system architectures are called feedforward and constraint satisfaction. The feedforward type architectures are currently the most prominent family. In this group, networks are structured in layers. Each layer may have any number of neurons (nodes) as required. Typically, the layers are fully interconnected. Figure 2.6 shows a three-layer feedforward type ANN referred to as a *Multi-Layer Perceptron (MLP).*

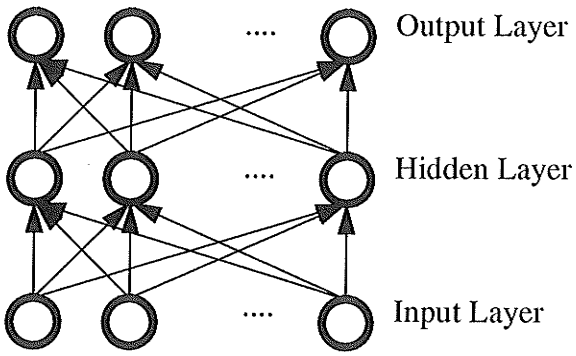


FIGURE 2.6. A Multi Layer Perceptron.

The training process for such networks consists of presenting each input pattern and then computing the difference vector between the network's expected and actual outputs. The network's overall performance is described by a *quality factor* typically described by a sum of squared errors equation similar to Equation 2.8:

$$E_{MLP} = \frac{1}{2}\sum_{c}\sum_{i}(O_{desired} - O_{actual})^2 \tag{2.8}$$

where the first summation is over all patterns and the second summation is over all output nodes.

The quality factor is frequently referred to as an energy function. Furthermore, the process of training the network is analogously described as an optimization procedure aimed at minimizing the energy of the net.

Rumelhart et al [42] described a training algorithm for MLPs by which the weights within the network are adapted in such a fashion as to minimize the network's energy. This training method is known as *Backpropagation of Errors* (Backprop for short) and in its simplest form is a gradient descent algorithm. The partial derivatives of the energy function are computed with respect to every weight within the network. Each weight w, then may be updated at iteration step t, according to Equation 2.9:

$$w(t) = w(t-1) + \Delta w(t) \tag{2.9}$$

where

$$\Delta w(t) = -\alpha \frac{\partial E_{MLP}}{\partial w(t)} + \beta \Delta w(t-1) \quad ; \quad \begin{matrix} \alpha \in [0, 1] \\ \beta \in [0, 1] \end{matrix} \tag{2.10}$$

α is the *learning rate* parameter and is usually kept small to prevent oscillations within the weight space during the optimization process, and β is a *momentum* parameter used to speed up the learning process by filtering high frequency oscillations of the weights.

2.2.2.2 Recurrent Neural Networks

A subclass within the feedforward category of supervised ANNs are distinguished by the fact that they contain *feedback* connections from the network's output units to a set of *state* neurons. Williams and Zipser [46] devised a learning algorithm for such structures which are commonly referred to as *recurrent neural networks (RNN)*. Some of the state neurons serve as *output* units and the rest are *hidden* units. The RNN also contains a set of non-recurrent *input* units. Thus, the RNN may be compared to a feedforward network

having an arbitrary number of identically interconnected layers. By their nature, these networks operate only in discrete time. Such networks have an innate ability to process temporal signals [37]. These input signals are sequences of arbitrary length varying over discrete time.

2.2.2.3 Constraint Satisfaction Networks

The networks of the second category of supervised ANNs are known as *constraint satisfaction networks*. These networks consist of N fully interconnected neurons. Research interest increased in such neural networks after Hopfield's original work in 1982 [24]. That work used binary neurons having possible states of '0' or '1'. The weights within the network are represented by a weight matrix, $\boldsymbol{T}$, with $T_{ii} = 0$. The neuron updates are asynchronous (random in time, but having a specific mean attempt rate). The state of each neuron v_i, is readjusted according to the rule defined by Equation 2.11:

$$\begin{matrix} V_i \rightarrow 1 \\ V_i \rightarrow 0 \end{matrix} \quad \text{if} \quad \sum_{j \neq i} T_{ij} V_j \quad \begin{matrix} > U_i \\ < U_i \end{matrix} \tag{2.11}$$

where U_i is the threshold or bias value for the i^{th} neuron.

This type of ANN functions as a content-addressable or associative memory. A set of neuron states $V^s, s = 1 \ldots n$ are stored in the network through the use of a one-shot storage prescription algorithm[24]

$$T_{ij} = \sum_s (2V_i^s - 1)\,(2V_j^s - 1) \text{ with } T_{ii} = 0. \tag{2.12}$$

Consider the special case where $T_{ij} = T_{ji}$ and an energy function is defined as:

$$E_H = -\frac{1}{2} \sum_{i \neq j} \sum T_{ij} V_i V_j \tag{2.13}$$

The stored states correspond to local energy minima in the state space. These energy minima have corresponding basins of attraction. If the network is placed into an initial state (not one of the stored states) and the neurons are allowed to adjust themselves according to their update rule, eventually, the network will end up in one of its local minima.

A Hopfield network's appealingly quick learning algorithm makes it a likely candidate for both associative memories and pattern classification problems. However, Hopfield nets do possess some limiting factors as well. Their pattern recall times are prolonged by the networks' *settling* time requirement. Additionally, storage capacity is relatively limited at about $0.15N$ for a network having N neurons [24]. If there is an attempt to store more items than that limit, spurious unwanted local minima occur. However, such minima are usually not very deep, and their detrimental effects may be somewhat alleviated through an *unlearning* process [25]. In [26], Hopfield showed that his networks could also be constructed of neurons having graded response characteristics.

Another type of ANN is known as a Boltzmann machine (BM) [1,22]. The BM is a generalization of the original Hopfield network. In the BM's learning algorithm, units update their states according to a *stochastic* decision rule. The neurons in a BM are separated into categories of input, hidden, and output units. Each input-output pair to be learned is coded as a state of the visible units of the network.

During training, three probability distributions are associated with each of these states. The first represents the probability that the inputs are in a state α. The second represents the probability that the outputs are in a state β given an input state of α. The final probability represents the probability that the outputs are in the state β when the inputs are clamped in the state α. With the incorporation of a *simulated annealing* approach in its learning algorithm, the BM learns to minimize the distance between the probability

distributions of the output units when they are clamped and when they are not clamped (for all of the states to be stored).

According to Hinton [23]...

> "Stochastic Boltzmann machines learn slowly, partly because of the time required to reach thermal equilibrium and partly because the learning is driven by the difference between two noisy variables, so these variables must be sampled for a long time at thermal equilibrium to reduce the noise."

This statement implies that BMs learn slowly partly because of their stochastic nature. A deterministic BM (DBM) was introduced by Peterson and Anderson [40] by using a mean field approximation to simulate the stochastic system. The capabilities of mean field theory (MFT) learning were explored in [41], and a proof that it performed gradient descent in weight space was shown in [23]. The DBM, also known as the *mean field machine*, is an appealing ANN due to its relatively short training times (as compared to BMs and backprop), its bidirectionality (may be a content-addressable memory as well as a classifier), and its ease of implementability [41].

2.3 Unsupervised Learning Algorithms

The pattern recognition techniques described in this section are represented by the bottom two quadrants of Figure 2.2, "A Taxonomy of Pattern Classifiers." on page 6. Since unsupervised pattern recognition deals with unlabeled data, a training datum's class is not previously known. Consequently, the number of classes to be found is usually not precisely known either. Most unsupervised pattern recognition techniques expect prior knowledge of the number of classes to be found within the data. Each class would then be represented by a prototype or centroid.

However, if the number of classes is not previously known, we cannot then provide the classifier any information regarding the number of prototypes to develop. There are two

ways to solve this dilemma. For the first method, *cluster validity indexes* are evaluated given an assumed number of classes in the data. These measures are evaluated over a range of cluster counts. The cluster count having the best validity index is then declared to be the best way to partition the data. The second approach involves the development of a large number of prototypes (vector quantization to many levels). Since there are now more prototypes than classes, a post-processing stage is invoked to group the prototypes into the appropriate number of clusters. Both classification techniques are discussed in Chapter 4 of this report.

As previously mentioned, all unsupervised learning algorithms perform vector quantization of the input space. This quantization or discretization of the input space is not uniform. Instead, the nodes end up being distributed in such a fashion as to best approximate the distribution of the data set. Therefore given a set number of nodes, the data representation and classification capabilities of the algorithms in this family are essentially equivalent. Such classifiers differ in the manner which they were trained. That is, each algorithm uses a unique node update procedure and contains the necessary data structures and abstractions for its implementation. In order to comparatively evaluate these algorithms, we must examine their relative complexities.

2.3.1 Classical Unsupervised Learning

Unsupervised pattern recognition is often referred to as *cluster analysis*. The classical approach pattern recognition using unsupervised learning has been to maintain one prototype per class to be identified within the data set. These prototypes, which are points within the data space, are defined to be the *centroids* of the clusters formed from the data set. Cluster membership values for each pattern in the data set are assigned based upon the pattern's relative distance with respect to all of the cluster prototypes. If the training

patterns have been properly clustered, the groups will accurately reflect the classes within the data.

Ball and Hall [3] introduced an iterative algorithm to compute the appropriate coordinates for the cluster prototypes. Originally known as *Iterative Self-Organizing Data Analysis Technique (ISODATA)*, the algorithm is now commonly referred to as the *C-means* algorithm (CM). Since its introduction, there have been numerous adaptations and hybridizations of the CM algorithm to accommodate classification in data sets having clusters of irregular shape, variable density, variable size, an unknown number, or noisy data [4,5,9,10,17,34]. A common property of these algorithms is that all of the data is considered in parallel. This necessitates complete knowledge of the training set prior to execution.

2.3.1.1 Data Partition Matrices

In addition to the cluster prototypes, each of these algorithms maintain a *data partition matrix (DPM)*. A particular DPM, U, consisting of elements u_{ik}, contains one row for each of the c clusters to be formed, and one column for each of the n patterns in the data set.

There are two distinct classes of matrices which are used to partition a data set. The differences stem from the sets of allowable values for u_{ik}, which describe the membership of each pattern to each class. Hard (Boolean) logic only permits a pattern to have either full membership in a particular category or none at all ($u_{ik} \in \{0, 1\}$). Using fuzzy sets introduced by Zadeh [47], allows patterns to have graded membership values in the range $[0, 1]$. In Bezdek's [4] notation, the sets of all hard DPMs, M_c, and all fuzzy DPMs, M_{fc}, which separate n data points into c clusters, are described by Equations (2.14) and (2.15) respectively:

$$M_c = \{U \in V_{cn} | u_{ik} \in \{0, 1\}\ \forall i, k; \sum_{i=1}^{c} u_{ik}=1\ \forall k; 0 < \sum_{k=1}^{n} u_{ik} < n \forall i\} \quad (2.14)$$

$$M_{fc} = \{U \in V_{cn} | u_{ik} \in [0, 1]\ \forall i, k; \sum_{i=1}^{c} u_{ik}=1\ \forall k; 0 < \sum_{k=1}^{n} u_{ik} < n \forall i\} \quad (2.15)$$

where V_{cn} is the set of all $c \times n$ matrices. The first constraint stipulates that the sum of each pattern's membership values for the separate clusters must always be equal to 1. The second constraint ensures that each cluster must be represented by at least one but not all of the data patterns.

2.3.1.2 C-Means

The CM algorithm begins with an initially random set of cluster prototypes, $\{v_i | i \in [1, c]\}$ in the input space $\Re^p$. Bearing in mind the above definitions for data partition matrices, we define the i^{th} cluster's prototype to be the centroid of its constituent patterns. This point in the input space is described by Equation 2.16:

$$v_i = \frac{\sum_{k=1}^{n} (u_{ik})^m x_k}{\sum_{k=1}^{n} (u_{ik})^m} \quad (2.16)$$

where $x_k \in X$ (the set of p dimensional input vectors) and m is referred to as being a fuzzification factor, meaning that the elements of the DPMs will become less crisp as m is increased. We set $m = 1$ for hard DPMs or let $m \in (1, \infty)$ is for fuzzy DPMs (typically, the useful range for $m = [1.5, 4]$). This results in two classes of CM classifiers, *Hard C-Means (HCM)* and *Fuzzy C-Means (FCM)* respectively.

The validity of the DPM (and subsequently the cluster prototypes) is reflected by every centroid's ability to represent each pattern in its class. A particular DPM's performance index is measured by a *Within Groups Sum of Squared Errors* functional $J_m(\boldsymbol{U}, \boldsymbol{v})$ as defined below:

$$J_m(\boldsymbol{U}, \boldsymbol{v}) = \sum_{i=1}^{c} \sum_{k=1}^{N} (u_{ik})^m (d_{ik})^2 \tag{2.17}$$

where $d_{ik} = \|\boldsymbol{x}_k - \boldsymbol{v}_i\|$ and $\|.\|$ is any inner product induced norm on $\Re^p$.

As mentioned previously, CM is an iterative algorithm. At each new iteration, the new DPM is computed in the following manner:

In the case of HCM:

$$\begin{aligned} &u_{ik}(t)=1 \quad \text{if} \quad d_{ik}(t) = \min_j (d_{jk}(t)) \\ &u_{ik}(t)=0 \text{ otherwise} \end{aligned} \tag{2.18}$$

In the case of FCM:

If $\exists d_{ik}(t) = 0$ then set

$$u_{ik}(t) = 1 \text{ and } u_{jk}(t) = 0 \quad \forall j \neq i \tag{2.19}$$

otherwise

$$u_{ik}(t) = \frac{1}{\sum_{j=1}^{c} (d_{ik}/d_{jk})^{2/(m-1)}}. \tag{2.20}$$

In the same step, each cluster prototype, $\boldsymbol{v}_i$, is updated according to Equation 2.16 above. The CM training algorithms may be viewed as a procedure of optimizing a DPM (or

alternately a set of prototype vectors) in order to minimize the objective functional. A proof of these algorithms' convergence to local minimum is given in [4,p.67].

2.3.2 Unsupervised Learning Neural Networks

Since unsupervised ANNs are not provided with target output values for use in the learning process, neurons output values do not directly represent the network's input vector's class. Instead, the network's task is to attempt to represent the input space. Each neuron assumes responsibility for the representation, i.e. acts as the prototype, of a subset of the set of input vectors. It outputs a dissimilarity measure which acts as an indication of the quality of its representation of the network's current input. Typically, the match value is expressed as a distance measure as defined by an inner product induced norm of the difference vector between the network input and the particular neuron's current weight vector. That is, given $x_i \in X \in \Re^p$ as the current input vector to the network. The output value of the jth neuron, o_j, is defined by:

$$o_j = d(i,j) = \|x_i - w_j\| \tag{2.21}$$

where w_j is the jth neuron's current weight vector and $\|.\|$ is any inner product induced norm on $\Re^p$.

During training, each neuron computes its match to the current input vector. Through a competitive process, usually "winner take all", a neuron's weight vector is updated in order to improve its ability to represent the current input. That is, if the jth neuron is the winner, it is moved closer to the current point in the input space by some fraction, $\lambda \in (0, 1)$ (often labelled as the network's learning rate), of its difference vector:

$$w_{j_{new}} = w_{j_{old}} + \lambda (x_i - w_{j_{old}}) \tag{2.22}$$

Unsupervised ANNs are often deemed to be more biologically plausible than their supervised counterparts, especially the MLP algorithm which is dependant on the computation of various partial derivatives.

2.3.2.1 Kohonen's Self-organizing Maps

Kohonen introduced a class of unsupervised ANNs known as Self-Organizing Maps (SOM) [31,32]. SOMs are biologically motivated from examples such as the cortical maps and tactile sensory maps in the brain. Figure 2.7 below illustrates the structure of a two dimensional SOM. The structure consists of a collection of neurons such as those described by Equation 2.21 above. Each of the neurons has weights connected to all of the network's input units. The neurons are arranged in a low dimensional array (typically 2-D or 3-D). Neighborhoods surrounding the neurons, $\aleph\ (j, k)$ (using a 2-D example), are defined to consist of the 'nearby' elements in the array.

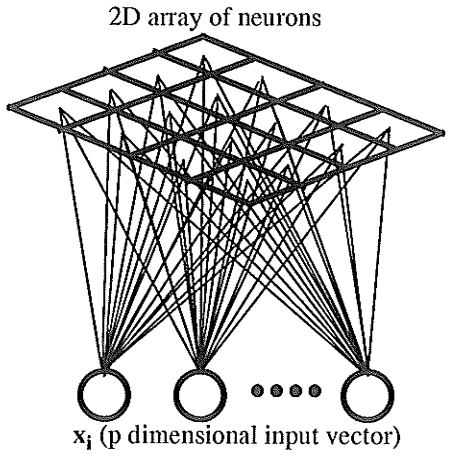


FIGURE 2.7. Kohonen's Self-organizing Map.

Upon the random presentation of an input vector to the network, a competitive process is initiated with a 'winner' node, for instance neuron (j, k), being declared if:

$$o_{jk} = min\,(o_{mn})\;\forall m, n \tag{2.23}$$

The weights of all the neurons in the SOM are then updated according to the following equations:

$$w_{mn}(t+1) = w_{mn}(t) + \Im\,(m, n, j, k, t)\;(x_i - w_{mn}(t))\;\forall\,(m, n) \in \aleph\,(j, k, t) \tag{2.24a}$$

$$w_{mn}(t+1) = w_{mn}(t) \quad \forall\,(m, n) \notin \aleph\,(j, k, t) \tag{2.24b}$$

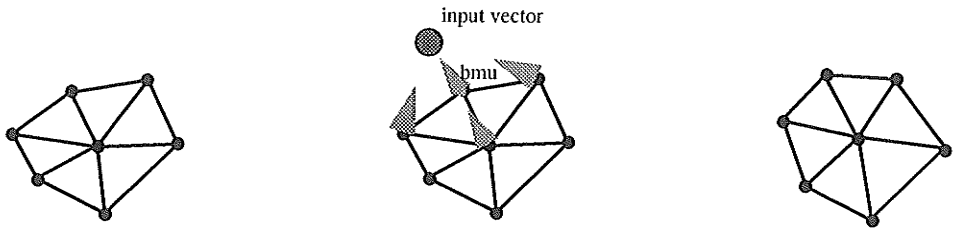
Equation 2.24a is similar to Equation 2.22. The difference between the two is the replacement of the simple learning parameter, λ, by $\Im\,(m, n, j, k, t) \in (0, 1)$, a function of the neuron's proximity to the winner node within the SOM and the training step. During training, both the learning parameter and the winner neuron's effective neighborhood are typically decreased with time. With each iteration of the algorithm, the positions of the neurons in the input space better represent the distribution of the input set. The training algorithm is terminated once the sum of weight changes in an *epoch* (one time through the training set) drops below a preset threshold value.

One of the SOM's accolades is that the mapping preserves the *topology* of the input space. This means that input vectors which are 'close' to each other in $\Re^p$ will result in winner nodes which are close to each other in the map. However, this feature of topology preservation is not a criterium which is specifically used to define the algorithm's weight update rules. Instead, it is a quality which emerges as a side-effect of the neighborhoods implemented around the winner nodes during the training. Using metric topology preserving measures, Bezdek and Pal [6] show that the topology preservation is not implemented as effectively as in other feature extracting algorithms such as *Principal Components Analysis* or *Sammon's algorithm*. Unlike the SOM, both of these algorithms have the ability to produce distinct mappings for each input pattern. SOMs must quantize the input space to as many points as there are nodes (typically many less nodes than input

vectors) and are therefore limited in the number of distinct mappings they may produce. It is this discrete nature that prohibits the SOM from improving the topology preservation of the mapping.

2.3.2.2 Growing Cell Structures

Fritzke[14] introduced a class of ANNs similar to SOMs which he called *Growing Cell Structures (GCS)*. The similarities arise from the weight update procedures and a more loosely defined neighborhood function. The GCS begins with three interconnected cells (neurons) each having a weight vector, $w_i \in \Re^p$. Figure 2.8 below depicts GCS's weight adaptation process (at a point where the structure has grown to seven cells).



(a) original cell positions.

(b) presentation of a new input vector.

(c) resulting cell positions.

FIGURE 2.8. The Growing Cell Structure's adaptation process. Source: Fritke[14].

All cells in the structure ascertain their similarity to the current input. A competition is then held to determine the 'best matching unit' which is declared the winner. The winning cell and its immediate neighbors are then updated according to the two equations below.

$$w_{bmu}(t+1) = w_{bmu}(t) + \alpha_{bmu}(x - w_{bmu}(t)) \quad (2.25a)$$

$$w_i(t+1) = w_i(t) \quad \forall i \in \text{bmu's neighbors} \quad (2.25b)$$

The winning cell also increases an internal error value by the distance to the input.

As with the SOM, each cell's neighborhood is rigorously defined. However, the GCS's key feature is its ability to dynamically redefine neighborhoods by adding or removing cells as required. After a preset number of input presentations, the GCS determines which cell has the highest error value. This so called 'black sheep' is located a region of the input space which the GCS is not adequately representing the distribution of the data. The link between this black sheep and its farthest neighbor, cell f, is then broken and a new cell, connected to both, is added at the midpoint. In order to maintain the plane of triangles structure, this new cell is also linked to each cell that the black sheep and f still have in common. The new cell's error value is initialized to the average of its neighbors' error values. Then the error values of the neighbors are proportionally reduced to keep the total error of the cells involved constant. Through this process, the GCS adjusts its weights and 'grows' to form a representation of the data set.

Each cell also tracks the number of presentations that have past since it was a best matching unit. A cell removal parameter, based upon the number of cells in the structure, is maintained in order to determine the probability of removing a cell from the structure. If a cell is located in a region of low input probability, it will not have won any competitions for a long time. Since this cell does not take part in the proper representation of the data, it should be removed along with its links. It has been reported that GCSs are able to detect clusters of similar patterns and accurately represent the (unknown) probability distribution of the input data set [14].

2.3.2.3 Adaptive Resonance Theory

In an attempt to overcome the stability-plasticity dilemma, Carpenter and Grossberg [7] developed the *Adaptive Resonance Theory (ART)* class of unsupervised ANNs. Nonstationary inputs necessitate the network having the ability to learn new patterns without necessarily forgetting the old. ART networks evaluate patterns with respect to a

set of templates for current classes. A *vigilance parameter* is then used to determine if the patterns have been correctly classified. If a pattern is encountered which was not closely matched by the current set of class prototypes, a new class may then be allocated from a pool of reserves. Due to time restrictions, this type of ANN was not implemented for the report.

2.3.2.4 Fuzzy Kohonen Clustering Networks

In order to overcome some of the SOM algorithm's deficiencies (lack of a well-defined objective function, sequential weight updates, and a poorly defined termination threshold), Bezdek et al [5] propose a class of unsupervised ANNs which they call *Fuzzy Kohonen Clustering Networks (FKCN).* The structure of the FKCN is shown in Figure 2.9 below. The algorithm is a hybridization of SOMs and CM - having elements from each.

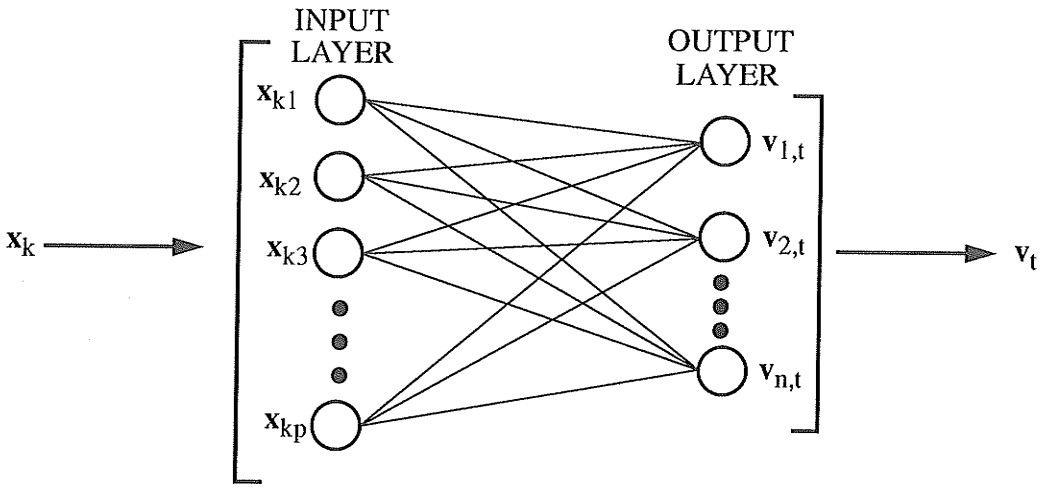


FIGURE 2.9. Structure of the FKCN.
Source: Bezdek[5].

From the CM algorithm, the FKCN training process optimizes a fuzzy objective functional (see Equation 2.17) which is constructed via a fuzzy DPM (see Equation 2.15.)

The elements of the FKCN's DPM are calculated according to Equation 2.26 below. The equation is identical to that used in the definition of the CM algorithm (Equation 2.20), with the single exception that the fuzzification factor, m, is linearly decreased with each iteration.

$$u_{ik,t} = \left(\sum_{j=1}^{c} (d_{ik}/d_{jk})^{2/(m_t - 1)} \right)^{-1} \tag{2.26}$$

The FKCN's weights are updated in a neural network fashion. A difference equation (variation of Equation 2.16), describing this process is given as follows:

$$v_{i,t+1} = v_{i,t} + \frac{\sum_{k=1}^{n} \alpha_{ik,t} (x_k - v_{i,t})}{\sum_{s=1}^{n} \alpha_{is,t}} \text{ where} \tag{2.27}$$

$$\alpha_{ik,t} = (u_{ik,t})^{m_t} \text{ where} \tag{2.28}$$

$$m_t = (m_0 - t\Delta m); \Delta m = (m_0 - 1)/t_{max} \tag{2.29}$$

The next positions for the prototypes are obtained by adjusting the previous points by a weighted average of distances from the input set.

There is no well-defined neighborhood function for the FKCN. However, the network is self-organizing because the m_t factor inherently "sharpens" the focus of the affect of each data vector on the weight updates. As t is increased, m_t tends to one. This in turn forces $\alpha_{ik,t}$ to either one or zero - thereby reducing the effective update neighborhood.

The FKCN's improvements over the SOM include a parallel weight updates, non-sequential data presentation and well-defined termination. However, these benefits are gain at the cost of the maintenance of a DPM.

2.4 Limitations of Unsupervised Learning

When using unsupervised learning in pattern recognition, we must be fully aware of the representational capabilities of the nodes we are using. Such representational capabilities are independent of the manner in which the node was developed; be it through cluster analysis or neural networks. Since the quality of a nodes's representation of an input pattern is given by a dissimilarity measure, the characteristics of the node's region of attraction (the portion of the input space which it best represents) are defined by the metric used. The metrics used to define the nodes' dissimilarity measures are given by inner-product norms on the input space. Therefore, a node's region of attraction is by definition limited to being convex and unimodal in nature.

Convexity implies that a single node will not be able to accurately represent a concave region of the input space. The accurate representation of such a region would require a union of the regions of two or more nodes. However, as we shall see in Chapter 4 of this report, such a task will necessitate the introduction of a post-processing interpretive stage to the classifier in order to properly group the nodes.

The node's unimodal nature implies that the quality of representation decreases (i.e. dissimilarity increases) for input patterns as they become further away from the node. The previous statement may at first may seem to be to be trivial. However, it implies that a node will not be able to well represent a multi-modal class in the problem. Such a case would again lead to the necessitation of a post-processing stage.

Let us now examine several possible norms to be used by the nodes when computing their distance functions. In particular, we shall discuss the choices of the $\|.\|_1$, $\|.\|_2$, and $\|.\|_A$ norms on the p dimensional input space. $\|.\|_1$ is a sum over all the dimensions of the absolute values of the component differences. That is, $\|x - w\|_1 = \sum_{i=1}^{p} |x_i - w_i|$. Choosing this norm leads to nodes having *hyper-cubical* regions of attraction. The $\|.\|_2$ norm is the Euclidean norm on a p-dimensional space, i.e., $\|x - w\|_2 = \left[\sum_{i=1}^{p} (x_i - w_i)^2 \right]^{1/2}$. Nodes using this norm are best able to represent *hyper-spherical* clusters in the data. Finally, the matrix induced norm, $\|.\|_A$, is defined by $\|x - w\|_A = [\,(x - w)^T A\,(x - w)\,]^{1/2}$. The use of this norm facilitates the representation of *hyper-ellipsoidal* clusters within the data. Moreover, if matrix **A** is chosen as the inverse of the fuzzy covariance matrix formed by the central members of the cluster being represented by the node, the characteristics of the node's region of attraction take on those of a multivariate gaussian cluster (see UFP-ONC algorithm in Chapter 4).

This leads us to the conclusion that unsupervised learning really is not completely unsupervised after all! In order to achieve satisfactory data classification or representation results, the *user* will be required to make a decision about which distance metric (i.e., expected cluster characteristics) should be used. Even an algorithm such as the UFP-ONC, in which each node's distance metric is constantly being refined, begins with the assumption that all clusters were generated by multivariate gaussian distributions. Additionally, especially difficult classification problems, may involve the user's expertise during a post-processing stage.

Vector quantization has become a popular choice for codebook generation in such applications as image and audio compression [19,33,44]. Indeed, small-scale SOM VLSI implementations of such applications have been reported [12,20,36]. The SOMs are trained until their nodes well represent the input data set (groups of LPC coefficients for audio or a set of pixel values for image compression). During the execution phase, the

winning node's index becomes the value to be stored or transmitted in order to later be deindexed for the sample's reconstruction. Such compression paradigms are lossy. However, compression ratios of 64:1 and higher, while maintaining tolerable levels of signal degradation, have been reported.

Hecht-Nielsen [21] suggests that vector quantizational techniques will not achieve the 1000:1 or greater compression ratios that were once dreamed of. For to accomplish such a task while maintaining tolerable levels of degradation would require a very large number of nodes. In such a case, the corresponding size of the codebooks and the number of bits that one would need to store or transmit in order to identify the index of the winner node would be prohibitive. Hecht-Nielsen [21] also suggests that in such a case, a better approach might be to transmit a winner node's co-ordinates rather than it's index, perhaps using Principal Components Analysis.

2.5 Summary

To summarize, in this chapter, we introduced pattern recognition systems and described the role of the classifier. We introduced the difference between supervised and unsupervised learning. Three intuitive classification techniques (together with corresponding classical and neural network implementations) were described. It was shown that unsupervised learning must take the form of nonuniform vector quantization. The chapter concluded with a discussion of some of the limitations of unsupervised learning, especially those stemming from the representational capabilities of a single node. The following chapter of the report describes the implementations of several of these algorithms and their respective features.

Chapter 3

Algorithms Implemented

In this chapter, we present the implementations of three of the unsupervised algorithms from Section 2.3 which were evaluated during the course of the research. Specifically, these are the SOM, CM, and FKCN algorithms. All the software to implement these algorithms was written in the C programming language and executed on SunSparc workstations under a unix based operating system.

3.1 SOM Algorithm

3.1.1 SOM Implementation

Figure 3.1 on page 38 shows a simplified flowchart for the implementation of the SOM algorithm as described in Section 2.3.2.1 of the previous chapter. The corresponding pseudo-code is show in Figure 3.2. From the flowchart, it is evident that there are two nested loops in which the majority of the necessary computations are performed. The major loop iterates over the training epochs and contains within it a loop for the presentation of each element in the data set with it corresponding weight update operations.

The minimal algorithm is remarkably simple. After the learning rate is set, and the node weights are initialized (typically to small random values), there are three basic steps to be performed for each data point:

- Calculate distances from the input to the N nodes.
- Determine the winner.
- Update the map's weights according to (2.24a) and (2.24b).

Neighborhoods are implemented as squares consisting of the nodes centered about the winner.

The algorithm's rate of convergence may be improved through such techniques as scheduling reduction rates per iteration for the learning parameter and neighborhood size. Throughout the research, exponential as well as linear rates of decay were investigated. It was evident that the convergence rate is notably dependant on the initial values for the weights and varies with each problem. Therefore, it is difficult, if not impossible, to determine beforehand which specific techniques would be best suited for a specific problem.

The SOM algorithm will converge for any norm induced metric used in calculating distances. However, the choice of metric to be used is dependant on the application for which the SOM is to be used. For instance, when using vector quantization in speech compression, it has been reported [33] that better reconstruction results are obtained if the Ikaturo-Saito metric is used instead of Euclidean distances. In this way, one may take advantage of deficiencies in human hearing in order to place more emphasis on properly reconstructing the important speech components, while not wasting effort on the others. For variable prediction applications, improved results are obtained if a weighted distance measure is chosen as opposed to Euclidean or manhattan metric. This results in the nodes more appropriately quantizing the dependant dimensions of the input set.

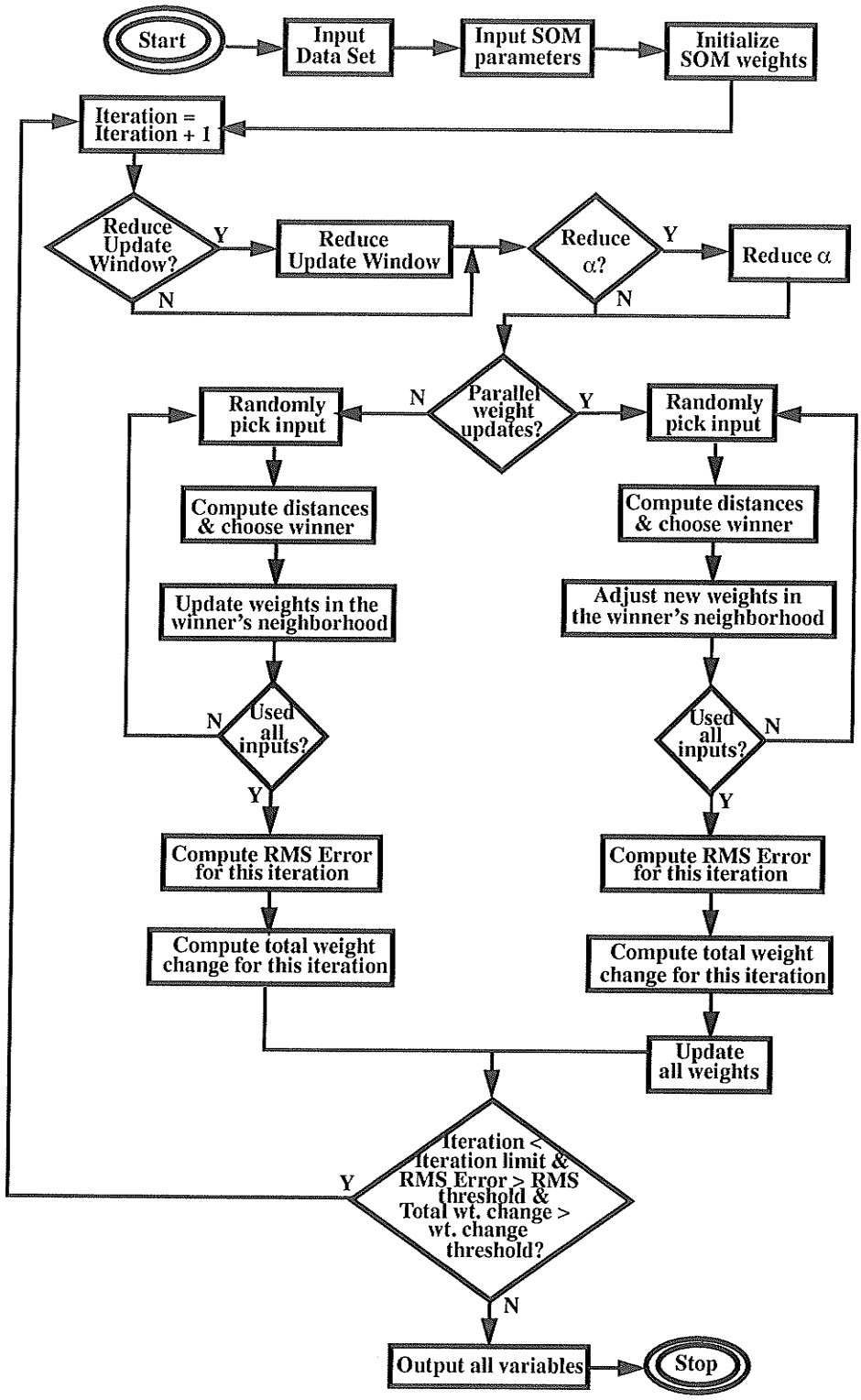


FIGURE 3.1. Flowchart for the SOM algorithm.

```
Begin {SOM}

    Input p, n, X = {x_i ∈ ℜ^p: i = 1, 2, ..., n} ;
    Input MapWidth, MapHeight, εRMS, εWeights, MaxIter;
    Initialize Weight(0) = {weight_ij ∈ ℜ^p:i=1,2, ...MapHeight;j = 1, 2, ..., MapWidth}
        with small random numbers;
    Input MetricType, ParallelUpdateFlag;
    Input WindowSize, ReduceWindowFlag, ReduceWindowType;
    Input Alpha, ReduceAlphaFlag, ReduceAlphaType;
    Input ConscienceFlag, ConscienceB, ConscienceC;
    Initialize p(0) = {p_ij = 0.0; i = 1, 2, ..., MapHeight;j = 1, 2, ..., MapWidth} ;
                      {win fraction}
    Initialize b(0) = {b_ij = 0.0; i = 1, 2, ..., MapHeight;j = 1, 2, ..., MapWidth} ;
    t ← 1;            {each node's conscience level}
    Loop {t}
        If ReduceWindowFlag then
            WindowSize ← ReduceWindow(WindowSize, ReduceWindowType) ;
        If ReduceAlphaFlag then Alpha ← ReduceAlpha(Alpha, ReduceAlphaType) ;
        If ParallelUpdateFlag then
            Begin
            Count ← 1;
            Loop {Count}
                If ConscienceFlag then UpdateThresholds(ConscienceC) ;
                RandomX ← RandomPick(X, n) ; i,j ← Winner(RandomX) ;
                If ConscienceFlag then UpdateWinFractions(ConscienceB, i,j) ;
                ParallelUpdateNeighborhoodWeights(i,j) ; Count ← Count + 1;
                Loop While (Count ≤ n) ; {Count}
            Weights ← UpdateWeights(Weights) ;
            RMS ← CalcRMS(X, Weights) ; DeltaWeights ← CalcDeltaWeights( ) ;
            End
        Else {sequential}
            Begin
            Count ← 1;
            Loop {Count}
                If ConscienceFlag then UpdateThresholds(ConscienceC) ;
                RandomX ← RandomPick(X, n) ; i,j ← Winner(RandomX) ;
                If ConscienceFlag then UpdateWinFractions(ConscienceB, i,j) ;
                UpdateNeighborhoodWeights(i,j) ;
                Weights ← UpdateWeights(Weights) ; RMS ← CalcRMS(X, Weights) ;
                DeltaWeights ← CalcDeltaWeights( ) ; Count ← Count + 1;
                Loop While (Count ≤ n) ; {Count}
            End
        Output RMS, DeltaWeights; t ← t + 1;
        Loop While ((t ≤ MaxIter) and (RMS ≥ εRMS) and (DeltaWeights ≥ εWeights)) ;
    OutputResults(Weights) ;

End. {SOM}
```

FIGURE 3.2. Pseudo-code for the SOM algorithm.

3.1.2 Conscience Mechanism

As stated in the previous subsection, the convergence properties of the SOM are highly dependant on initial conditions. The techniques of reducing the learning rate and neighbourhoods about winners were discussed. Even so, it is not trivial to determine the most appropriate combination of parameters for each application. One possible indication of a sub-optimal parameter combination is the under utilization of a subset of the SOM's nodes. During training, one may track the number competitions won by each node. Unless all the parameters and weights have been appropriately set or initialized, it will be evident that some relatively small subset of nodes won a majority of the competitions. Such a scenario leads to a variety of undesirable results; ranging from decreased topology preservation efficiency to non-convergence.

One method proposed to alleviate these difficulties is the so-called *conscience mechanism*. This mechanism is used in an attempt to ensure that all the SOM's nodes win their fair share of the competitions. Each neuron in the map tracks the fraction of time it wins the competition. DeSieno [11] suggests an effective means of generating such fractions according to Equation 3.1 below:

$$p_{i,j}^{new} = p_{i,j}^{old} + \beta\,(y_{i,j} + p_{i,j}^{old}) \tag{3.1}$$

where $0 < \beta \ll 1$ and $y_{i,j}$ is 1 if node (i,j) was the winner of the competition or 0 otherwise. When the conscience mechanism is turned on, the competitive process produces results according to Equations (3.2a) and (3.2b) below:

$$z_{i,j} = 1 \text{ if } \| w_{i,j} - x \| - b_{i,j} < \| w_{m,n} - x \| - b_{m,n} \forall\, (m \neq i,\, n \neq j) \tag{3.2a}$$

$$z_{i,j} = 0 \quad \text{otherwise} \tag{3.2b}$$

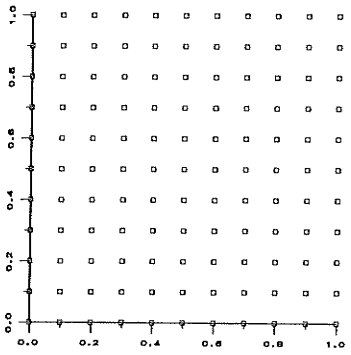
where $b_{i,j} = C(\frac{1}{N} - p_{i,j})$.

If $z_{i,j} = 1$, then neuron (i,j) is declared the winner of the competition. The SOM weights are then updated in the normal fashion according to Equations (2.24a) and (2.24b).

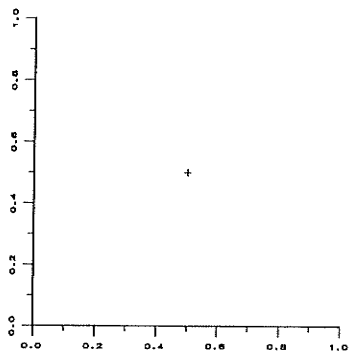
By ensuring an equitable distribution of win events, the conscience mechanism allows the SOM to quickly position its nodes to those areas of the input space which have high input density. It is critical that this 'coarse tuning' of the nodes be accomplished while the learning parameter is still relatively high. As the learning parameter is reduced, the possible weight adjustments become smaller with each iteration. Eventually, only very fine tuning of the weights will be permitted. During these final stages of the SOM's training, any conscience mechanisms should be turned off to permit the map to properly 'settle' to its best representation of the input set.

3.1.3 Illustrative Examples

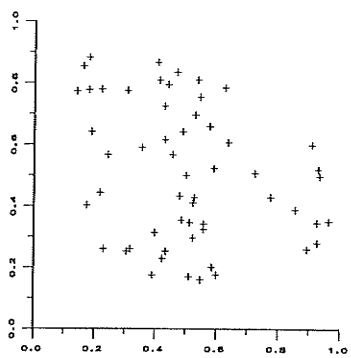
Figure 3.3(a) shows a constructed two-dimensional data set. This set contains 121 data points arranged in a uniformly spaced grid over [0,1]x[0,1]. A SOM consisting of nodes arranged in an 11x11 array was then trained on this data set. Optimal training results would leave each node exactly representing a corresponding element from the input set. The SOM's weight values during various training stages are presented in Figure 3.3(b) through (q). The weights are initialized to the mean of the input set. The learning parameter is initialized to 0.5 and is decreased linearly to 0 at the epoch iteration limit (set at 200). Throughout the training, static neighborhoods for each node were set to the eight nearest nodes in the array. The conscience mechanism was disabled after the 75th epoch.



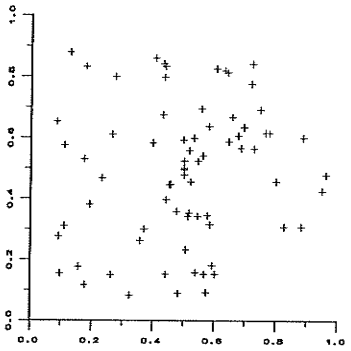
(a) Training Data.



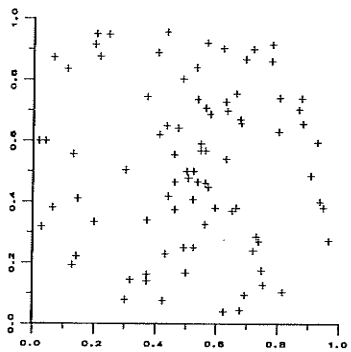
(b) Initial Weights.



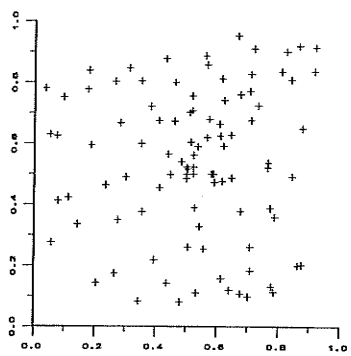
(c) After 50 presentations. (0.41 epochs.)



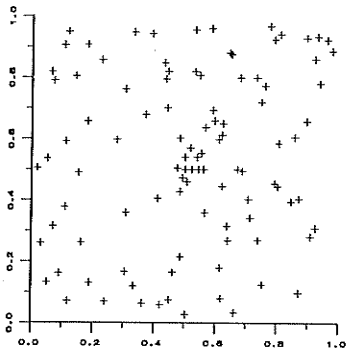
(d) After 100 presentations. (0.83 epochs.)



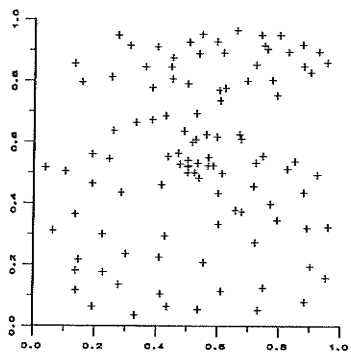
(e) After 200 presentations. (1.65 epochs.)



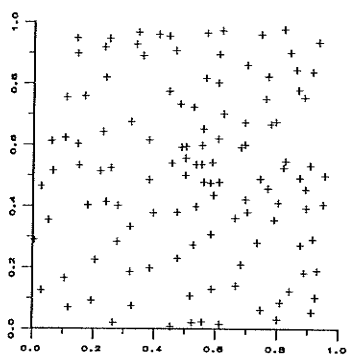
(f) After 300 presentations. (2.48 epochs.)



(g) After 400 presentations. (3.31 epochs.)

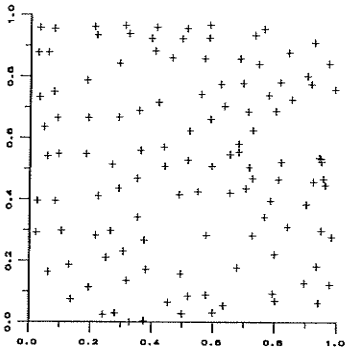


(h) After 500 presentations. (4.13 epochs.)

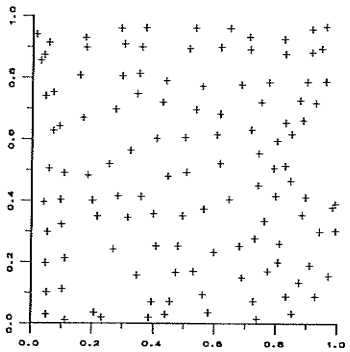


(i) After 1000 presentations. (8.26 epochs.)

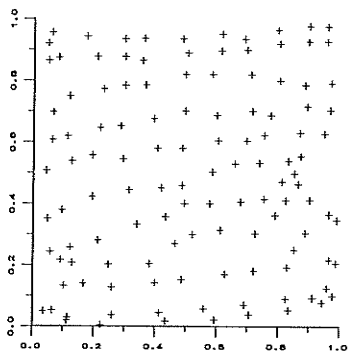
FIGURE 3.3. SOM Training Stages.
(a) Training Data Set. (b) - (i). Incremental SOM Training Results.
(continued on the following page).



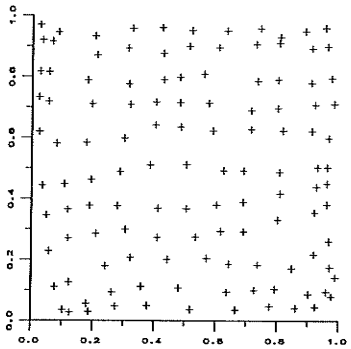
(j) After 2000 presentations. (16.53 epochs.)



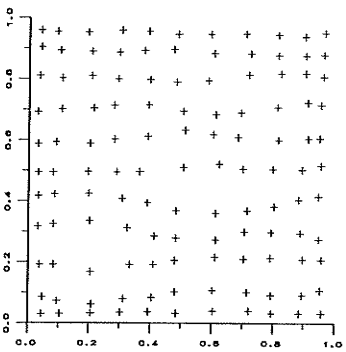
(k) After 3000 presentations. (24.79 epochs.)



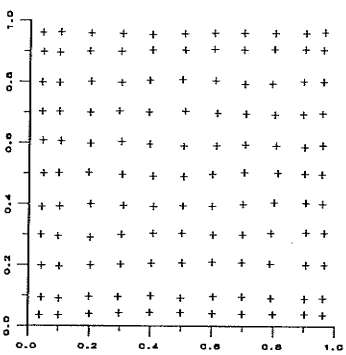
(l) After 4000 presentations. (33.06 epochs.)



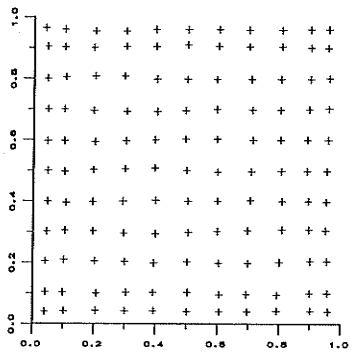
(m) After 5000 presentations. (41.32 epochs.)



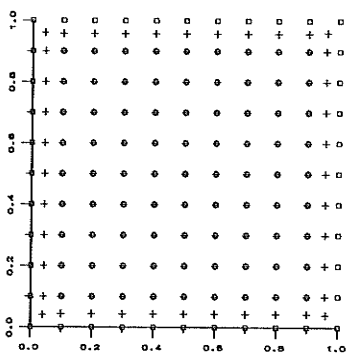
(n) After 10000 presentations. (82.64 epochs.)



(o) After 15000 presentations. (123.97 epochs.)



(p) After 20000 presentations. (165.29 epochs.)



(q) Final Weights after 24200 presentations. (200 epochs.) (Shown with the training data).

FIGURE 3.3. (continued). SOM Training Stages.
(j) - (p). Incremental SOM Training Results.
(q) SOM weights upon termination (after 200 epochs through the data set).

Referring to Figures 3.3(b) through 3.3(q), we clearly see the SOM's dynamics during training. Most of the motion in the nodes within the weight space occurs during the first 15 training epochs. During this period of coarse tuning, the map 'unfolds' itself very quickly to take on a reasonable approximation of the input vectors distribution. The great majority of the training (for this example, the final 185 epochs) are used to perform the fine tuning of the weights. A quantitative method of viewing these dynamics is shown in Figure 3.4 where the values of the average node updates and *root-mean-square* (RMS) reconstruction, i.e. representation, error after each training epoch (121 presentations) are plotted.

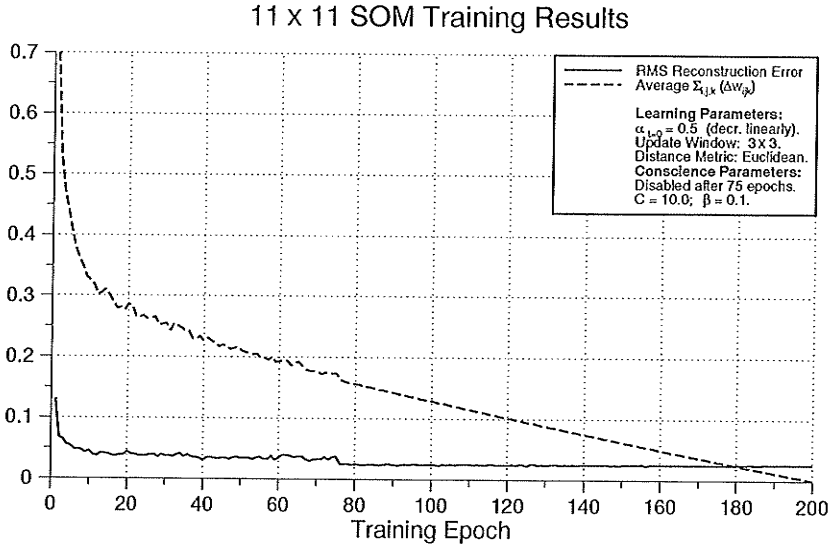


FIGURE 3.4. SOM Training Statistics for the 2-D Array Data Set.

Figure 3.4 again shows that despite the gradual decrease in the learning parameter throughout the training process, the first 15 epochs exhibit a steeper rate of reduction in the node motion. During this portion of the training, there is a correspondingly

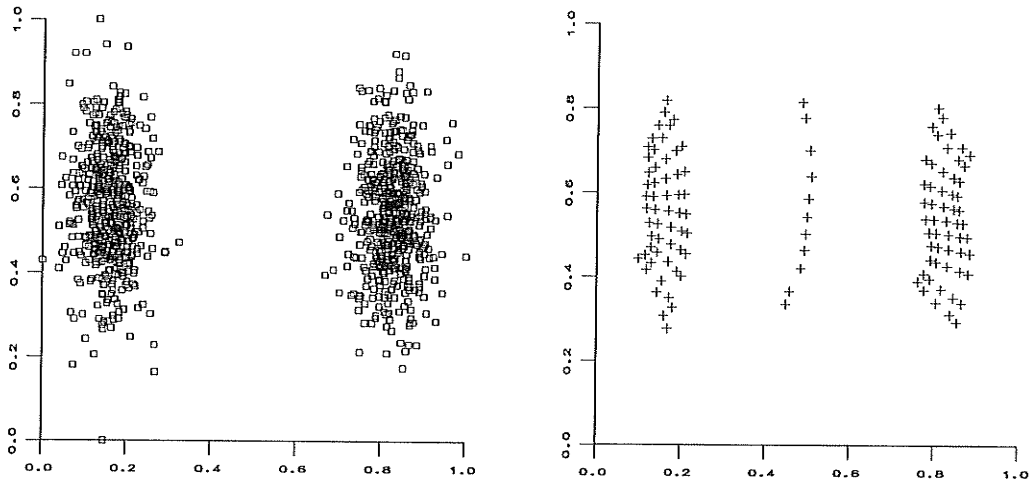
sharp decrease in the RMS reconstruction error; the root-mean-square of distances from the input vectors to their respective winner nodes.

Also evident in the figure are the effects of the conscience mechanism which is active during the first 75 training epochs. Since the conscience compels the SOM to update its under-utilized nodes, the RMS reconstruction error is kept at an 'artificially high' level (approximately 0.4). Once the conscience mechanism is disabled, the level of the RMS reconstruction error quickly drops to reflect the SOM's inherent ability to represent the data (for this case, approximately 0.25).

The final results of the SOM training are displayed in Figure 3.3(q) where the node positions are shown together with the original data. Since there are as many nodes as input vectors, perfect representation of the data would be indicated by each node exactly covering one input. In the example, all the 'interior' nodes of the map were correctly positioned. However, an obvious representational deficiency exists at the map boundaries. The SOM is inherently unable to accurately represent the vectors around the perimeter of the input set's distribution. Continued training at this point would give evidence to the *over-training* phenomena. That is, additional training would be detrimental for it would only cause the SOM's boundary nodes to move towards the mean of the input data. This phenomena is also indicated by an increase in the SOM's RMS reconstruction error. One approach to improving the SOM's mapping of the vectors along the input distribution boundaries is to 'fix' some nodes to positions just outside the input region. Due to its effects on the map's self-organization, such *partial supervision* should only be used when the input data is well-understood.

To further demonstrate some of the SOM training process intricacies, we use a second data set. This training data set, shown in Figure 3.5(a), consists of two

dimensional vectors which were generated by a pair of distinct and well-separated multivariate gaussian distributions (with 500 examples from each). The final training results are presented in Figure 3.5(b).



(a). Training Data. (b). Results after 20 training epochs.

FIGURE 3.5. SOM Training Using a 2 Class Normal Distribution Data Set.

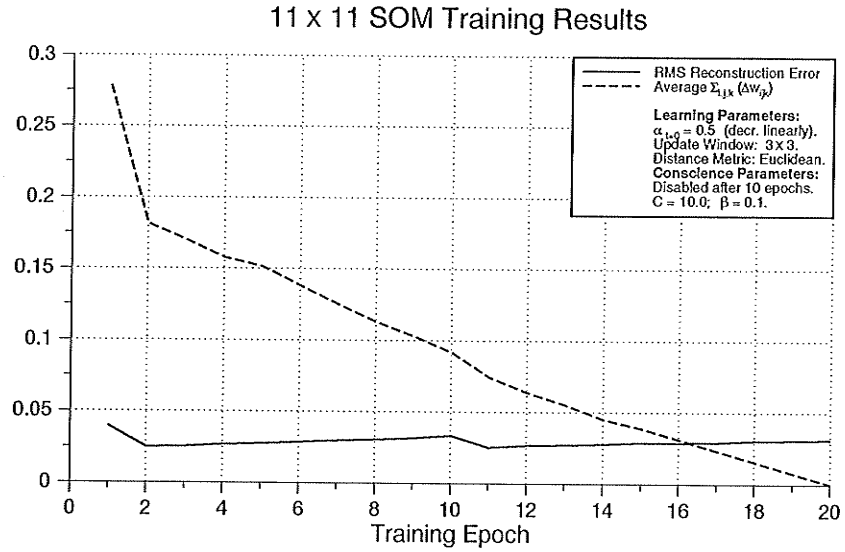


FIGURE 3.6. SOM Training Statistics for the 2 Class Normal Data Set.

Corresponding training statistics are shown in Figure 3.6. Here, the SOM used is the same size as in the previous example. All the training parameters are also set to their previous values, except that the iteration limit is set to 20 epochs and the conscience mechanism is active only during the first half of the training. When comparing the node distribution with the data which they are to represent, it is immediately obvious that a small subset, consisting of 11 nodes, were not used in the representation. During the training, these nodes became trapped in the region of low input probability between the two clusters.

Even if the conscience mechanism was used throughout the entire training process, this subset of nodes would still become trapped because each of these nodes has one or more neighbors 'pulling' it towards either cluster within the data. This effect may be countered through the development of more complex implementations of the effective neighborhood parameters in Equation 2.24a. However, the additional parameters will also be problem dependant. Such an approach will slow down the learning. If proper positioning of all nodes is required, Fritzke's [14] Growing Cell Structures described in Section 2.3.2.2 of the report should be used instead of SOMs.

3.2 GCS Algorithm

3.2.1 GCS Implementation

Figure 3.1 presents a simplified flowchart for the implementation of the GCS algorithm as described in Section 2.3.2.2 of the previous chapter of the report. The corresponding pseudo-code is given in Figure 3.2 on page 39. From these figures, it is evident that the bulk of the algorithm's complexity is due to the two nested iterative constructs used to present the data and perform the corresponding weight updates.

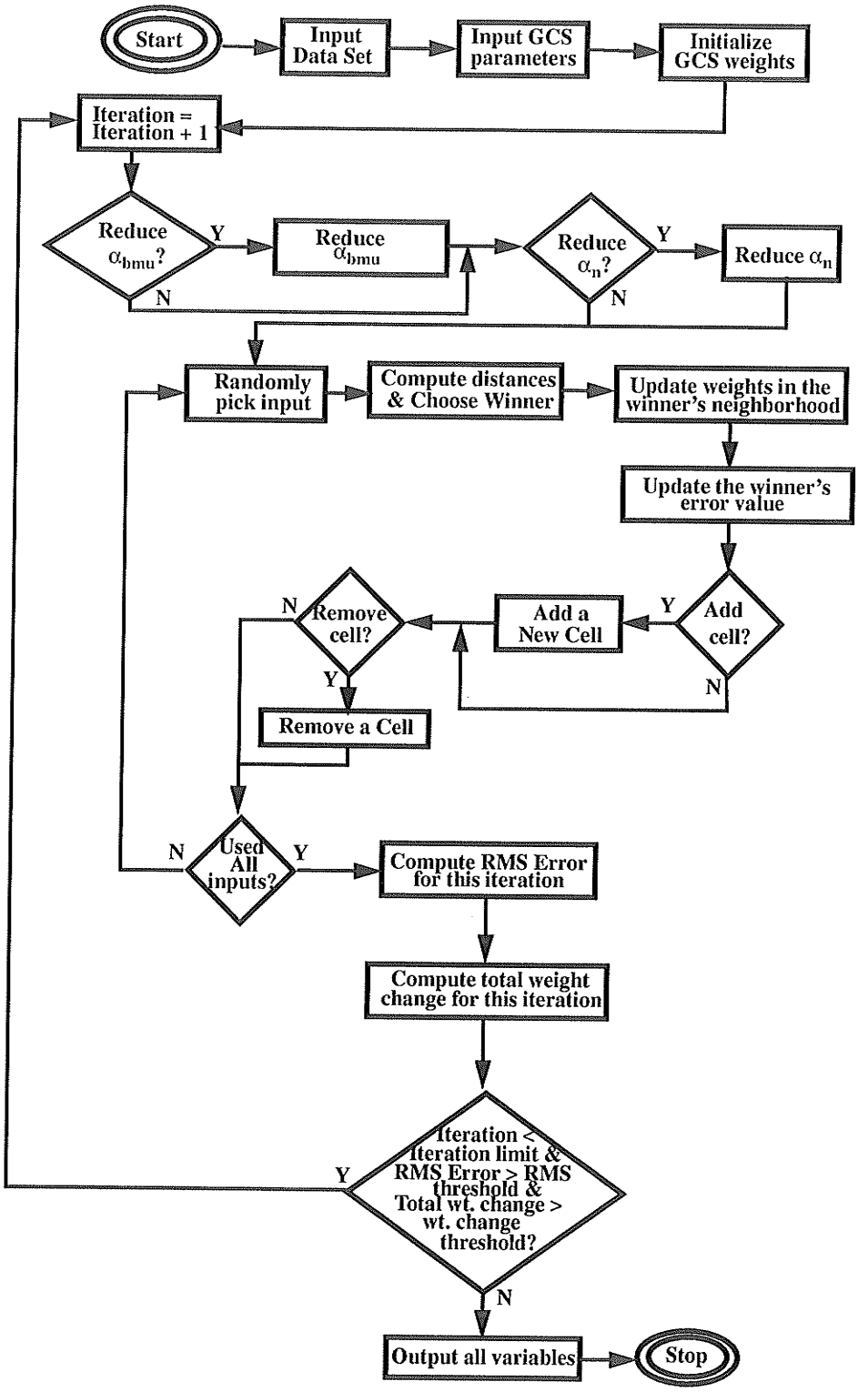


FIGURE 3.7. Flowchart for the GCS algorithm.

```
Begin {GCS}

    Input p, n, X = {x_i ∈ ℜ^p: i = 1, 2, ..., n} ;
    Input MaxCells, εRMS, εWeights, MaxIter;
    Initialize Weight(0) = {weight_i ∈ ℜ^p:i=1,2,3} ;
    Input MetricType;
    Input AlphaBmu, ReduceAlphaBmuFlag, ReduceAlphaBmuType;
    Input AlphaNeigh, ReduceAlphaNeighFlag, ReduceAlphaNeighType;
    CellCount ← 3; t ← 1;
    Loop {t}
        If ReduceAlphaBmuFlag then
            AlphaBmu ← ReduceAlphaBmu(AlphaBmu, ReduceAlphaBmuType) ;
        If ReduceAlphaNeighFlag then
            AlphaNeigh ← ReduceAlphaNeigh(AlphaNeigh, ReduceAlphaNeighType) ;
        Count ← 1;
        Loop {Count}
            RandomX ← RandomPick(X, n) ; i ← Winner(RandomX) ;
            UpdateNeighborhoodWeights(i) ;
            UpdateError(i) ; UpdateLastWin(i) ;
            Weights ← UpdateWeights(Weights) ; RMS ← CalcRMS(X, Weights) ;
            DeltaWeights ← CalcDeltaWeights( ) ; Count ← Count + 1;
            If AddCell( ) then CellCount ← CellCount + 1;
            If RemoveCell( ) then CellCount ← CellCount − 1;
            Loop While (Count ≤ n) ; {Count}
        Output RMS, DeltaWeights; t ← t + 1;
        Loop While ((t ≤ MaxIter) and (RMS ≥ εRMS) and (DeltaWeights ≥ εWeights)) ;
    OutputResults(Weights) ;

End. {GCS}
```

FIGURE 3.8. Pseudo-code for the GCS algorithm.

The input presentation and weight update procedures of the GCS are markedly similar to those employed by the SOM algorithm. However, due to the GCS's dynamically defined 'neighborhoods', its procedure for updating the neighboring cells of the winner node is slightly more complex than the one used by the SOM. Here, each node must continually track the list of its neighbors. Unlike the SOM, where neighborhoods are defined by the subscripts of an array, the cells of the GCS must maintain individual linked-lists of the identities of their neighbors.

Given the large number of input presentation - node update events which occur during the course of training, it would seem that such an arrangement for the neighborhood

representations could lead to intolerably large memory requirements for the GCS representation and long delays in the execution of each iteration. However, this is not the case.

Since the GCS begins with a small number of cells, it is able to make initial updates very quickly. The coarse adjustments of the nodes is not burdened with the necessary computation of similarity measures to many nodes. It is after the coarse tuning of the cell positions has been completed, that the GCS begins to add the additional cells required to improve the overall representation of the data. This is also the stage at which the removal of the poorly placed cells occurs. Therefore in general, given networks of equal size, GCS training times are expected to be similar, if not shorter, than those obtained for the SOM.

3.2.2 Illustrative Examples

Figure 3.9 shows the final results of a GCS which was trained on the data set given in Figure 3.3(a). This figure may be compared directly to the results of a equivalent sized SOM which was also trained for 200 epochs (c.f. Figure 3.3(q)). In the figure, the final positions of the cells are marked with crosses and are shown together with the training data, indicated with squares. In comparison to the results obtained for the SOM algorithm, it is clear that the GCS was not able to exactly represent the interior points of the data set to the same extent as the SOM. However, just as obvious is the fact that the GCS's representation of the boundary of the input distribution is superior to that given by the SOM. These results may be explained by the following.

For this example, the GCS's learning parameters, α_{BMU} and $\alpha_{NEIGHBOR}$, were held constant at 0.5 and 0.1 respectively. Since these parameters were held constant, the algorithm's rate of convergence was reduced. Had the algorithm been allowed to continue the training, the representation of the interior of the data's distribution would be improved.

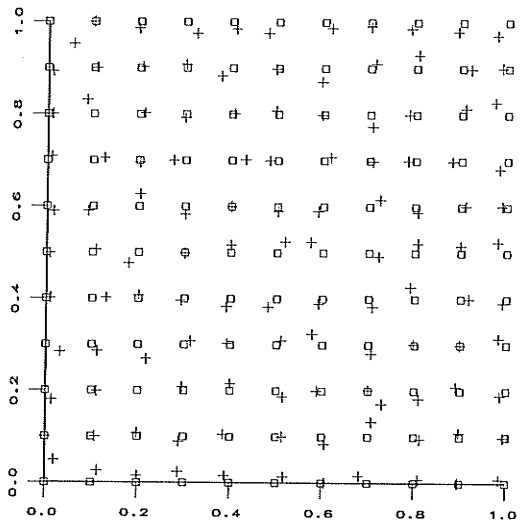


FIGURE 3.9. Results of the GCS Algorithm Trained on the 2-D Array Data Set.

However, eventually this additional training will give rise to the overtraining phenomena mention in the previous subsection of the report. Alternatively, we may attempt to improve the interior representation by scheduling reductions in the learning parameters - especially the neighbor cells learning parameter. Such an approach reduces the interaction between cells during the latter epochs of training, thus preventing the cells from being 'pulled in the wrong direction'.

The GCS's improved ability to represent the boundaries of the input distribution is directly associated to the dynamic nature of the neighborhoods within the GCS. Since those cells being used to represent the outmost data points have the greatest error value, it is in these regions where the majority of new cells will be added to the structure. Any new cell will act as a buffer from the effects of the interior nodes, and allow the exterior cells to improve their representations of the edges of the input distribution. It was found that these newest cells were often removed from the structure for they invariably failed to become winner nodes. These cells were mainly used for their buffering capacity.

Here again, the overtraining phenomena is eventually unavoidable. For as the GCS's cell count reaches its limit, the algorithm is unable to add new cells to act as buffers. If too much additional training is allowed, the cells will again tend towards the mean of the input set.

The RMS reconstruction error of the GCS's representation of the data set at the end of each epoch is presented in Figure 3.10. We see that discounting the effects of the SOM's conscience mechanism, the shapes of this curve and the one in Figure 3.4 are similar. That is, they both have a relatively short coarse tuning phase and longer fine tuning phase.

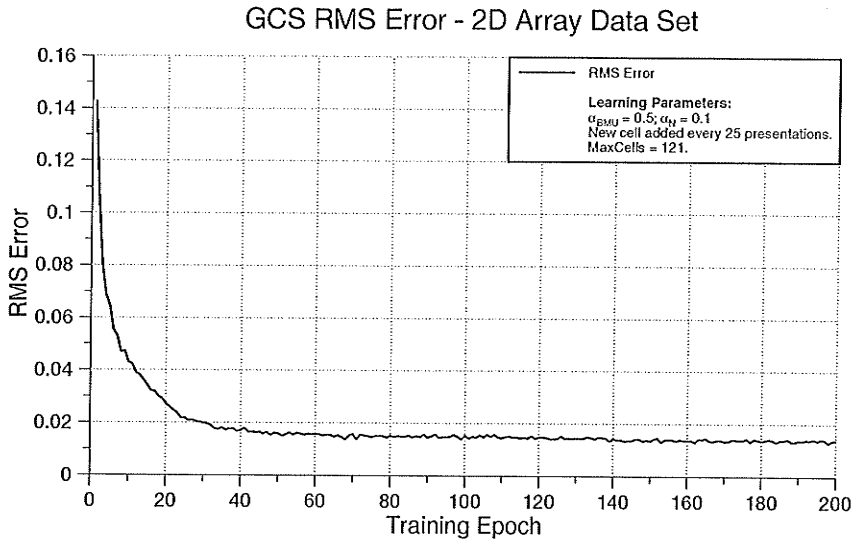
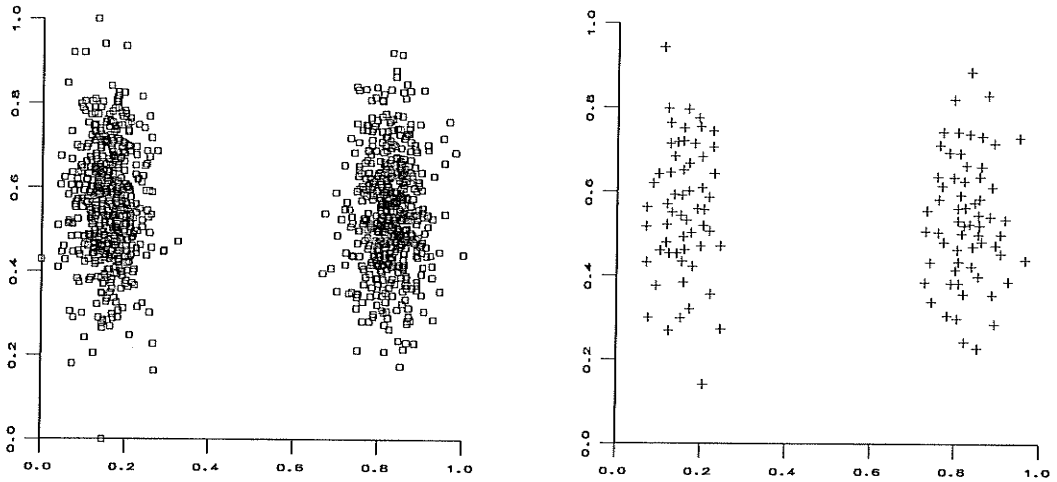


FIGURE 3.10. GCS RMS Reconstruction Error for the 2-D Array Data Set.

Due to it's improved ability to represent the boundaries of the input distribution, the GCS ended up with a better overall representation of the entire data set. The GCS had a final RMS reconstruction error of 0.013781, in comparison to the SOM's final RMS reconstruction error of 0.25491.

Figure 3.11(b) shows the final results of a GCS which was trained on the data set originally presented in Figure 3.5(a). This figure may be compared directly to the results of a equivalent sized SOM which was also trained for 20 epochs. Here the training set consists of 1000 points, and the GCS's cell count is again limited to 121 nodes. The learning parameters have been held constant at those values used in the previous example.



(a). Training Data. **(b). Results after 20 training epochs.**

FIGURE 3.11. GCS Training Using a 2 Class Normal Distribution Data Set.

In comparing the results of the GCS training, Figure 3.11(b), to those obtained for the SOM algorithm,Figure 3.5(b), it is evident that the GCS was able to better represent the shapes of the clusters within the data. Furthermore, it is also evident that none of the cells ended up trapped in the region between the two clusters. Each of the GCS's cells was used to represent the distribution of the data.

These improved results are again due to the GCS's dynamically defined neighborhoods around the cells. New cells are always added in the region where they are most needed,

thereby improving the representation of the data. Any cells which become trapped in a low input probability region will soon be removed.

Figure 3.12 shows the curve for the RMS reconstruction error after each epoch of the GCS training. A comparison between this curve and the one for the corresponding SOM, shown in Figure 3.6, indicates that the GCS converged faster and to a lower RMS reconstruction error, 0.011925, than the SOM which had a final RMS reconstruction error of 0.030939.

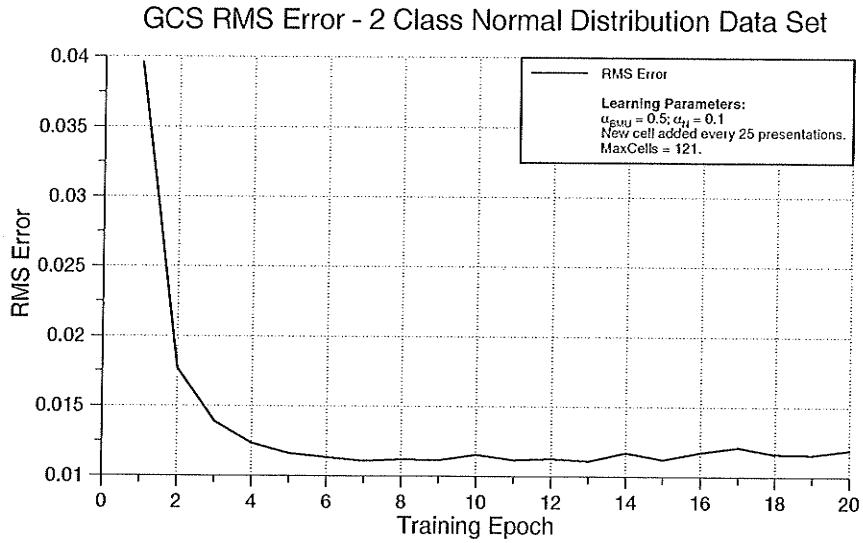


FIGURE 3.12. GCS Training Statistics for the 2 Class Normal Data Set.

3.2.3 GCS Versus SOM

The two examples presented in the previous subsection illustrate the superior representational capabilities of the GCS over the SOM. Due to its dynamical structure of adding and removing cells as required, the GCS better utilizes the available cells than the SOM, which invariably has a subset of its nodes trapped in the regions between clusters. Since the GCS begins with a small number of cells, it is able to quickly perform the preparatory coarse tuning, and thereby tends to converge faster than its SOM counterpart.

Finally, the dynamical nature of the GCS prevents the need for a conscience mechanism to be included in the algorithm.

3.3 HCM/FCM Algorithms

3.3.1 HCM/FCM Implementation

Figure 3.13 shows a simplified flowchart for the implementation of the HCM and FCM algorithms described in Section 2.3.1.2. The corresponding pseudo-code for these algorithms is given in Figure 3.14 on page 57. The CM algorithm requires the number of clusters to be found to be specified before execution. After the prototypes for each cluster are initialized, a DPM is created. A within groups sum of squared errors functional is defined using the DPM and the positions of the prototypes. The solution of the Lagrangian form of the error functional provides the update equations for the prototypes and DPM elements. Using these equations while iterating over the data set allows the CM algorithm to converge to a local minima for the error functional.

The algorithm's termination is based upon thresholds for the differences between successive DPMs or successive prototype positions. Since all the input data is considered simultaneously as the prototypes are updated, the CM algorithm typically terminates after significantly fewer epochs than the SOM or GCS algorithms. However, the additional overhead of the DPM maintenance results in greater memory requirements and longer execution times per epoch than the SOM or GCS. The parallel presentation of all inputs imposes another requirement on the CM algorithm. Contrary to the case for the SOM and GCS, where the data may be acquired and presented sequentially, CM requires all of the training data to be known and stored prior to the commencement of training.

Any metric may be used to define the distance measure used by the algorithm. For these examples, a simple Euclidean distance was used. However, the implementation also

allowed for Minkowski and generalized Hamming distances to be used. The choice of metric was not found to have significant effects on the rate of convergence for the algorithm, but may exhibit an effect on the final classification results.

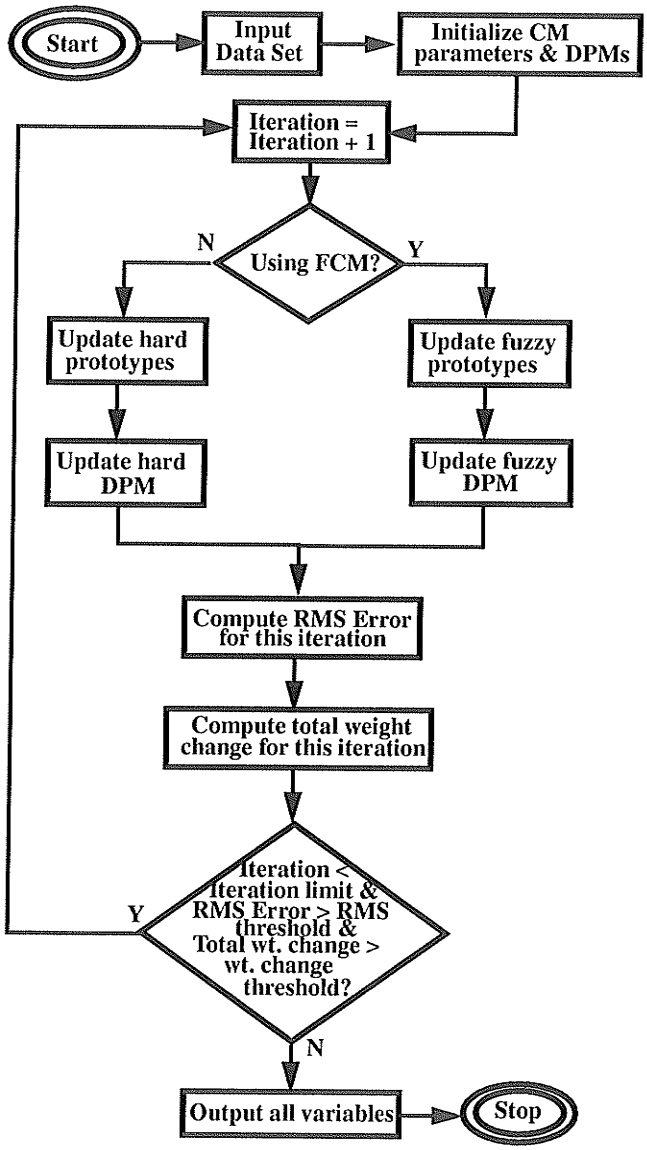


FIGURE 3.13. Flowchart for the CM algorithm.

Begin {CM}

Input p, n, $\boldsymbol{X} = \{\boldsymbol{x}_i \in \Re^p : i = 1, 2, ..., n\}$;
Input c, m, ε, $MaxIter$, $FuzzyFlag$, εRMS;
$t \leftarrow 0$;
Initialize prototypes $\boldsymbol{V}(0) = \{\boldsymbol{v}_i \in \Re^p : i = 1, 2, ..., c\}$;
If $FuzzyFlag$ **then**

$$u_{ik}(0) \leftarrow \left(\sum_{j=1}^{c} (d_{ik}^2 / d_{jk}^2)^{1/(m-1)} \right)^{-1};$$

Else

$$u_{ik}(0) \leftarrow \left\{ \begin{array}{l} 1 \text{ if } d_{ik}(0) = \min\limits_{j} d_{jk}(0) \\ \\ 0 \text{ otherwise} \end{array} \right\};$$

Loop

$$\boldsymbol{v}_i(t) \leftarrow \left(\sum_{k=1}^{p} (u_{ik})^m \boldsymbol{x}_k \right) / \sum_{k=1}^{p} (u_{ik})^m;$$

If $FuzzyFlag$ **then**

$$u_{ik}(t) \leftarrow \left(\sum_{j=1}^{c} (d_{ik}^2 / d_{jk}^2)^{1/(m-1)} \right)^{-1};$$

Else

$$u_{ik}(t) \leftarrow \left\{ \begin{array}{l} 1 \text{ if } d_{ik}(t) = \min\limits_{j} d_{jk}(t) \\ \\ 0 \text{ otherwise} \end{array} \right\};$$

$MaxError \leftarrow \max\limits_{i,j} [|u_{ij}(t) - u_{ij}(t-1)|]$;
$RMS \leftarrow CalcRMS(\boldsymbol{X}, \boldsymbol{V})$; $t \leftarrow t+1$;
Loop While $((t \le MaxIter)$ **and** $(MaxError \ge \varepsilon)$ **and** $(RMS \ge \varepsilon RMS))$;
$OutputResults(\boldsymbol{V})$;

End. {CM}

FIGURE 3.14. Pseudo-code for the CM algorithm.

3.3.2 Illustrative Examples

A new data set, consisting of four clusters was created to test the HCM and FCM algorithms. This data is shown in Figure 3.15. The clusters are very well separated in the input space. However, each cluster is of a different size and density, and consist of 400, 300, 200, and 100 patterns respectively. If the clusters were too close to each other, we may expect the algorithm to 'abandon' the smaller clusters in favor of better representing

the larger ones. Such action may be explained by the nature of the within groups sum of squared errors functional which is being minimized. Since the example is intended to illustrate the convergence properties of the algorithms as well as their ability to properly classify the data, the clusters were constructed with a large degree of separation.

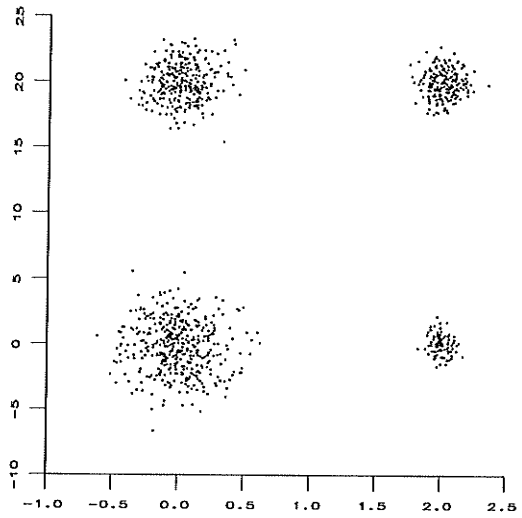


FIGURE 3.15. Training Data.

Indeed, the HCM and FCM algorithms were both able to properly classify all of the data by placing the prototypes at approximately the centroid for their respective clusters. The plots for the RMS reconstruction errors of the training are presented in Figure 3.16 and 3.17 for the HCM and FCM algorithms respectively. Both algorithms were allowed to run for a maximum of 30 epochs. Instead of running to its iteration limit, the HCM algorithm terminated after 16 epochs because the DPM exhibited no change from the previous iteration. The FCM training terminated after 15 epochs when the greatest change for any element in the DPM dropped below a threshold of 0.00001. The algorithms' property of convergence to a local minima is illustrated by the fact that although they each correctly classify every pattern, they terminated with different RMS reconstruction error values.

The HCM, with a final RMS reconstruction error value of 5.582598, has placed its nodes suboptimally.

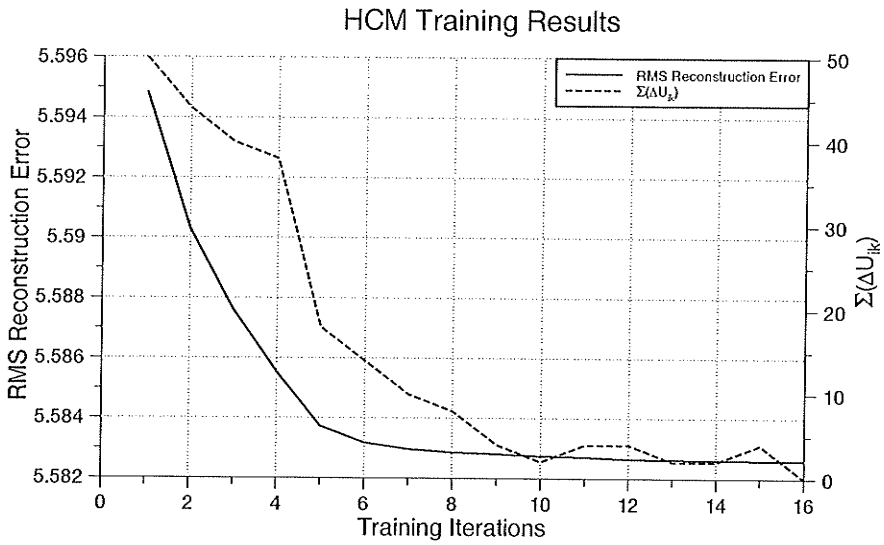


FIGURE 3.16. HCM Training Results.

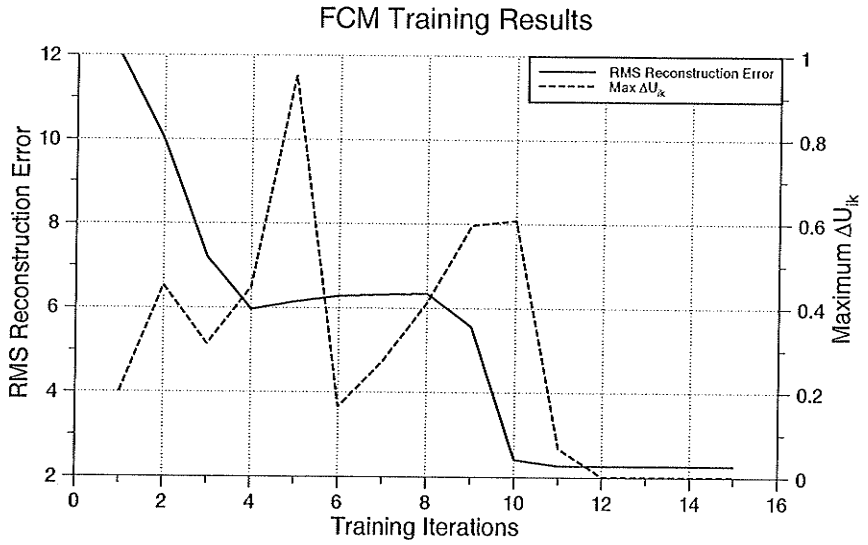


FIGURE 3.17. FCM Training Results.

Comparing the two curves, we see that the HCM algorithm is almost monotonically decreasing, while the FCM curve indicates some initial trouble with improving the classification at each iteration. In both cases, the prototypes were initialized to positions near the mean of the data set. Since the DPM for the FCM algorithm contains graded memberships, its elements are initialized to values near $\frac{1}{c}$. The greater the FCM's fuzzification parameter, m, the more these elements tend to remain at $\frac{1}{c}$. Thus, the next time the prototypes are updated using Equation 2.16, this averaging effect will prevent them from moving very far. This effect is illustrated by the dotted line in Figure 3.17 which shows the value of the greatest change for a DPM element during an epoch.

3.4 FKCN Algorithm

3.4.1 FKCN Implementation

Figure 3.18 shows a simplified flowchart for the implementation of the FKCN algorithm described in Section 2.3.2.4. The corresponding pseudo-code for the implementation is given in Figure 3.19 on page 62. Although the FKCN is a hybrid of the FCM and SOM algorithms, it has most of its features from the former. The only attribute inherited from its SOM ancestor is the weight update Equation 2.27, which is a modified version of the one used by the FCM. Thus, like FCM, the FKCN algorithm also requires the number of clusters to be found to be specified before execution. Data structures for the algorithm are also created and maintained as for the FCM. Again, the algorithm's termination is based upon thresholds for the differences between successive DPMs or successive prototype positions. The implementation also allowed for a variety of choices in the metric used to define distance measures.

One of the FKCN's innovative features is that the fuzzification parameter, m_t, is linearly decreased towards a value of 1.0. This reduction action effectively implements

neighborhoods about the prototypes. As the value of m_t is decreased, these neighborhoods are effectively reduced. In addition to having all of the operations per epoch that are found in the FCM algorithm, the FKCN is required to perform exponentiation of each element of the DPM to the power of the m_t prior to updating the prototypes. Therefore each epoch of the FKCN will take longer to execute than the same sized FCM.

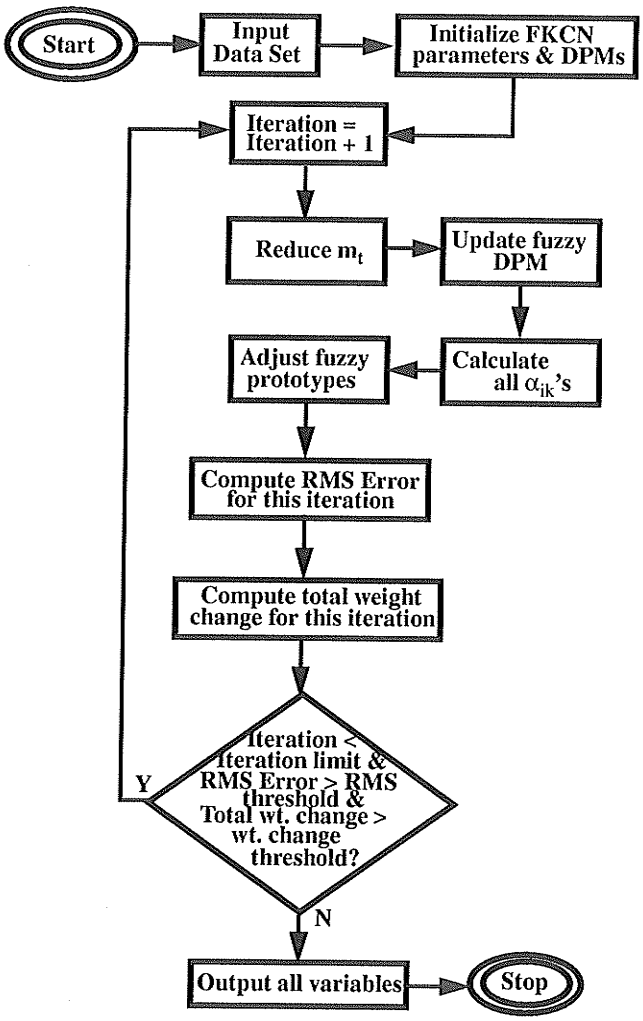


FIGURE 3.18. Flowchart for the FKCN Algorithm.

Begin {FKCN}

Input p, n, $\boldsymbol{X} = \{\boldsymbol{x}_i \in \Re^p : i = 1, 2, ..., n\}$;
Input c, $m0$, $MaxIter$, $MetricType$, εRMS, $\varepsilon Weights$;
$DeltaM \leftarrow (m0 - 1)/MaxIter$;
Initialize prototypes $\boldsymbol{V}(0) = \{\boldsymbol{v}_i \in \Re^p : i = 1, 2, ..., c\}$;
$t \leftarrow 0$;
Loop
$m(t) \leftarrow m0 - DeltaM \times t$;
$m \leftarrow 1/(m(t) - 1)$;

$$u_{ik}(t) \leftarrow \left(\sum_{j=1}^{c} (d_{ik}^2 / d_{jk}^2)^{m} \right)^{-1} ;$$

$$\alpha_{ik}(t) \leftarrow (u_{ik}(t))^{m(t)} ;$$

$$\boldsymbol{v}_i(t) \leftarrow \boldsymbol{v}_i(t-1) + \left[\sum_{k=1}^{n} \alpha_{ik}(t) \, (\boldsymbol{x}_k - \boldsymbol{v}_i(t-1)) \right] / \sum_{k=1}^{n} \alpha_{ik}(t) ;$$

$DeltaWeights \leftarrow CalcDeltaWeights(\boldsymbol{V})$;
$RMS \leftarrow CalcRMS(\boldsymbol{X}, \boldsymbol{V})$; $t \leftarrow t + 1$;
Loop While
$((t \le MaxIter)$ **and** $(DeltaWeights \ge \varepsilon Weights)$ **and** $(RMS \ge \varepsilon RMS))$;
$OutputResults(\boldsymbol{V})$;

End. {FKCN}

FIGURE 3.19. Pseudo-code for the FKCN algorithm.

3.4.2 Illustrative Examples

Figure 3.20 shows the training results for a 4 node FKCN which was trained using the data set provided in Figure 3.15. In this example, the scales for the plots are different from those of the FCM because the data set was normalized. The training parameters used are the same as those specified for the FCM example in the previous sub-section of the report. In this example, the algorithm terminated after 14 epochs due to the threshold on the change in the positions of the prototypes for successive epochs. As expected, the FKCN was able to place the prototypes in order to properly classify all of the input patterns. The shape of the RMS reconstruction error curve is similar to that found in the case of the FCM, i.e., approximately concave monotonically decreasing. However, we note that unlike the FCM, the FKCN seems to steadily approach convergence with each epoch. This trait is due to the difference in the weight update equation. Here, the algorithm updates its

prototypes by an adjustment of an averaged distance to its previous value instead of completely re-determining the new positions from the DPM. This results in a more steady motion of the prototypes within the search space.

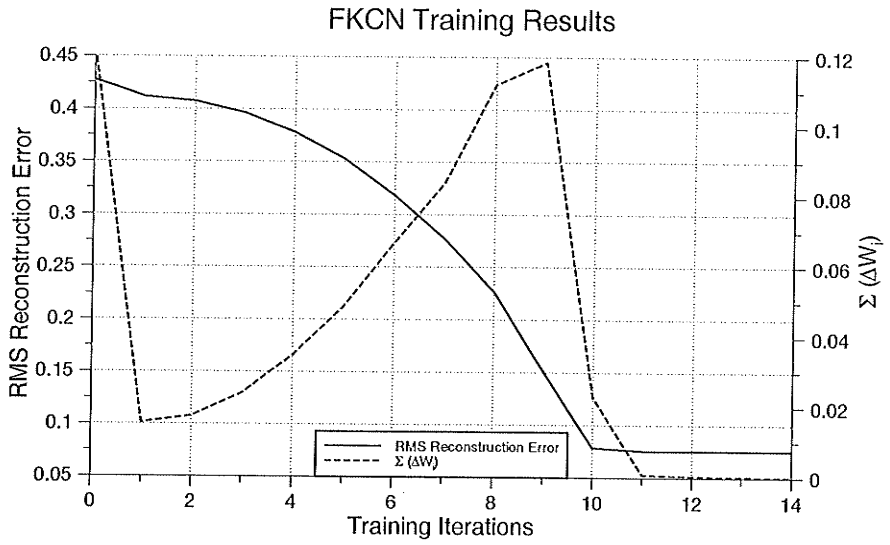


FIGURE 3.20. FKCN Training Results.

To achieve a better RMS reconstruction error for the training, the FKCN was once again trained with the previous data set. This time, the number of nodes to be used was increased to 10. All other parameters were held constant at their previous values. The final positions of the prototypes are shown in Figure 3.21 and the corresponding training results are presented in Figure 3.22. Coincidentally, this training example also terminated after 14 epochs. As expected, the final RMS reconstruction error of the representation is an improvement over the results obtained for the 4 node FKCN.

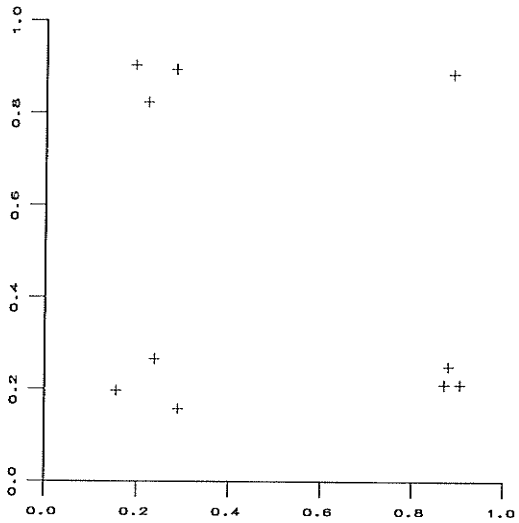


FIGURE 3.21. Final Results for a 10 Node FKCN.

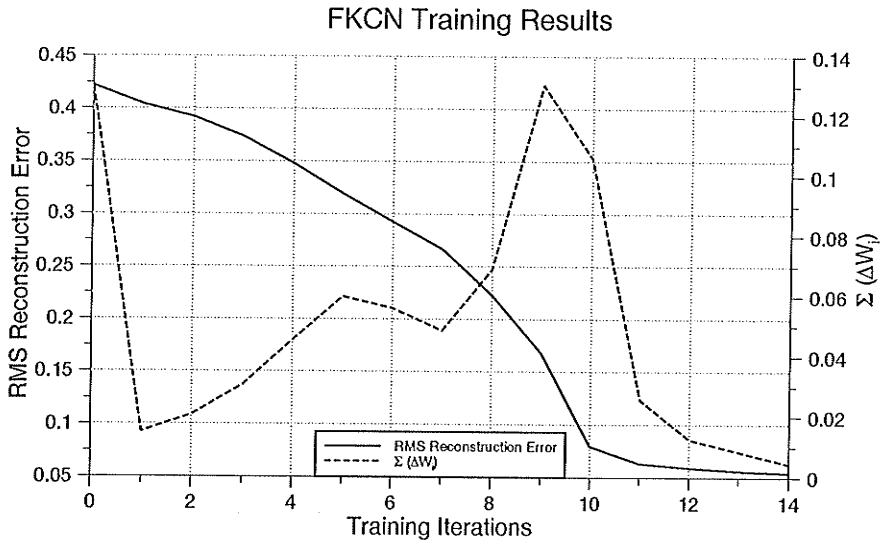


FIGURE 3.22. Training Results for the 10 Node FKCN.

This example also well illustrates the local minima convergence of the algorithm. Since there were more nodes provided than there are clusters in the data, the FKCN will end up placing these extra nodes so as to minimize its error functional. The variable size and density of the clusters in the data dictates that the larger clusters 'claim' more nodes than the smaller ones for their representation. Such is not the case for this example. The FKCN ended up allocating 3 nodes to the smallest cluster when one of the remaining larger clusters only received 1 node for its representation.

Chapter 4

Cluster Validity

One of the challenges in unsupervised pattern recognition is to accurately decide on the number of classes in the data. Since one typically has no information about the problem except for a set of input coordinates, this essential task is not trivially solved. The problem is further complicated by the nature of real world complexities of the input space. The simplest tasks will have compact and well-separated clusters. Typical problems will contain overlapping clusters which create regions of classification uncertainty. In the real world, clusters will have variable densities and covariances. Therefore, separate clusters would be of varying size, shape and orientation. Thus, a large cluster may include points which are closer to another cluster's centroid even though they are well-separated. Very small clusters may be overlooked if they do not contain enough points to justify receiving a centroid of their own. The problem is further complicated in situations where portions of the training set are missing or if the data is noisy. Additional problems are encountered when the clusters to be recognized have multimodal distributions.

As stated in Section 2.3 of the report, there are two approaches to determining valid clustering. The first category uses a single prototype per cluster. Then *cluster validity indexes* are evaluated to measure the propriety of the data partition. A vector quantized representation of the data is used in the second category. This approach requires a post-processing stage to appropriately cluster the numerous prototypes. Both of these methods are presented in the following two sections.

4.1 Single Prototype per Class

An valid data partition will contain only as many prototypes as there are classes in the data and correctly assign class membership for each input pattern. When using conventional unsupervised learning algorithms for pattern recognition, we must initially make an assumption of the number of classes present in the data. After the DPM has been constructed and optimized, some method of evaluation is required to determine the appropriateness of the assumed prototype count. We now present several interesting candidates for such cluster validity indexes.

4.1.1 Cluster Validity Indexes

In his book, Bezdek [4] describes among others several suggestions of functionals for use as cluster validity indexes: degree of separation, partition coefficient, partition entropy, normalized partition entropy, and separation coefficient. The degree of separation, partition coefficient, partition entropy, normalized partition entropy functionals all attempt to evaluate a particular clustering assignment based solely on the corresponding DPM.

Degree of Separation: (maximize)

$$Z(\boldsymbol{U};c) = 1 - \left[\bigcup_{k=1}^{n} \left(\bigcap_{i=1}^{c} u_{ik} \right) \right], \forall \boldsymbol{U} \in M_{fc}. \tag{4.1}$$

- $Z(\boldsymbol{U};c) = 1 \Leftrightarrow \boldsymbol{U} \in \boldsymbol{M}_c$

$Z = 1$ if $\boldsymbol{U} \in \boldsymbol{M}_c$, so it is not useful when using hard logic to partition the data. Z is too 'coarse' to be useful in evaluating fuzzy partitions for it only measures the largest minimum over the columns of $\boldsymbol{U}$. In such a case, a particular partition containing only one column having equimembership values could have the same degree of separation as another partition in which all of the columns contained equimembership values.

Partition Coefficient: (maximize)

$$F(\boldsymbol{U};c) = \sum_{k=1}^{n}\sum_{i=1}^{c}(u_{ik})^2/n. \tag{4.2}$$

- $1/c \le F(\boldsymbol{U};c) \le 1$
- $F(\boldsymbol{U};c) = 1 \Leftrightarrow \boldsymbol{U} \in \boldsymbol{M}_c$
- $F(\boldsymbol{U};c) = 1/c \Leftrightarrow \boldsymbol{U} = [1/c]$

F suggests that good fuzzy partitions will tend to be 'crisp'. Such an assumption is not necessarily the case because real world data typically contains significant areas of cluster overlap which should be reflected by fuzzy columns in the partition matrix. F is also typically monotonically decreasing with an increase in the number of clusters to be found. Therefore, as a cluster validity index, it is biased towards partitions having a smaller number of cluster prototypes.

The information-theoretic concept of entropy is often used in pattern recognition. Since fuzzy membership values may be likened to probabilities for choosing classes, a partition entropy index may be defined as follows:

Partition Entropy: (minimize)

$$H(\boldsymbol{U};c) = -\sum_{k=1}^{n}\sum_{i=1}^{c}u_{ik}\log_a(u_{ik})/n \tag{4.3}$$

where $a \in (1,\infty)$, $u_{ik}\log_a(u_{ik}) \equiv 0$ whenever $u_{ik} = 0$.

- $0 \le H(\boldsymbol{U};c) \le \log_a(c)$
- $H(\boldsymbol{U};c) = 0 \Leftrightarrow \boldsymbol{U} \in \boldsymbol{M}_c$
- $H(\boldsymbol{U};c) = \log_a(c) \Leftrightarrow \boldsymbol{U} = [1/c]$

Like F, H assumes that good partitions will tend to be crisp. Using H as a cluster validity index is also hampered by the measure's predisposition to validate partitions where the number of clusters c, approaches the number of data patterns p.

By normalizing the partition entropy, we may eliminate this preference for high cluster counts.

Normalized Partition Entropy: (minimize)

$$\hat{H}(\boldsymbol{U};c) = H(\boldsymbol{U};c) / [1 - (c/n)] \tag{4.4}$$

The F, H and $\hat{H}$ functionals all measure the fuzziness of the DPM $\boldsymbol{U}$ to indicate its validity. Since the clusters reside in the data set $\boldsymbol{X}$, Bezdek [4] suggests that the validity indexes should also depend on the elements of the algorithm which creates $\boldsymbol{U}$ from $\boldsymbol{X}$. One such index, the separation coefficient, is defined below.

Separation Coefficient: (maximize)

$$G(\boldsymbol{U};\boldsymbol{v};c;\boldsymbol{X};d) = 1 - \max_{i+1 \le j \le c} \{ \max_{1 \le i \le c-1} \{ (r_i + r_j)/c_{ij} \} \} \tag{4.5}$$

where $r_i = \max_{1 \le k \le n} \{u_{ik} d(\boldsymbol{x}_k, \boldsymbol{v}_i)\}$ for $1 \le i \le c$, and $c_{ij} = d(\boldsymbol{v}_i, \boldsymbol{v}_j)$ for $1 \le i \ne j \le c$.

- $G \in (-\infty, 1)$

In addition to $\boldsymbol{U}$ and c, G also depends on the positions of the prototypes, the data themselves and the distance function used. After executing a clustering algorithm, G is used to interpret the results as follows:

- $0 < G < 1 \Leftrightarrow$ {no pair of closed balls $\overline{B}(\boldsymbol{v}_i, r_i)$ intersect one another}
- $G = 0 \Leftrightarrow$ {the closest pairs of $\overline{B}(\boldsymbol{v}_i, r_i)$'s are exactly tangent}
- $G < 0 \Leftrightarrow$ {at least one pair of $\overline{B}(\boldsymbol{v}_i, r_i)$'s intersect one another}

Even though the separation coefficient gives conceptually appealing indications of the validity of the clustering, it does not completely overcome the deficiencies of an algorithm that does not take into account variable cluster density, size, shape and orientation. If the same metric is used to define distances with respect to each cluster, every prototype will have an identical region of attraction in its neigborhood. In order to optimize the cluster representation capabilities of all the prototypes, each must make use of its own distance metric. For the purpose of determining proper clustering, only those cluster validity indexes defined in the following subsection were implemented for this thesis.

4.1.2 UFP-ONC Algorithm

The probability density function for a normal variable having mean μ and standard deviation σ is given by Equation 4.6 below.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{4.6}$$

If the input set X (having cardinality n) is assumed to be drawn from a mixture of c p-variate normal distributions, i.e. $F(x) = \sum_{j=1}^{c} p_j g(x|j)$, the class conditional probability density of assigning random normal vector x to class j is found by extending Equation 4.6 to:

$$g(x|j) = \frac{e^{-\frac{1}{2}((x-\mu_j)^T \Sigma_j^{-1}(x-\mu_j))}}{(2\pi)^{p/2}(det\Sigma_j)^{1/2}} \tag{4.7}$$

where Σ_j is the p X p covariance matrix for class j.

In mixture theory, maximum likelihood estimation [4] is used to determine where to locate the decision boundary in order to minimize the misclassification rate when discerning between two overlapping random distributions.

Gath and Geva [17] introduced an algorithm for clustering data without the need for a priori assumptions of the number of subsets to be found. The *Unsupervised Fuzzy Partition - Optimal Number of Clusters (UFP-ONC)* algorithm is based on an extension of *Fuzzy Maximum Likelihood Estimation (FMLE)* [4], which defines 'exponential' distance measures to the prototypes. The exponential distance measures are used to determine $h(i|\boldsymbol{x}_j)$, the posterior probability of selecting cluster i given the input vector $\boldsymbol{x}_j$.

$$h(i|\boldsymbol{x}_j) = \sum_{k=1}^{c} \left(\frac{d_e^2(\boldsymbol{x}_j, \boldsymbol{v}_i)}{d_e^2(\boldsymbol{x}_j, \boldsymbol{v}_k)} \right)^{-1} \tag{4.8}$$

$$d_e^2(\boldsymbol{x}_j, \boldsymbol{v}_i) = \frac{[det(\boldsymbol{F}_i)]^{1/2}}{P_i} e^{\frac{1}{2}((\boldsymbol{x}_j - \boldsymbol{v}_i)^T \boldsymbol{F}_i^{-1} (\boldsymbol{x}_j - \boldsymbol{v}_i))} \tag{4.9}$$

where $\boldsymbol{F}_i$ is the p X p fuzzy covariance matrix of the ith cluster, and P_i is the a priori probability of selecting the ith cluster.

$$\boldsymbol{F}_i = \frac{\sum_{j=1}^{n} h(i|\boldsymbol{x}_j)\,(\boldsymbol{x}_j - \boldsymbol{v}_i)\,(\boldsymbol{x}_j - \boldsymbol{v}_i)^T}{\sum_{j=1}^{n} h(i|\boldsymbol{x}_j)} \tag{4.10}$$

$$P_i = \frac{1}{n} \sum_{j=1}^{n} h(i|\boldsymbol{x}_j) \tag{4.11}$$

The $h(i|\boldsymbol{x}_j)$ probabilities are very similar to the u_{ik} elements (Equation 2.20) from the FCM algorithm. A new centroid equation is defined by the following:

$$\boldsymbol{v}_i = \frac{\sum_{j=1}^{n} h(i|\boldsymbol{x}_j)^2 \boldsymbol{x}_j}{\sum_{j=1}^{n} h(i|\boldsymbol{x}_j)^2} \tag{4.12}$$

Together, equations (4.8) through (4.12) from the FMLE iterative algorithm to minimize an objective functional similar to the one defined by Equation 2.17 (with $m = 2$). Since the cluster prototypes each use a unique matrix norm induced distance metric (as determined from the DPM in the previous step), they have hyperellipsoidal regions of attraction with variable size, shape and orientation.

Due to its exponential distance metric, a centroid tends to have steep slopes at the boundaries of its region of attraction. Therefore, it quickly tends to assign high membership values to any pattern within its 'reach' and thus the FMLE algorithm converges to a local minimum. This is why Gath and Geva's UFP-ONC algorithm uses FCM as a first stage to predetermine a set of suitable initial centroid positions for its FMLE second stage.

Three new cluster validity indexes, *fuzzy hypervolume, average partition density* and *partition density* are defined below (keeping the notation from [17].) The fuzzy hypervolume is defined solely by the sum of square roots of the determinants of the fuzzy covariance matrices of the clusters. The density measures calculate the 'sum of central members' per unit hypervolume.

Fuzzy Hypervolume: (minimize)

$$F_{HV} = \sum_{i=1}^{c} [det(\boldsymbol{F}_i)]^{1/2} \tag{4.13}$$

Average partition density: (maximize)

$$D_{PA} = \frac{1}{c} \sum_{i=1}^{c} \frac{s_i}{[det(\boldsymbol{F}_i)]^{1/2}} \tag{4.14}$$

where $s_i = \sum_{j:\{x_j | [(x_j - v_i)\boldsymbol{F}_i^{-1}(x_j - v_i)] < 1\}} u_{ij}$.

Partition density: (maximize)

$$P_D = \frac{s}{F_{HV}} \tag{4.15}$$

where $s = \sum_{i,j:\ \{x_j \mid [\,(x_j - v_i) F_i^{-1} (x_j - v_i)\,] < 1\}} u_{ij}$.

In this manner, partitions producing compact and dense clusters are validated over those having low density.

As previously mentioned, the UFP-ONC algorithm consists of two stages which are sequentially executed with each increase in the assumed number of clusters. Beginning with a two class clustering, the FCM stage is executed normally. Upon its completion, the resultant DPM and cluster prototypes are used to initialize the FMLE stage. Once the FMLE algorithm is terminated, the F_{HV}, D_{PA}, and P_D cluster validity indexes are computed and stored. Then the number of clusters is incremented, and the training process is repeated. The algorithm is terminated based upon a maximum number of clusters to be evaluated and/or trends in the values of the cluster validity indexes.

The UFP-ONC algorithm does typically result in good data partitions having the correct number of clusters. However, such quality clustering comes at a high computational cost. In general, the calculation of the fuzzy partition matrices and their inverses is computationally intensive, and it grows rapidly as the dimensionality of the input space increases.

4.1.2.1 UFP-ONC Implementation

A simplified flowchart for the implementation of the UFP-ONC algorithm is shown in Figure 4.1. The algorithm's corresponding pseudo-code is given in Figure 4.2 on page 75. As previously indicated, the UFP-ONC algorithm is a two stage process. First the FCM

algorithm is executed for a given number of nodes. The results of this first stage are then used to initialize the FMLE second stage.

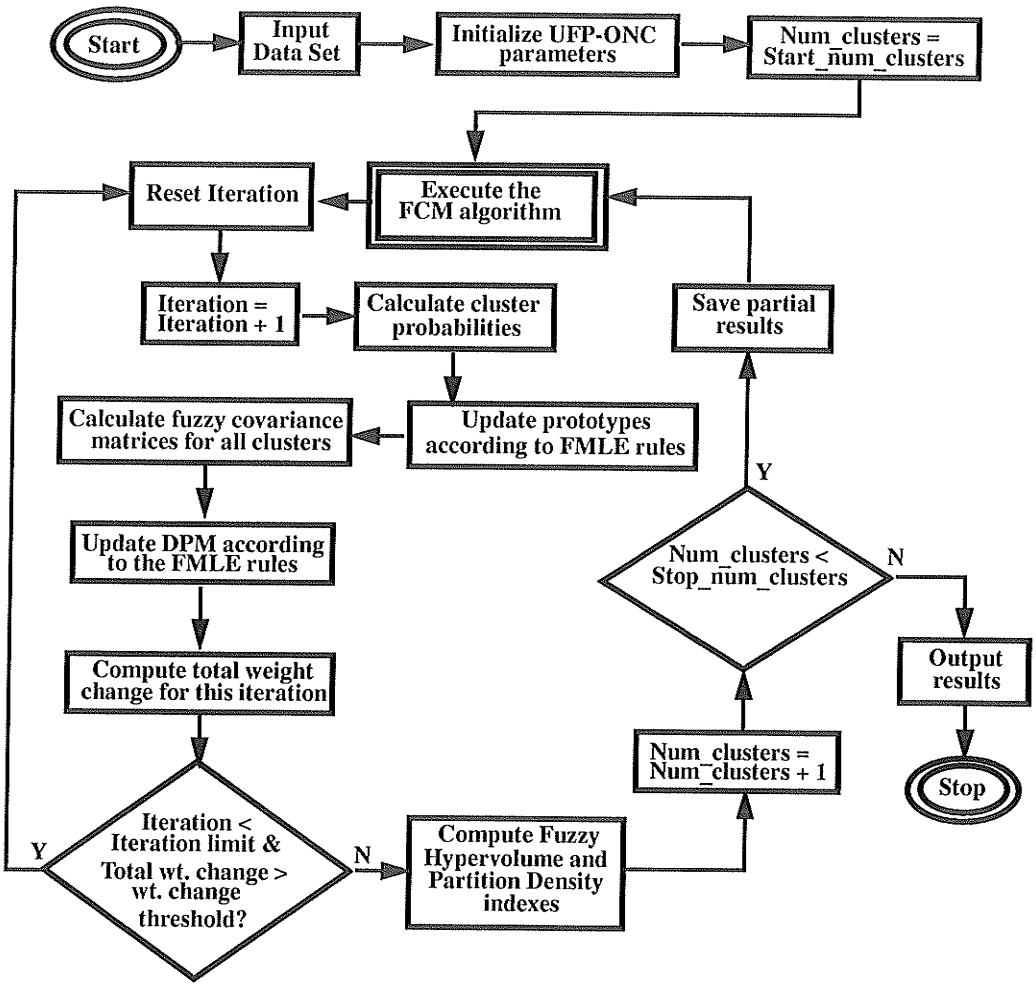


FIGURE 4.1. The flowchart for UFP-ONC algorithm.

The FMLE algorithm contains one loop in which the bulk of the computations are performed. At each iteration, the cluster probabilities and covariance matrices are computed. This information, together with the DPM from the previous step, is used to update the positions of the prototypes. The final step in each iteration is to create a new DPM using the new cluster prototypes and their statistics. After the FMLE has converged, the fuzzy hypervolume and partition density cluster validity indexes are calculated and

stored. The number of clusters is incremented and the FCM and FMLE stages are repeated.

Begin {UFP-ONC}

Input p, n, $X = \{x_i \in \Re^p : i = 1, 2, ..., n\}$;
Input c, m, ε, $MaxFcmIter$, $MaxFmleIter$;
Input $StartNumClusters$, $EndNumClusters$;
$c \leftarrow StartNumClusters$;
Loop {clusters}
Initialize prototypes $V(0) = \{v_i \in \Re^p : i = 1, 2, ..., c\}$;

$$u_{ik}(0) \leftarrow \left(\sum_{j=1}^{c} (d_{ik}^2 / d_{jk}^2)^{1/(m-1)} \right)^{-1} ; \quad t \leftarrow 0 ;$$

Loop {FCM}
$t \leftarrow t + 1$;

$$u_{ik}(t) \leftarrow \left(\sum_{j=1}^{c} (d_{ik}^2 / d_{jk}^2)^{1/(m-1)} \right)^{-1} ; \qquad v_i(t) \leftarrow \left(\sum_{k=1}^{p} (u_{ik})^m x_k \right) / \sum_{k=1}^{p} (u_{ik})^m \ ;$$

$$MaxError \leftarrow \max_{ij} \left[|u_{ij}(t) - u_{ij}(t-1)| \right] ;$$

Loop While $((t \le MaxFcmIter)$ **and** $(MaxError \ge \varepsilon))$; **{FCM}**

$$t \leftarrow 0 ; \ P_i \leftarrow \frac{1}{n} \sum_{j=1}^{n} u_{ij} ; \quad v_i(t) \leftarrow \left(\sum_{k=1}^{p} u_{ik} x_k \right) / \sum_{k=1}^{p} u_{ik} \ ;$$

$$F_i \leftarrow \left(\sum_{j=1}^{n} u_{ij} (x_j - v_i)(x_j - v_i)^T \right) / \sum_{j=1}^{n} u_{ij} ; \quad u_{ik}(t) \leftarrow (1/d_{eik}^2) / \left(\sum_{j=1}^{c} (1/d_{ejk}^2) \right) ;$$

Loop {FMLE}
$$t \leftarrow t + 1 ; \ P_i \leftarrow \frac{1}{n} \sum_{j=1}^{n} u_{ij} ;$$

$$v_i(t) \leftarrow \left(\sum_{k=1}^{p} u_{ik} x_k \right) / \sum_{k=1}^{p} u_{ik} \ ; \quad F_i \leftarrow \left(\sum_{j=1}^{n} u_{ij} (x_j - v_i)(x_j - v_i)^T \right) / \sum_{j=1}^{n} u_{ij} \ ;$$

$$u_{ik}(t) \leftarrow (1/d_{eik}^2) / \left(\sum_{j=1}^{c} (1/d_{ejk}^2) \right) ; MaxError \leftarrow \max_{i,j} \left[|u_{ij}(t) - u_{ij}(t-1)| \right] ;$$

Loop While $((t \le MaxFmleIter)$ **and** $(MaxError \ge \varepsilon))$; **{FMLE}**

$$F_{HV} \leftarrow \sum_{i=1}^{c} [det(F_i)]^{1/2} ; \quad s \leftarrow \sum_{i,j:\ \{x_j |\ [(x_j - v_i) F_i^{-1} (x_j - v_i)] < 1\}} u_{ij} ; \ P_D \leftarrow \frac{s}{F_{HV}} ;$$

Output F_{HV}, P_D; $c \leftarrow c + 1$;
Loop While $(c < EndNumClusters)$; **{clusters}**

End. {UFP-ONC}

FIGURE 4.2. Pseudo-code for the UFP-ONC algorithm.

4.1.2.2 Illustrative Example

Figure 4.3 below shows two of the four dimensions of Anderson's [2,13] Iris data set. It is evident that the first class is well-separated from the other two. However, the second and third classes contain a small region of overlap. Due to this overlap, we may expect the algorithm to have some difficulty in learning a proper classification scheme.

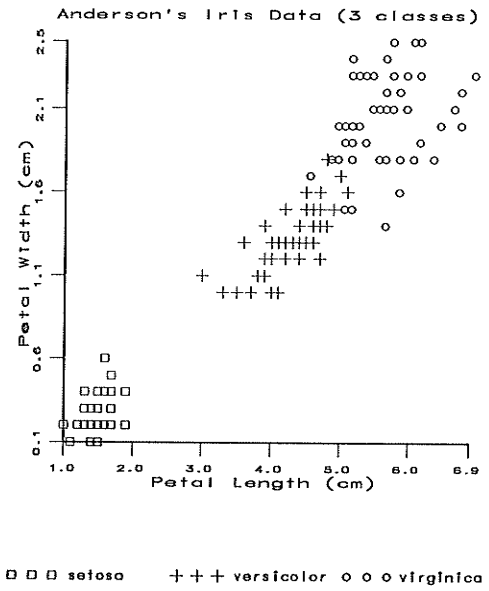


FIGURE 4.3. Scatter-plot of Two Dimensions From Anderson's [2] Iris Data.

The UFP-ONC algorithm was executed using the above data set. Limits of 25 iterations each were imposed on the FCM and FMLE stages. After both stages were completed, the fuzzy hypervolume and partition density indexes were computed and stored. The algorithm investigated the clustering performance over a number of prototypes range from two to five. Training results and their corresponding validity indexes for each cluster count are presented in Figure 4.4 and Figure 4.5 respectively. In this case, the minima for the fuzzy hypervolume index, and the maxima for the partition density index, both clearly

occur when 3 clusters are used. This results indicate that the UFP-ONC was able to suggest the correct number of classes in the data.

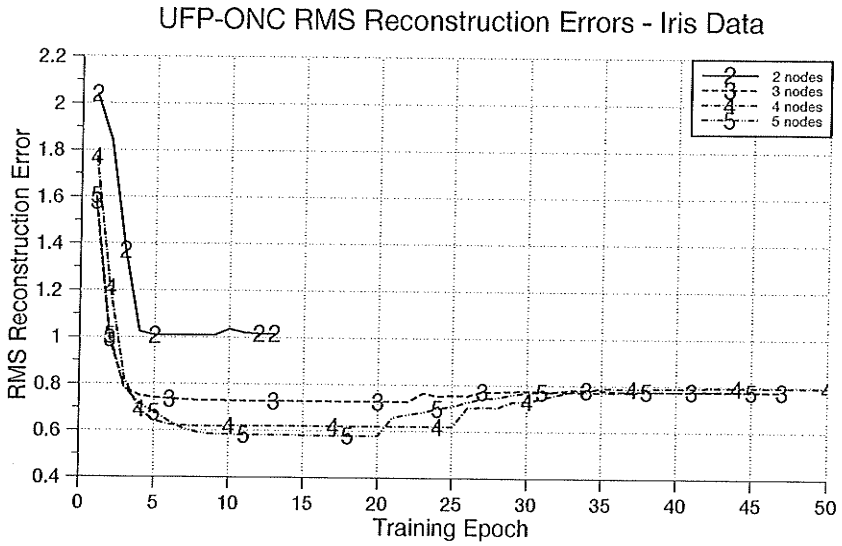


FIGURE 4.4. UFP-ONC Anderson's Iris Data training.

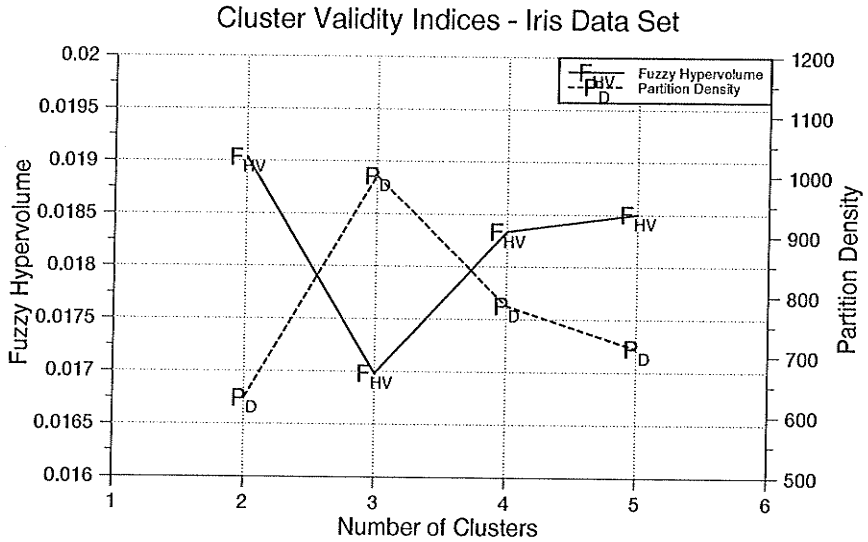


FIGURE 4.5. UFP-ONC Iris Data cluster validity indexes. (3 clusters indicated)

4.2 Multiple Prototypes per Class

4.2.1 SOM Interpretation

In their work with SOMs, Pedrycz and Card [38] introduced a method of linguistically interpreting the results. This approach is extendable to the entire family of vector quantization based algorithms. Since SOMs typically contain many more nodes than the actual number of classes in the data, interpretive post-processing is required to achieve logical classification. In essence, this is an automated extraction process which determines the rules for forming the nodes into groups. In the end, the user is presented with an indication of the number of rules, i.e. groups, that were found, a specification of the constituent nodes for each group and a structurally relative description for each class in the data set.

For this thesis, a graphical software package dubbed SOFM Interpreter, implementing this interpretive process, was created to operate under XWindows for Sun/Sparc workstations. Figure 4.6 displays the SOFM Interpreter's start-up window with the main menus.

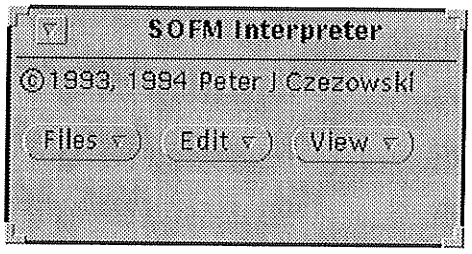


FIGURE 4.6. SOFM Interpreter Main Menus.

The process begins with the definition of linguistic variables or fuzzy labels introduced by Zadeh [47]. Figure 4.7 shows the program's fuzzy label input facility. Up to seven labels may be defined in any triangular, trapezoidal or arbitrary fashion. In this case, three labels have been created; small, medium and large.

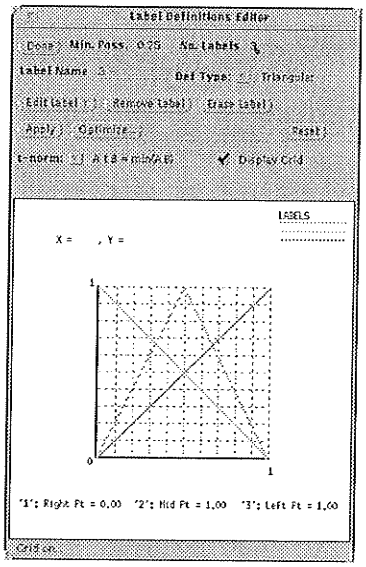


FIGURE 4.7. Fuzzy Label Definition Editor.

Since all inputs to the SOM are pre-normalized, the fuzzy variables are defined to operate over an input domain of $[0, 1]$. The output of the fuzzy labels also range from 0 (no membership) to 1 (complete membership). In Figure 4.7, the fuzzy labels have been defined as follows:

$$\begin{aligned} &\textit{Small: } small(x) = 1 - x. \\ &\textit{Medium: } medium(x) = 1 - \left| \frac{0.5 - x}{2} \right| . \\ &\textit{Large: } large(x) = x. \end{aligned}$$

The n-dimensional coordinates (weights) of the SOM's nodes are loaded from a file and are displayed on the screen. The SOMs nodes have been arranged in a 10 X 10 array. In this example, the SOM is trained using Anderson's Iris data [2,13]. The data set consists of 50 labelled examples from each of 3 classes. Although the class of each vector is not provided during the training, we may use this information to determine the topology of the transformations. The class for each input which the node was the winner is shown in Figure 4.8 below. We see that the nodes in the upper-right corner of the map ended up

representing the first class. The nodes more or less along the upper left - lower right diagonal of the map were the winners for the second class. This leaves the nodes of the bottom-left corner to represent the third and final class. Two nodes have been marked with an 'X' to indicate that they were the winners for inputs from more than one class. Additionally, we see that 26 nodes did not end up representing any of the inputs, as indicated by the 'blank' nodes.

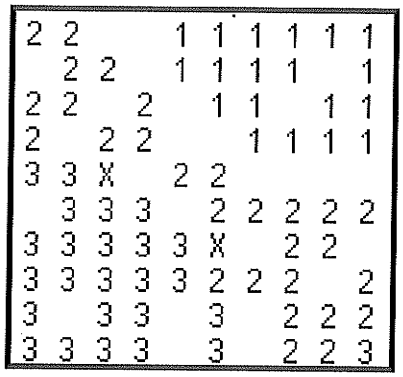


FIGURE 4.8. Distribution of the Classes in the Iris Data Set.

It is evident that the SOM has properly preserved the topology of the input space, for the winning nodes of the second class are 'between' those for the first and third classes. Nodes marked 'X' are to be expected given the fact that the second and third classes a small region in common. In accordance with the results shown in Section 3.1.3, a subset of 'blank' nodes is evident due to the region of separation between the first and second classes.

Figure 4.9 displays the weights of a SOM after it has been trained. Each weight map represents one of the data set's four dimensions. With this representation scheme, darker shades represent weight values closer to 1.0.

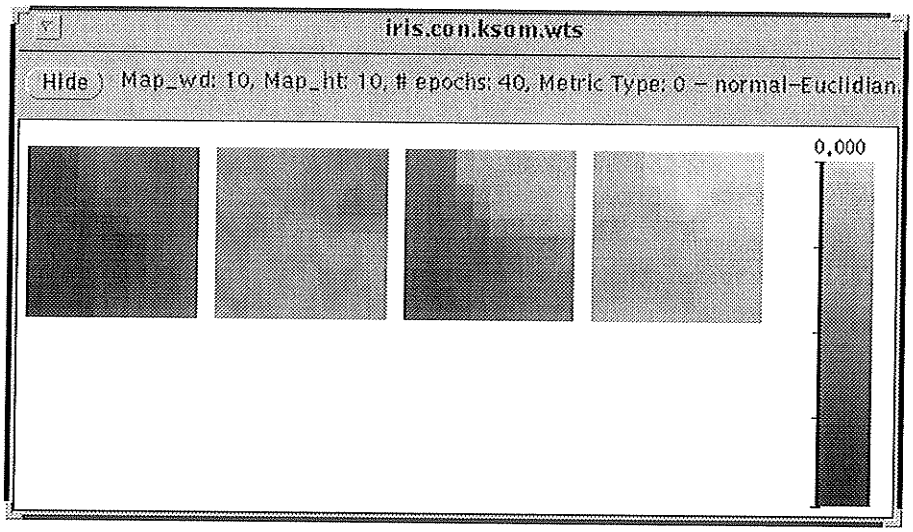


FIGURE 4.9. Displaying the SOM's weights (10 X 10 nodes).

The next step consists of applying each of the labels to each of the weight maps. Figure 4.10 shows the twelve linguistically transformed weight maps that were obtained in the example.

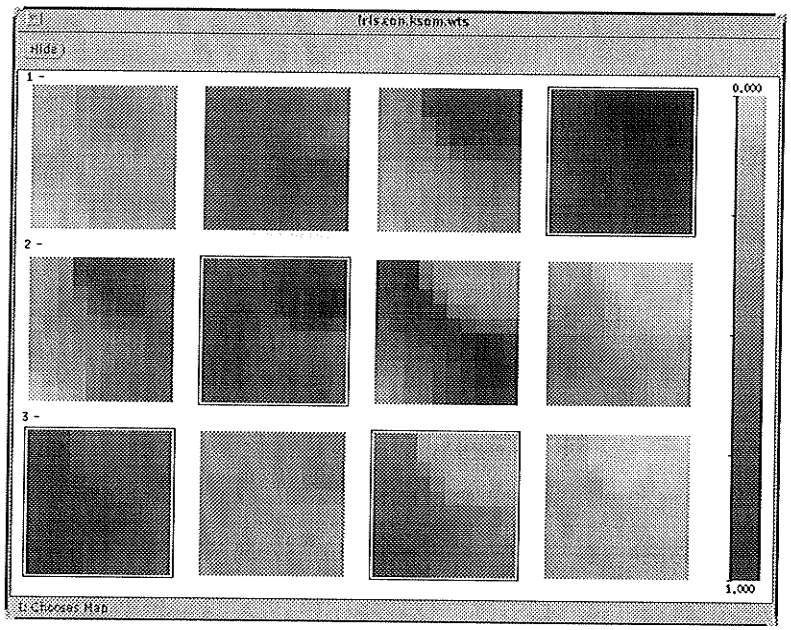


FIGURE 4.10. Weight maps transformed by linguistic variables.

Combinations of these transformations now quantize the descriptions possible for the weights in the map. In the figure above, the rows of maps (starting at the top) represent the application of the fuzzy labels for small, medium and large respectively. The shaded values in the maps display the grades of compatibility of the weights with their corresponding linguistic labels. Specifically, the darker the transformed weight's shade, the greater is its compatibility with the proposed label.

In order to describe a node in the map, one label for each dimension must be chosen. This fuzzy description D of a point in the input space is taken to be the logical intersection of its components:

$$D = A_{i1}(w_1) \cap A_{i2}(w_2) \cap \ldots A_{in}(w_n) \tag{4.16}$$

where A_{ij} is an element of the set of fuzzy labels,
w_i is the value of the i^{th} dimension weight for the node, and
n is the dimensionality of the input space.

In fuzzy decision theory, intersection is defined by any function from a family known as *t-norms* [39]. The SOFM Interpreter software allows the user to specify which t-norm to use from the following list:

$$AtB = min(A, B),$$
$$AtB = AB, \text{ or}$$
$$AtB = max(0, A + B - 1).$$

Possible descriptions for the 2-D SOM are obtained by extending Equation 4.16 to the form

$$D = [D_{j1,j2}] = A_{i1}(w_{j1,j2,1}) \cap A_{i2}(w_{j1,j2,2}) \cap \ldots A_{in}(w_{j1,j2,n}) \tag{4.17}$$

Each combination D may be referred to as a rule to describe the map and best represents a variable number of the SOMs nodes. The *possibility* (π_D) of D is defined as $\max_{j1,j2}(D_{j1,j2})$ which indicates the overall extent to which the description is compatible

with the map. A high value for π_D indicates good compatibility of the rule for at least some portion of the map.

The total number of distinct rules is given by A^n where A is the number of fuzzy labels available, and n is the problem's dimensionality. Using the example case, each of four dimensions being quantized to three fuzzy labels yields a total set of $3^4 = 81$ distinct rules $\{SSSS, SSSM, SSSL, \ldots, LLLL\}$. Depending on the distribution of the data, some (typically most) of the rules will not be supported. Determining which rules should be accepted or discarded becomes a task of generating each rule D and evaluating its support (π_D). The level of support for each rule is recorded and a threshold is allowed to vary in order to increase or reduce the number of rules which are accepted. The union of all the accepted rules is then taken to give a good representation of the entire map. However, such a union will typically result in many nodes being well represented by multiple rules. Such overlapping representation stems directly from the overlap in the definitions of the original fuzzy variables and causes the need for the application of another threshold.

Since each rule may be expressed as a series of α-*cuts* [39] which identify the nodes in the map having a grade of membership not less than α, an α-cut threshold is then used to minimize the rule overlap over all the nodes. Apart from reducing representational overlap, α threshold also increases the confidence level in the interpretation of the SOM.

This approach to grouping nodes is intuitively appealing. However as presented, the method is time and resource consuming, and grows exponentially with the increased dimensionality of the problem!

We now propose a refinement of the interpretive process which avoids an exhaustive search through the complete set of rules and minimizes the number of rules to be evaluated. The solution is to examine the input space only at those points which are

necessary; the positions of the nodes. Since t-norms are by definition monotonically nondecreasing, the best rule for the neighborhood about any node is given as the combination of best labels for each dimension of it's weights. Under such an assumption, the best rule to describe node $(j1, j2)$ of the SOM is then defined by Equation 4.18 below.

$$D^{opt}{}_{j1,j2} = \max_{A_{i1}} (A_{i1}(w_{j1,j2,1})) \cap \ldots \max_{A_{in}} (A_{in}(w_{j1,j2,n})) \tag{4.18}$$

The name of each distinct rule found is then saved in a list. Thus the maximum number of rules to generate and evaluate now becomes equal to the number of nodes in the SOM. Typically, this process will drastically reduce the number of rules requiring further examination. The SOM interpretation process may now continue as before.

Figure 4.11 depicts the possibilities of the six rules (as opposed to 81 previously) that were determined to be appropriate for further examination. By adjusting the possibility and α-cut thresholds, it was determined that the three most supported rules sufficiently described the SOM. This result indicates that three classes were found in the data. Figure 4.12 displays the chosen rules, indicating the extent to which each rule described the nodes of the map. Also indicated are the nodes which constitute each rule (after applying the α threshold). For the example case, the rules chosen were MMSS, LMMS and LMLS.

Figure 4.13 shows the extent to which the entire SOM is represented by the three chosen rules together. The number of nodes covered by the rules and the number of nodes being represented by overlapping rules are given as a percentage of the total number of nodes in the map. The three chosen rules were thresholded at an α value of 0.4. Ninety percent of the SOM's nodes were accounted for, with an overlap of seventeen percent. These results correspond well with the data since botanists label the iris flowers into three subspecies.

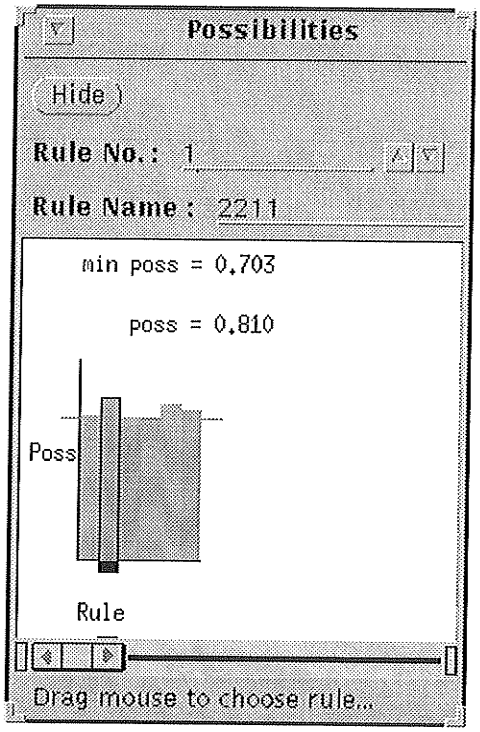


FIGURE 4.11. Rules to be further considered.

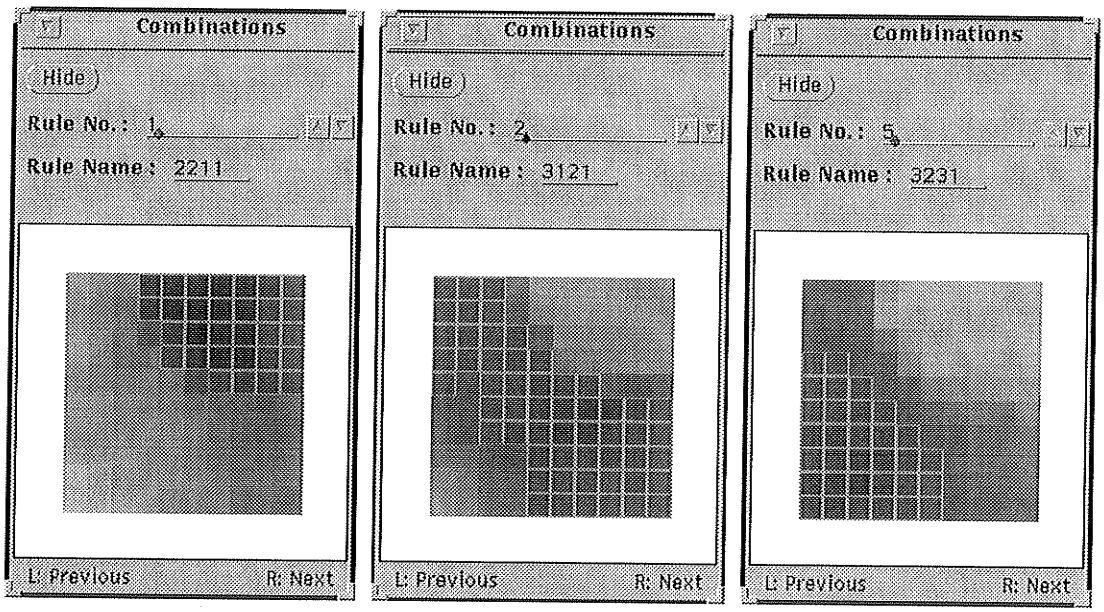


FIGURE 4.12. The chosen rules (three classes found).

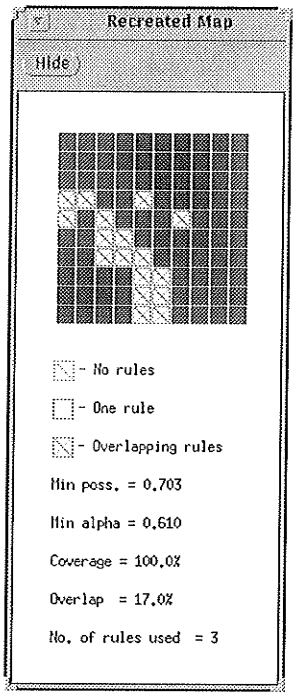


FIGURE 4.13. Reconstructed map.

Often, simple possibility level thresholding leads to suboptimal rule selection. Such a case typically occurs when many rules have approximately equal possibility values. Perhaps it would have been better to include a rule having lower possibility while excluding one which survived the threshold. For this purpose, two different rule selection mechanisms were included with the interpreter package: manual selection/deselection and a genetic algorithm optimization [18] routine.

Through mouse controls, the software allows the user to view each rule. With the manual rule select/deselect feature, the user is able to specify which rules should be used in the optimization as well as exercise control over the α threshold level. To further automate the advanced rule selection process, a simple genetic algorithm is engaged in an attempt to optimize a objective (fitness) function. This function is constructed as a weighted sum of

the SOM's node coverage, rule overlap, number of rules selected, minimum selected π and minimum selected α values.

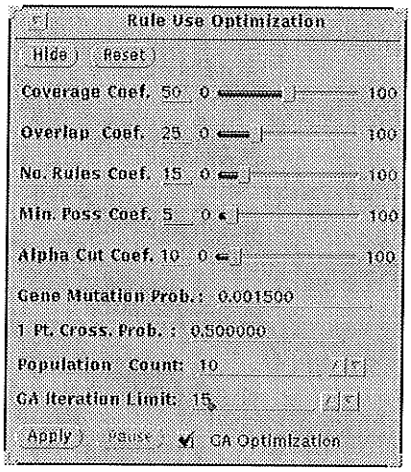


FIGURE 4.14. Controlling the Genetic Algorithm optimization.

Figure 4.14 above shows the software's genetic algorithm optimization facility with its slider controls for adjusting each of the fitness function's parameter weighting. The software makes use of user specified variables to control the chromosome population, generation limit and chromosome operations parameters.

Additional extensions to the rule extraction process would need to include a procedure for optimizing the fuzzy labels by tailoring them for the data set. One could also include a separate set of fuzzy labels for each dimension of the input. Such a step would further reduce the overlap of the rules for each class while increasing the overall confidence in the rules. It may not be feasible to increase the total node coverage. Especially if the data contains well separated clusters. This is because the local nodal interaction during the SOM training process will result in the placement of some nodes in areas of low input density. Such nodes will not be good representatives for any portion of the data. Therefore these nodes are not used by the SOM in its representation of the data and the rules to be

formed should not attempt to include descriptions for them. Even further optimization of the rule generation process may be achieved by allowing it to make use of 'don't care' conditions for the separate inputs.

4.2.2 GCS Interpretation

One effect of the GCS's continual addition and removal of cells is that eventually the structure becomes disjointed. Once a portion of the structure has split away from the remainder, it will never re-attach itself. These separate structures continue to grow and adjust themselves independently. Eventually, any well separated clusters in the data will be accurately represented by their own subsets of the GCS's cells.

A simple gathering process may now be used to determine how many of these subsets exist. The number of disjoint subsets in the GCS now gives a reasonable estimate of the number of classes existing in the data. However, in a real world data set, distinct classes are not always well separated. In fact, these classes often contain regions of overlap among them. Since the positions of the cells in the GCS provide a 'good' compact representation of the distribution of the data, we may use their weights as a new and substantially reduced data set with which to run the UFP-ONC algorithm for examining cluster validity indexes. With the assumption that the classes have multivariate gaussian distributions, we may simultaneously execute UFP-ONC algorithms on each of the separate subsets of the GCS. A revised estimate of the actual number of classes in the data is then obtained by summing the suggested number of classes within each subset of the GCS. This procedure will result in faster classification because of the great computational intensity that would be required by applying the UFP-ONC algorithm to the complete original data set. As a cautionary note, we state that a relatively large number of cells must be available for the GCS to permit the reasonable preservation of the distributions of the classes within the data. Otherwise, the UFP-ONC results may be erroneous.

Chapter 5

Prediction of Input Component Values Through Unsupervised Learning

Once a network has formed the final representation of a given data set, an interesting application of the results is to determine the predictive capabilities of its nodes. To accomplish this task, a subset of a test vector's inputs is presented to the network. Those dimensions of the network which correspond to the available inputs are then used to determine the distance to each node. A competition is then held, and the best matching node is declared the winner. The weights of this node, which correspond to the missing input dimensions are then outputted as predictions for the missing values.

5.1 A Constructed Example

To test this predictive ability, a new five dimensional data set was constructed. This set was divide into one set, consisting of 1296 patterns for training, and a second 256 pattern test set. None of the test patterns were present in the training set. These patterns were constructed according to the sum of exponentials equation show below:

$$x_1 = e^{-0.6x_2} + e^{-0.7x_3} + e^{-0.8x_4} + e^{-0.9x_5} \tag{5.1}$$

Where x_2 through x_5 are all independent variables.

Since any vector-quantizing compression of the data is lossy, to quantify the quality of the predictions, we will compare the results to the achievable optimal which is given by using the entire training set for the prediction process. The optimal results for the given data set are presented in Figure 5.1 which shows the expected, predicted and error values for each

of the test vectors. For this case, the RMS prediction error was found to the 0.053089 units.

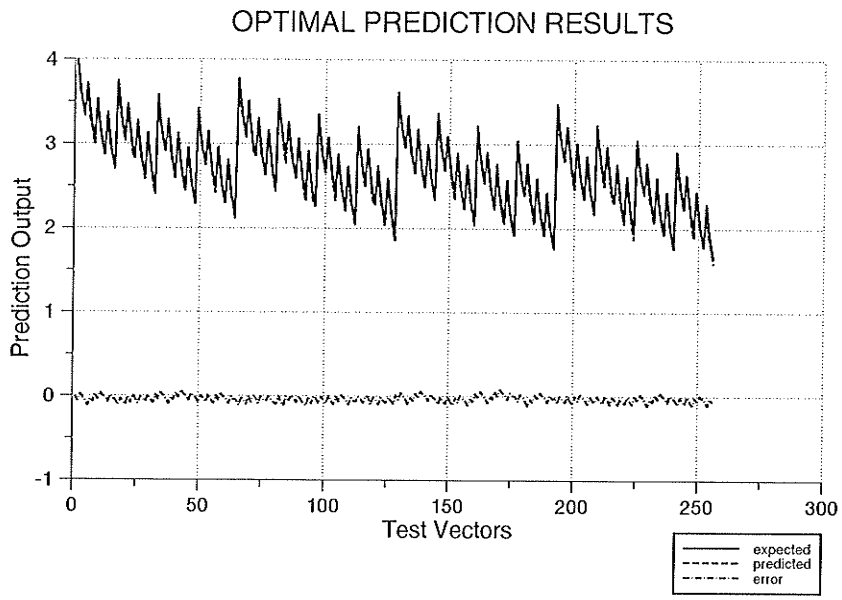


FIGURE 5.1. Optimal Prediction of the Dependent Component.

Next, the GCS algorithm was executed using the data, and the structure was allowed to grow to the size of 100 cells. These cells give a compression ratio of 12.96 : 1 for the data. Figure 5.2 displays the prediction results for this GCS. Although the RMS prediction error of 0.168888 units is 3.18 times the optimal, the network's ability to predict the dependent variable would be suitable in many applications. The prediction results may be improved by reducing the data compression ratio through the addition of more cells.

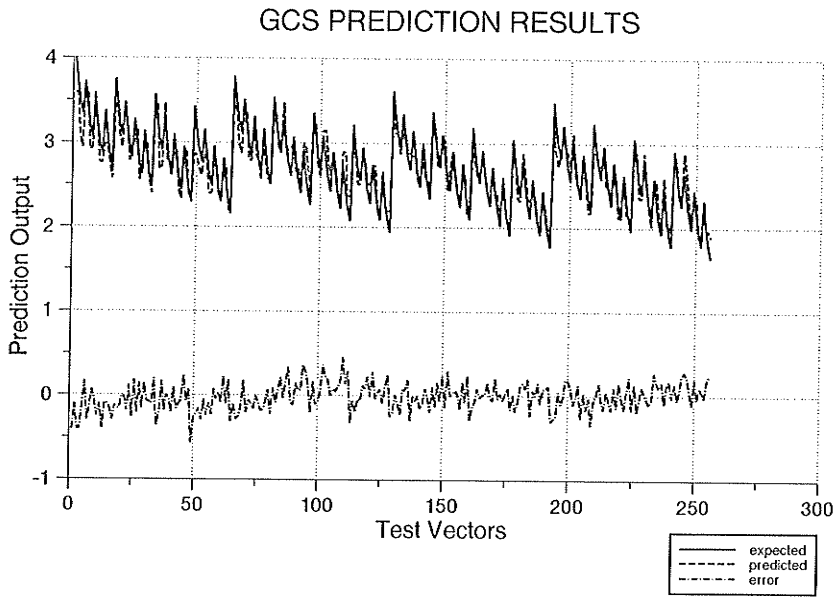


FIGURE 5.2. Predicting the Dependant Component Using a 100 Cell GCS.

5.2 Forecasting the Load on a Hydro Utility

Another example prediction application is presented in order to evaluate the results in a realistic setting. On any given day, a hydro-electric utility would benefit from a reliable estimate of the actual load at each hour of the day. Such information is useful for the appropriate scheduling of generator maintenance times and contracts for the sale of surplus power to other utilities. This load curve is essentially nonrepeating. However there typically are similarities in the shape of the curve for each day. Even though this curve is nonstationary in the long run, there are short periods for which an accurate forecast may be made. For this example the data set consists of 4872 training, and 1176 testing patterns. Each pattern is a 7 dimensional vector consisting of the gathered information for the hour, temperature on forecast day, wind speed on forecast day, wind chill on forecast day, load, temperature on previous day, and wind chill on previous day. Since all dimensions of the data have been normalized prior to training, the forecasting results units are normalized

megawatts. Figure 5.3 shows that the optimal forecasting results have an RMS prediction error of 0.028548 units.

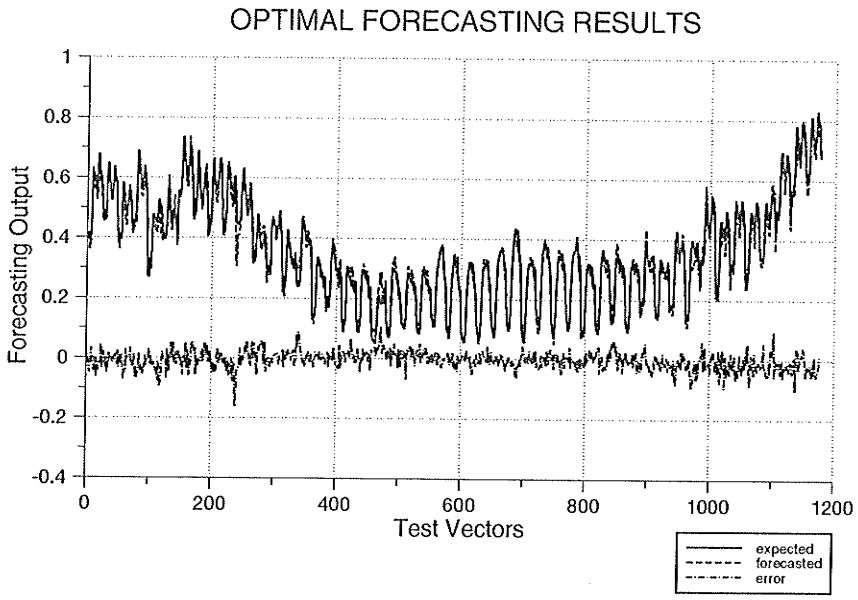


FIGURE 5.3. Optimal Load Forecasting.

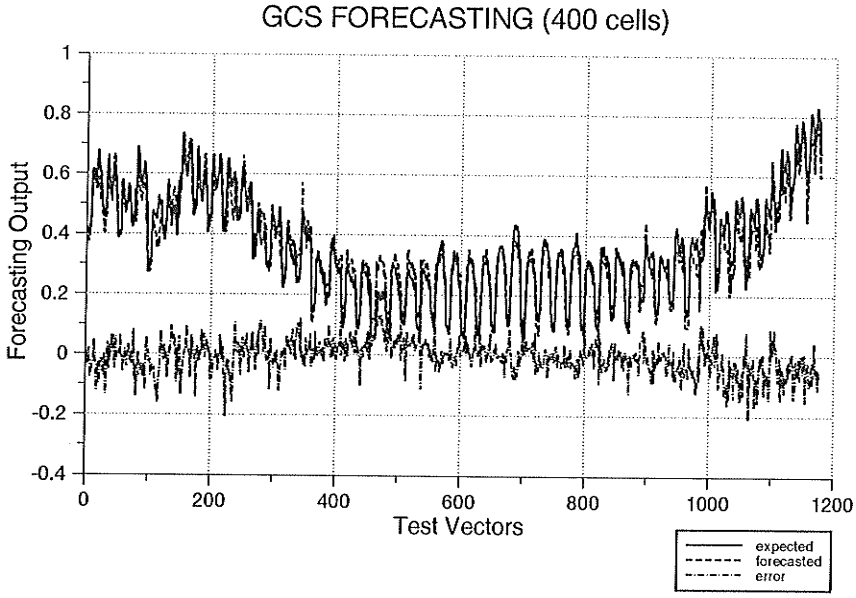


FIGURE 5.4. Results of Load Forecasting Using a 400 Cell GCS.

Figure 5.4 above shows that allowing a GCS trained on the data to grow to a size of 400 cells, results in a data compression ratio of 12.18 : 1, and an RMS prediction error of 0.053047 units, which is only 1.86 times the optimum.

For comparison, the forecasting results shown in the previous figure are evaluated against those for a 400 node SOM, shown in Figure 5.5. In this case, the SOM achieved a better RMS forecast error of 0.047667 units, which is 1.67 times the optimum.

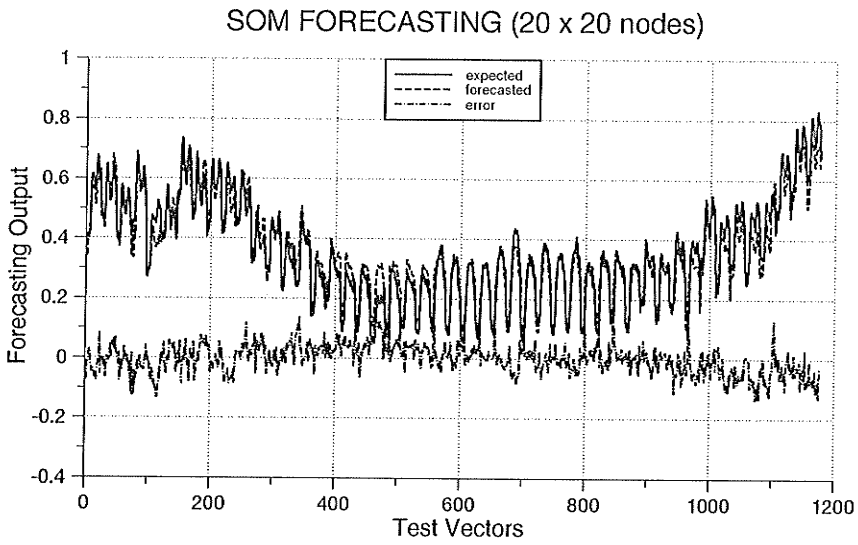


FIGURE 5.5. Results of Load Forecasting Using a 400 Node SOM.

Having the SOM out perform its counterpart GCS in this case is not especially surprising for we would intuitively expect any clusters in the load data to have much overlap. Therefore, a SOM would not waste a large proportion of its nodes on the representation of the regions of low input probability. However, the execution time for the SOM algorithm was 1.54 times that of the same sized GCS. In both these cases, the forecasting results rival those which are currently being obtained through a linear regression technique and are within the utility's error tolerance range.

Chapter 6

Conclusions and Recommendations

Although it is referred to as *unsupervised* learning, this description is not completely accurate. Some information, in the form of a metric to be used in defining a node's region of attraction, must be provided. We described how the choice of metric could be made to appropriately identify clusters of various shapes within the data. Since all unsupervised learning algorithms form a non-uniform vector quantization of the data, their representational capabilities are essentially equivalent. However, these algorithms do vary greatly in their complexities. Although the DPM based algorithms provide an immediate representation of the classes in the data, they are hampered by the additional burdens of the DPM maintenance and a requirement of prior knowledge of the number of classes to be found. The cluster validity indexes in the UFP-ONC algorithm provide an alternative to the latter problem, but this algorithm is further hampered the by additional cost of the numerous calculations of many covariance matrices. The interpretation of a trained GCS, as presented in Section 4.2.2, was the simplest method found to reliably classify the data. Moreover, due to its resistivity to boundary effects, the GCS showed the best ability to represent data along the edges of the input distribution.

Chapter 4 also presented an improvement to a previously reported procedure for interpreting the results of SOM training. This process no longer grows exponentially with an increase in the dimensionality of the problem space. Through a graphical computer program, the user is left with a suggestion for the proper classes in the problem, and the ability to easily adjust the rules to be used in order to search for better results.

Chapter 5 presented a previously unencountered application of these networks with respect to the prediction of dependent data. Results of the application of this procedure were found to be good for the prediction of stationary as well as non-stationary data.

Since these types of networks have shown promise in data compression, especially for audio and video signals, it is recommended that further work in this area be done in the direction of hardware implementations of such systems. Large scale implementations, containing many nodes, will prove to be useful in the transmission and storage of such data.

References

[1] Ackley, D.H., et al, "A learning algorithm for Boltzmann machines", *Cognitive Science,* 9, 147-169, 1985.

[2] Anderson, E., "The Irises of the Gaspe Peninsula", *Bulletin of the American IRIS Society,* 59, 2-5, 1935.

[3] Ball, G.H., and D.J. Hall, "A clustering technique for summarizing multivariate data", *Behav. Sci.,* 12, 153-155, 1967.

[4] Bezdek, J.C., *Pattern Recognition with Fuzzy Objective Function Algorithms.* New York: Plenum Press, 1981.

[5] Bezdek, J.C., et al., "Fuzzy Kohonen clustering networks", *Proc. FUZZ-IEEE '92, San Diego,* 1035-1043.

[6] Bezdek, J.C., and N.R. Pal, "A critique of self-organizing feature maps", Submitted to *IEEE Trans. Neural Networks*: March 1993.

[7] Carpenter, G.A., and S. Grossberg, "ART 2: self-organization of stable category recognition codes for analog input patterns", *Applied Optics,* 26, 4919-4930, 1987.

[8] Cover, T.M., and P.E. Hart, "Nearest neighbor pattern recognition", *IEEE Trans. Information Theory,* IT-13, 21-27, January 1967.

[9] Dave, R.N., "Characterization and detection of noise in clustering", *Pattern Recognition Letters,* 12, 657-664, 1991.

[10] Dave, R.N., "Robust fuzzy clustering algorithms", *Proc. FUZZ-IEEE '93, San Francisco,* 1281-1286.

[11] DeSieno, D., "Adding a conscience to competitive learning", *Proc. IEEE Int. Conf. Neural Nets,* 1, 117-124, 1988.

[12] Fang, W.-C., et al, "A VLSI Neural Processor for Image Data Compression Using Self-Organization Networks", *IEEE Transactions on Neural Networks,* 3, 506-518, May 1992.

[13] Fisher, R.A., "The use of multiple measurements in taxonomic problems", *Annals of Eugenics,* 7, 179-188, 1936.

[14] Fritzke, B., "Unsupervised clustering with growing cell structures", *Proc. IEEE Int. Conf. Neural Nets,* 2, 531-536, 1991.

[15] Fukunaga, K., *Introduction to Statistical Pattern Recognition,* New York: Academic Press, 1972.

[16] Fukunaga, K., "Statistical Pattern Recognition", Young, T.Y. and K.S. Fu, eds., *Handbook of Pattern Recognition and Image Processing*, Orlando: Academic Press, 3-22, 1986.

[17] Gath, I., and B. Geva, "Unsupervised optimal fuzzy clustering", *IEEE Trans. Pattern Anal. and Mach. Int.,* 11(7), 773-781, July 1989.

[18] Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.

[19] Gray, R.M., "Vector quantization", *IEEE ASSP,* 1, 4-29, April 1984.

[20] He, Y., and U. Cilingiroglu, "A Charge-Based On-Chip Adaptation Kohonen Neural Network", *IEEE Transactions on Neural Networks*, 4, 462-469, May 1993.

[21] Hecht-Nielsen, R., "Information theory and neural networks", *Tutorials of the 2nd Int. Conf. Fuzzy Logic and Neural Networks,* (Iizuka, Japan, July, 1992), 31-39.

[22] Hinton, G.E., and T.J. Sejnowski, "Learning and relearning in Boltzmann machines", D.E. Rumelhart and J.L. McClelland, eds., *Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Vol.1*, Cambridge, MA: MIT Press, 1986.

[23] Hinton, G.E., "Deterministic Boltzmann learning performs steepest descent in weight-space", *Neural Computation,* 1, 143-150, 1989.

[24] Hopfield, J.J., "Neural networks and physical systems with emergent collective computational abilities", *Proc. Natl. Acad. Sci. USA,* 79, 2554-2558, April1982.

[25] Hopfield, J.J., et al, "Unlearning has stabilizing effects in collective memories", *Nature,* 304, 158-159, 1983.

[26] Hopfield, J.J., "Neurons with graded response have collective computational properties like those of two-state neurons", *Proc. Natl. Acad. Sci. USA,* 81, 3088-3092, May 1984.

[27] Hush, D.R., and B.G. Horne, "Progress in Supervised Neural Networks: What's New Since Lippmann?", *IEEE Signal Processing Magazine*, January 1993.

[28] Jain, A.K., "Pattern recognition", R.C. Dorf, ed., *International Encyclopedia of Robotics Applications and Automation,* New York: John Wiley and Sons, 2, 1052-1063, 1988.

[29] Johnson, R.A., and D.W. Wichern, *Applied Multivariate Statistical Analysis,* Prentice Hall, NJ, 1988.

[30] Kohonen, T., *Self-Organization and Associative Memory,* Springer-Verlag, 1989.

[31] Kohonen, T., "Self-Organized formation of topologically correct feature maps", *Bio. Cybern.,* 43, 59-69, 1982.

[32] Kohonen, T., "The self-organized map", *Proc. IEEE,* 78(9), 1464-1480, 1990.

[33] Krishnamurthy, A.K., et al., "Neural networks for vector quantization of speech and images", *IEEE Journal on Selected Areas in Communication,* 8(8), 1449-1457, October 1990.

[34] Krishnapuram, R., et al., "The fuzzy c spherical shells algorithm: a new approach", *IEEE Trans. Neural Networks,* 3(5), 663-670, September 1992.

[35] Lippman, R.P.,"An introduction to neural networks", *IEEE ASSP,* 4, 4-22, April 1987.

[36] Macq, D., et al, "Analog Implementation of a Kohonen Map with On-Chip Learning", *IEEE Transactions on Neural Networks,* 4, 456-461, May 1993.

[37] Miller, C.B., and C.L. Giles, "Experimental Comparison of the effect of order in recurrent neural networks", to be published in *International Journal of Pattern Recognition and Artificial Intelligence,* Special Issue on Applications of Neural Networks to Pattern Recognition, 1993.

[38] Pedrycz, W., and H.C. Card, "Linguistic interpretation of self-organizing maps", *Proc. FUZZ-IEEE '92, San Diego,* 371-380.

[39] Pedrycz, W., *Control and Fuzzy Systems,* Taunton, Somerset, England: Research Studies Press, 1989.

[40] Peterson, C., and J.R. Anderson, "A mean field theory learning algorithm for neural networks", *Complex Systems,* 1, 995-1019, 1987.

[41] Peterson, C., and E. Hartman, "Explorations of the mean field theory learning algorithm", *Neural Networks,* 2, 475-494, 1989.

[42] Rumelhart, D.E., and J.L. McClelland, eds., *Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Vol.1,* Cambridge, MA: MIT Press, 1986.

[43] Sammon, J.W., "A nonlinear mapping for data structure analysis", *IEEE Trans. Comput.,* 18, 401-409, 1969.

[44] Sharad, S., et al., "Source coding of speech and video signals", *Proc. IEEE,* 78(7), 1233-1249, 1990.

[45] Srikanth, R., et al., "Fuzzy elastic clustering", *Proc. FUZZ-IEEE '93, San Francisco,* 1179-1182.

[46] Williams, R.J., and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks", *Neural Computation,* 1, 270-280, 1989.

[47] Zadeh, L.A., "Fuzzy sets", *Inf. Control,* 8, 338-353, 1965.