2006

Genetically Engineered Adaptive Resonance Theory (art) Neural Network Architectures

Ahmad Al-Daraiseh
University of Central Florida

GENETICALLY ENGINEERED ADAPTIVE RESONANCE THEORY (ART) NEURAL NETWORK ARCHITECTURES

by

AHMAD A. AL-DARAISEH
B.S. Yarmouk University, 1998
M.S. University of Central Florida, 2001

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2006

Major Professor: Michael Georgiopoulos

ABSTRACT

Fuzzy ARTMAP (FAM) is currently considered to be one of the premier neural network architectures in solving classification problems. One of the limitations of Fuzzy ARTMAP that has been extensively reported in the literature is the category proliferation problem. That is Fuzzy ARTMAP has the tendency of increasing its network size, as it is confronted with more and more data, especially if the data is of noisy and/or overlapping nature. To remedy this problem a number of researchers have designed modifications to the training phase of Fuzzy ARTMAP that had the beneficial effect of reducing this phenomenon.

In this thesis we propose a new approach to handle the category proliferation problem in Fuzzy ARTMAP by evolving trained FAM architectures. We refer to the resulting FAM architectures as GFAM. We demonstrate through extensive experimentation that an evolved FAM (GFAM) exhibits good (sometimes optimal) generalization, small size (sometimes optimal size), and requires reasonable computational effort to produce an optimal or sub-optimal network. Furthermore, comparisons of the GFAM with other approaches, proposed in the literature, which address the FAM category proliferation problem, illustrate that the GFAM has a number of advantages (i.e. produces smaller or equal size architectures, of better or as good generalization, with reduced computational complexity).

Furthermore, in this dissertation we have extended the approach used with Fuzzy ARTMAP to other ART architectures, such as Ellipsoidal ARTMAP (EAM) and Gaussian ARTMAP (GAM) that also suffer from the ART category proliferation problem. Thus, we have designed and experimented with genetically engineered EAM and GAM architectures, named GEAM and GGAM. Comparisons of GEAM and GGAM with other ART architectures that were introduced in the ART literature, addressing the category proliferation problem, illustrate similar advantages observed by GFAM (i.e, GEAM and GGAM produce

smaller size ART architectures, of better or improved generalization, with reduced computational complexity).

Moverover, to optimally cover the input space of a problem, we proposed a genetically engineered ART architecture that combines the category structures of two different ART networks, FAM and EAM. We named this architecture UART (Universal ART). We analyzed the order of search in UART, that is the order according to which a FAM category or an EAM category is accessed in UART. This analysis allowed us to better understand UART's functionality. Experiments were also conducted to compare UART with other ART architectures, in a similar fashion as GFAM and GEAM were compared. Similar conclusions were drawn from this comparison, as in the comparison of GFAM and GEAM with other ART architectures.

Finally, we analyzed the computational complexity of the genetically engineered ART architectures and we compared it with the computational complexity of other ART architectures, introduced into the literature. This analytical comparison verified our claim that the genetically engineered ART architectures produce better generalization and smaller sizes ART structures, at reduced computational complexity, compared to other ART approaches.

In review, a methodology was introduced of how to combine the answers (categories) of ART architectures, using genetic algorithms. This methodology was successfully applied to FAM, EAM and FAM and EAM ART architectures, with success, resulting in ART neural networks which outperformed other ART architectures, previously introduced into the literature, and quite often produced ART architectures that attained optimal classification results, at reduced computational complexity.

I dedicate this work to the greatest woman on earth, my Mother, to my greatest Father, to my soul mate my wife, to my daughter NOOR the light of my life, to my son MO'MEN the faith of it, and finally to my unborn yet baby (Abdallah if male, Tasneem if female).

I love you all

ACKNOWLEDGMENTS

I would like to express my gratitude to my advisor, Dr. Michael Georgiopoulos, who gave me his time and advice to help me finish this dissertation. Without his encouragement, support, and guidance, this dissertation would not have been published. I would like to thank my committee members, Dr. Ronald F. Demara, Dr. Kent Williams, Dr. Sheau-Dong Lang, and Dr. Takis C. Kasparis, for their support and willingness to serve on my defense examination.

I would like to thank Dr. Gerd Brummel my manager at Siemens PG. Dr. Brummel was one of those who always encouraged me and stood by my side until I finished this work. Thank you very much Dr. Brummel.

I would like to thank my parents and wife for their help and support, and my kids Noor and Mo'men for their disturbance ☺. I also would like to thank my friends and brothers who helped me and prayed for me.

Special thanks to UCF and to those who are working hard to improve it and make it better, please don't raise your tuition fees ☹ anymore.

TABLE OF CONTENTS

LIST OF FIGURES x
LIST OF TABLES xiii
LIST OF ACRONYMS/ABBREVIATIONS xiv
1. INTRODUCTION 1
1.1 ART, Features and Limitations 1
1.2 Genetic Algorithms and Neural Networks Combination 3
1.2.1 Using GA with MLP NN 4
1.2.2 Using GAs with Other NN Models (other than MLP-NNs) 6
1.2.3 Using GAs with ART NNs 6
1.3 Motivation 7
1.4 Research Overview 8
1.4.1 Using GAs to Evolve ART Architectures 8
1.4.2 Universal ART (UART) 9
1.4.3 Experiments and Comparisons 10
1.4.4 Analysis 10
1.4.5 User Interface Development 11
2. BACKGROUND 13
2.1 Fuzzy ARTMAP (FAM) 13
2.1.1 FAM Category Geometrical Representation 15
2.1.2 FAM Operations and Parameters 16
2.2 Ellipsoidal ARTMAP (EAM) 20
2.2.1 EAM Category Geometrical Representation 21
2.2.2 EAM Operations and Parameters 22
2.3 Gaussian ARTMAP (GAM) 26
2.3.1 GAM Operations and Parameters 28
2.4 Genetic Algorithms 31
2.4.1 Chromosome Representation 32
2.4.2 Genetic Operators 33
2.4.3 Selection 34
3. GENETIC FUZZY ARTMAP (GFAM) 36
3.1 Justification of the Evolutionary Choices for GFAM 41
3.1.1 Justification of the Fitness Function Choice for GFAM 41
3.1.2 Justification of the Genetic Operators Choices for GFAM 44
3.2 Experiments with GFAM 51
3.2.1 Databases 51
3.2.2 Experimental Procedure – Experimental Results 57
3.3 GFAM Performance 57
3.3.1 Performance Comparisons of GFAM and other ART Networks 58
3.3.2 Performance Comparisons of GFAM and Other Neural Networks 63
3.4 GFAM Summary and Conclusions 65
4. GEAM AND GGAM 67
4.1 Genetic Ellipsoidal ARTMAP (GEAM) 67
4.1.1 GEAM Experiments and Results 72
4.1.1.1 GEAM Performance 72
4.1.1.2 Performance Comparisons of GEAM and other ART Networks 73
4.1.2 Summary/Conclusions 78
4.2 Genetic Gaussian ARTMAP (GGAM) 79
4.2.1 GGAM Experiments and Results 84

4.2.1.1 GGAM Performance 85
4.2.1.2 Performance Comparisons of GGAM and other ART Networks 86
4.2.2 Summary/Conclusions 92
5. UNIVERSAL ART (UART) 93
5.1 UART Design 95
5.1.1 Performance Phase of UART 97
5.1.2 Geometry Selection Phase (Genetic Phase) of UART 99
5.2 Results of UART 105
5.2.1 UART Performance 105
5.2.2 Performance Comparisons of UART and other ART Networks 106
5.2.3 Performance Comparisons of UART and other Genetic ART Networks 112
5.3 UART Summary 115
6. ANALYSIS 117
6.1 UART Order of Search Analysis 117
6.2 Time Complexity Analysis 129
7. USER INTERFACE 133
7.1 GFAM User Interface 133
7.1.1 GFAM Controls 134
7.1.2 GFAM UI Abstract Design 144
7.1.2.1 GFAMForm Object 144
7.1.2.2 GraphForm Object 146
7.1.2.3 FNode Object 147
7.1.2.4 PtrnNode Object 147
7.1.2.5 Chrom Object 148
7.1.2.6 Cat Object 149
7.1.2.7 AddCatForm Object 149
7.2 GEAM User Interface 149
7.2.1 GEAM Controls 151
7.2.2 GEAM UI Abstract Design 156
7.2.2.1 GEAMForm Object 157
7.2.2.2 GraphForm Object 158
7.2.2.3 ENode Object 158
7.2.2.4 PtrnNode Object 159
7.2.2.5 Chrom Object 159
7.2.2.6 Cat Object 159
7.2.2.7 AddCatForm Object 160
7.3 GGAM User Interface 160
7.3.1 GGAM Controls 161
7.3.2 GGAM UI Abstract Design 166
7.3.2.1 GGAMForm Object 167
7.3.2.2 GraphForm Object 167
7.3.2.3 GNode Object 167
7.3.2.4 PtrnNode Object 168
7.3.2.5 Chrom Object 168
7.3.2.6 Cat Object 169
7.3.2.7 AddCatForm Object 169
7.4 UART User Interface 169
7.4.1 UART Controls 171
7.4.2 UART UI Abstract Design 177
7.4.2.1 UARTForm Object 178

7.4.2.2 GraphForm Object 181
7.4.2.3 FNode, ENode and PtrnNode Objects 182
7.4.2.4 Chrom Object 182
7.4.2.5 Cat Object 183
7.4.2.6 AddCatForm Object 183
8. SUMMARY/CONTRIBUTIONS, AND FUTURE WORK 184
8.1 Summary/Contributions 184
8.2 Future Work 185
APPENDIX A: TERMINOLOGY 186
APPENDIX B: FAM STEP-BY-STEP TRAINING & TESTING 191
APPENDIX C: EAM STEP-BY-STEP TRAINING & TESTING 196
APPENDIX D: GAM STEP-BY-STEP TRAINING & TESTING 202
APPENDIX E: USER MANUAL 209
REFERENCES 212

LIST OF FIGURES

Figure 2-1: Simple FAM Architecture....13
Figure 2-2: A rectangle representation of a FAM category that learned seven input patterns 15
Figure 2-3: The distance of an input pattern a from the rectangle R_j^a is the minimum distance of the pattern a from the border of the rectangle R_j^a 16
Figure 2-4: FAM Learning, a. A category with 0 size; b. Introducing a new pattern a2; c. The category expands to include a2; d. Since a3 is inside the category, it doesn't change its size; e. Pattern a4 is presented; f. Since a4 is outside the category, the category is expanded to include a4, within its boundaries. 19
Figure 2-5: Simple EAM Architecture 20
Figure 2-6: An EAM category that encodes 3 patterns....22
Figure 2-7: Creation and expansion of an EAM category 26
Figure 2-8: A 2D GAM category that encodes 5 patterns within 2 standard deviations....28
Figure 2-9: One-point crossover 33
Figure 2-10: Two-point crossover 33
Figure 2-11: Uniform crossover....34
Figure 3-1: GFAM chromosome structure 38
Figure 3-2: Crossover implementation 40
Figure3-3a: 3D plot of log(fit(p)) 43
Figure 3-4: a: Problem 1 (Four squares in a square problem), b: (Asymmetric squares within a square problem), c: Problem 3 (Two circles in a square problem)....47
Figure 3-5: Average Fitness value of the Best FAM produced by GFAM for Problem 1. The average is computed over the 50 runs. The average fitness values are shown with respect to all pairs of category add and category delete probabilities. The different colored curves correspond to the three different values of the mutation probability.....49
Figure 3-6: Average Fitness value of the Best FAM produced by GFAM for Problem 2. The average is computed over the 50 runs. The average fitness values are shown with respect to all pairs of category add and category delete probabilities. The different colored curves correspond to the three different values of the mutation probability.....50
Figure 3-7: Average Fitness value of the Best FAM produced by GFAM for Problem 2. The average is computed over the 50 runs. The average fitness values are shown for all pairs of category add and category delete probabilities. The different colored curves correspond to the three different values of the mutation probability....50
Figure 3-8: Average Fitness value of the Best FAM produced by GFAM for Problems 1, 2 and 3. The average is computed over the 50 runs. The average fitness values are shown for all pairs of category add and category delete probabilities. The different colored curves correspond to the two different non-zero values of the mutation probability....51
Figure 3-9: Gaussian Databases (2-dimensional, 2, 4 or 6 class, 5, 15, 25 and 40 % of overlap) 56
Figure 3-10: Structures within Structure Databases 56
Figure 3-11a: Performance and Size comparison of GFAM vs ssFAM....61
Figure 3-11b: Performance and Size comparison of GFAM vs ssEAM 61
Figure 3-11b: Performance and Size comparison of GFAM vs ssGAM....62
Figure 3-11d: Performance and Size comparison of GFAM vs microARTMAP 62
Figure 4-1: GEAM chromosome structure 69
Figure 4-2a: Performance and Size comparison of GEAM vs ssFAM....76
Figure 4-2b: Performance and Size comparison of GEAM vs ssEAM 76
Figure 4-2c: Performance and Size comparison of GEAM vs ssGAM....77

Figure 4-2d: Performance and Size comparison of GEAM vs microARTMAP 77
Figure 4-3: GGAM Chromosome Structure 82
Figure 4-4a: Performance and Size comparison of GGAM vs ssFAM 90
Figure 4-4b: Performance and Size comparison of GGAM vs ssEAM.....90
Figure 4-4c: Performance and Size comparison of GGAM vs ssGAM.....91
Figure 4-4d: Performance and Size comparison of GGAM vs microARTMAP.....91
Figure 5-1: These figures show what happens when using unsuitable classifiers for a certain problem. 94
Figure 5-2: a classification problem where the boundaries can't be optimally covered by FAM, GAM or EAM's categories. 94
Figure 5-3: Using UART to solve the problem in figure 5-2, notice that parts of the problem space are not covered, but because UART encourages smaller size, it might sacrifice little accuracy to get optimal size. 95
Figure 5-4: A simple UART structural diagram during the training phase 98
Figure 5-5: A simple UART structural diagram during the performance phase 99
Figure 5-6: GFAM chromosome structure 101
Figure 5-7: Crossover implementation 103
Figure 5-8a: Performance and Size comparison of UART vs ssFAM.....110
Figure 5-8b: Performance and Size comparison of UART vs ssEAM 110
Figure 5-8c: Performance and Size comparison of UART vs ssGAM.....111
Figure 5-8d: Performance and Size comparison of UART vs microARTMAP 111
Figure 5-9a: Performance and Size comparison of UART vs GFAM.....114
Figure 5-9b: Performance and Size comparison of UART vs GEAM 114
Figure 5-9c: Performance and Size comparison of UART vs GGAM 115
Figure 6-1: Case # 1, a pattern inside both a FAM category and an EAM category.....118
Figure 6-2: Case # 2, a pattern inside a FAM category but outside an EAM category 119
Figure 6-3: Case # 3, a pattern inside an EAM category but outside a FAM category 121
Figure 6-4: Case # 4, a pattern outside both FAM and EAM categories.....123
Figure 7-1: GFAM User interface.....134
Figure 7-2: An open dialogue window, allows the user to select the training, validating and testing files.....135
Figure 7-3: A dialogue box that displays the results after an interruption of the process 137
Figure 7-4: A 2D Graph that displays the data points as well as the categories.....138
Figure 7-5: Same graph as in figure 7-4 but displaying the categories only 138
Figure 7-6: A 2D graph displaying the classification borders of this GFAM as well as the categories 139
Figure 7-7: After pushing the "Del All" button, the GFAM does not have any more categories 140
Figure 7-8: This figure shows figure 7-7, but only displaying the categories (none in this case) 141
Figure 7-9: An add category dialogue box 141
Figure 7-10: This figure is the same as figure 7-9 but after filling in some values.....141
Figure 7-11: This figure shows the manually added category.....142
Figure 7-12: This figure shows the classification borders of the manually added category..142
Figure 7-13: This figure shows the values of the endpoints of the second manually added category.....143
Figure 7-14: This figure shows the two manually added categories 143
Figure 7-15: This figure shows the classification borders of the manually added categories143
Figure 7-16: GEAM user interface 150

Figure 7-17: a 2D graph displaying a GEAM network; note here ellipsoids are represented by circles 151
Figure 7 -18: A 2D graph showing the classification borders of the GEAM network 152
Figure 7 -19: This figure shows the GEAM network after pushing the “Del All” button 153
Figure 7-20: An add GEAM category dialogue box 153
Figure 7-21: Manually filling in values for the first category 154
Figure 7-22: GEAM network after manually adding a category 154
Figure 7-23: Classification borders of the manually added category 155
Figure 7-24: Manually adding a new category 155
Figure 7-25: GEAM network after adding the second category 156
Figure 1-26: Classification borders of the GEAM network 156
Figure 7-27: GGAM user interface 161
Figure 7-28: A 2D graph showing a GGAM network; note here a GGAM category is represented by a large dot 162
Figure 7-29: A 2D graph showing the classification boundaries of the GGAM network 162
Figure 7-30: After deleting all the categories 163
Figure 7-31: An add GGAM category dialogue box 163
Figure 7-32: Add category dialogue box with numbers in the available boxes 164
Figure 7-33: A figure showing the manually added category 164
Figure 7-34: The classification boundaries corresponding to the manually added category 165
Figure 7-35: Filling in numbers for the second category 165
Figure 7-36: The GGAM network after adding the second category 165
Figure 7-37: The classification boundaries of the GGAM after adding two categories 166
Figure 7-38: UART user interface 170
Figure 7-39: A UART network after randomly mixing FAM categories with EAM categories 172
Figure 7-40: A 2D graph showing only the categories 172
Figure 7-41: The classification borders of the UART network after the random mixing 173
Figure 7-42: After deleting all the categories of UART 174
Figure 7-43: An add category dialogue box 174
Figure 7-44: Filling in data for an EAM category 175
Figure 7-45: UART after manually adding an EAM category 175
Figure 7-46: The classification boundaries of UART after addition 176
Figure 7-47: Filling in data for a FAM category 176
Figure 7-48: UART after manually adding an EAM and a FAM category 177
Figure 7-49: UART classification boundaries after the manual addition 177
Figure e-1: Error message 210

LIST OF TABLES

Table 3-1: The values of the probabilities for mutation, category add, and category delete used in the experiments to determine good values for the GA parameters 46
Table 3-2: For each problem (database) we ran 3 experiments. For each experiment we used the depicted combinations of number of generations, and population size (3 combinations). We evolved the trained Fuzzy ARTMAPs 50 different times (50 random seeds), and for each time we used the combinations of probability values, shown in Table 3-1. Hence, the FAMs were evolved 1350 times for each problem, or a total of 4050 times for all the problems. 46
Table 3-3: Databases used in the Genetic ARTMAP experiments.....55
Table 3-4: Accuracy and size results achieved by GFAM and other ART networks. Note that:Safe uAM: Safe microARTMAP; FAM: Fuzzy ARTMAP; EAM: Ellipsoidal ARTMAP; GAM: Gaussian ARTMAP; ss*: semi-supervised version.....63
Table 3-5: Accuracy and size results achieved by GFAM and other ART networks. Note: dFAM: Distributed Fuzzy ARTMAP, FasART, dFasART : Distributed FasART, GFAM : Genetic Fuzzy ARTMAP.....65
Table 4-1: Accuracy and size results achieved by GEAM and other ART networks. Note that:Safe uAM: Safe microARTMAP; FAM: Fuzzy ARTMAP; EAM: Ellipsoidal ARTMAP; GAM: Gaussian ARTMAP; ss*: semi-supervised version.....78
Table 4-2: Accuracy and size results achieved by GGAM and other ART networks. Note that:Safe uAM: Safe microARTMAP; FAM: Fuzzy ARTMAP; EAM: Ellipsoidal ARTMAP; GAM: Gaussian ARTMAP; ss*: semi-supervised version.....89
Table 5-1: UART performance and size compared to other ART architectures 109
Table 5-2: UART performance and size compared to other genetic ART architectures.....113

LIST OF ACRONYMS/ABBREVIATIONS

- NN: Neural Network
- ART: Adaptive Resonance Theory
- FAM: Fuzzy ARTMAP
- Category Proliferation: An ART limitation that causes it to create more categories when confronted with noisy/overlapping data
- GA: Genetic Algorithm
- GFAM: Genetic Fuzzy ARTMAP
- EAM: Ellipsoidal ARTMAP
- GEAM: Genetic Ellipsoidal ARTMAP
- GAM: Gaussian ARTMAP
- GGAM: Genetic Gaussian ARTMAP
- UART: Universal ART
- stability plasticity Dilemma: Creating a stable neural network that can learn new inputs without relearning
- MLP: Multi-Layer Perceptron
- BP: Back Propagation
- RBF : Radial Basis Function
- UI: User Interface
- Committed node: A node that has established a connection to an output node (class)
- CCF: Category Choice Function
- CMF: Category Match Function
- PCC: Percent Correct Classification

1. INTRODUCTION

The main focus of this dissertation is to present a methodology of how to use genetic algorithms (GA) to construct optimal or sub-optimal ART networks that solve deficiencies existing in current ART models, such as category proliferation, poor coverage of the problem's input space, and large dependency on parameters. This task is accomplished while at the same time producing genetically engineered ART architectures that improve generalization at a reduced computational cost. The various sections in this introduction give the reader a better understanding of the main components of this dissertation along with some related literature.

1.1 ART, Features and Limitations

The Adaptive Resonance Theory (ART) was developed by (Grossberg, 1976) as a solution to the stability plasticity dilemma. One of the most celebrated ART architectures is FAM (short for Fuzzy ARTMAP) (Carpenter et al, 1992), which has been successfully used in the literature for solving a variety of classification problems. Some of the advantages that Fuzzy ARTMAP possesses is that it can solve arbitrarily complex classification problems, it converges quickly to a solution (within a few presentations of the list of the input/output patterns belonging to the training set), it has the ability to recognize novelty in the input patterns presented to it, it can operate in an on-line fashion (new input/output patterns can be learned by the system without re-training with the old input/output patterns), and it produces answers that can be explained with relative ease.

Despite all the great features that Fuzzy ARTMAP possesses, it suffers from the category proliferation problem, especially when it is confronted with data that are of noisy and/or overlapping nature. Quite often the category proliferation problem, observed in Fuzzy ARTMAP architectures, is connected with the issue of overtraining in Fuzzy ARTMAP. Over-training happens when Fuzzy ARTMAP is trying to learn the training data perfectly at

the expense of degraded generalization performance (i.e., classification accuracy on unseen data) and also at the expense of creating many categories to represent the training data (leading to the category proliferation problem). Another reason for the category proliferation problem is that ART architectures rely on a single category structure (hyper-rectangles) to represent the data in the input space. Genetically engineered ART architectures that combine more than one category structure (such as hyper-rectangles and ellipsoids) address this problem.

A number of authors have tried to address the category proliferation problem in Fuzzy ARTMAP by tackling either the training method or the geometrical representation that FAM architectures use. Amongst them we refer to the work by Marriott (Marriott and Harrisson, 1995), where the authors eliminate the match tracking mechanism of Fuzzy ARTMAP when dealing with noisy data, the work by Charalampidis (Charalampidis, et al., 2001), where the Fuzzy ARTMAP equations are appropriately modified to compensate for noisy data, the work by Verzi (Verzi, et al., 2001), (Anagnostopoulos, et al., 2003 & 2001a), and (Gomez-Sanchez, et al., 2002 & 2001) , where different ways are introduced of allowing the Fuzzy ARTMAP categories to encode patterns that are not necessarily mapped to the same label, the work by Koufakou (Koufakou, et al., 2001), where cross-validation is employed to avoid the overtraining/category proliferation problem in Fuzzy ARTMAP, and the work by Carpenter (Carpenter, 1998), Williamson (Williamson, 1997), Parrado-Hernandez (Parrado-Hernandez, et al., 2003), where the ART structure is changed from a winner-take-all to a distributed version and simultaneously slow learning is employed with the intent of creating fewer ART categories and reducing the effects of noisy patterns.

Another limitation with ART architectures is that their performance depends on a number of network parameters, and sometimes it becomes computationally expensive to discover a set of network parameters that produces a network with good generalization and

small size. What makes this issue more difficult is that the best network parameters are problem dependent. As it will be seen later, genetically engineered ART architectures solve this problem as well.

1.2 Genetic Algorithms and Neural Networks Combination

Genetic algorithms are a class of population-based stochastic search algorithms that are developed from ideas and principles of natural evolution. An important feature of these algorithms is their population based search strategy. Individual chromosomes in a population compete, exchange and modify information with each other in order to perform certain tasks. Genetic algorithms have been widely used to evolve artificial neural networks. For a thorough exposition of the available research literature in evolving neural networks the interested reader is advised to consult (Yao, 1999). In (Yao, 1999) the author distinguishes three different strategies in evolving neural networks. The first strategy is the one used to search for the weights of the neural network, the second one is the one used to design the structure of the network, and the third one is the one where the learning rules of the neural network are evolved. Examples of the first strategy are ones where the weights found by the genetic algorithm are used in the neural network without further refinement (Whitley, et al., 1989 and 1990). An alternative to this strategy is to use the pertinent neural network learning to further refine the weights that the GA algorithm produces (Belew, et al., 1991 and Lee, 1996). Beside searching for weights, GAs may also be used to select the features that are input to the neural network. Since the pioneering work by Siedlecki and Sklanski (Siedleck and Sklansky, 1989), genetic algorithms have been used for many selection problems using neural networks (Brotherton, Simpson, 1994, Yang and Honavar, 1998), and other classifiers, such as decision trees (Bala, et al., 1996), k-nearest neighbors (Kelly and Davis, 1991, Punch, et al., 1993), and naïve Bayes classifiers (Inza, et al., 1999, Cantu-Paz, 2002). As it has been

verified in the literature the topology of the neural network is crucial to its performance. If a network has too few nodes it might not be able to learn the required task. On the other hand, if the network has too many nodes it may overfit the training data and thus exhibit poor generalization. Miller, Todd and Hedge (Miller, et al., 1989) defined two major approaches to use GAs to design the topology of the neural networks: use of direct encoding to specify every connection of the network, or to evolve an indirect specification of the connectivity. The direct encoding GA approach implies that every connection between every node be directly represented in a chromosomal string. Direct encoding has been used effectively to prune neural networks with good results (Whitley, et al., 1990, Hancock, 1992). On the other hand, a simple indirect encoding method is to commit to a particular topology (e.g., feed-forward or recurrent NN) and a learning algorithm (e.g. back-prop learning algorithm), and then use a GA to find the parameter values that complete the network specification. For example, the GA in the feed-forward neural network approach can search for the number of layers and the number of units (nodes) per layer. The indirect encoding scheme is far more sophisticated while being theoretically capable of representing complicated topologies with finesse. It encodes the most important parameters and leaves the remainder to be determined elsewhere. Harp, et al. (see Harp, et al., 1989) used segments of two parts in an encoding scheme entitled blueprints. The first segment held parameter specifications including address, organization and number of nodes, and learning parameters associated with the nodes. The second segment described the connections between themselves by specifying the density between the current area and the target area, the target's area address, organization of the connections, and parameters of learning associated with the connection weights.

1.2.1 Using GA with MLP NN

A thorough study of the literature shows that most of the articles that involved GA and NN used a Multi-Layer Perceptron (MLP) NN. For instance, Hruschka (Hruschka, et al.,

2000) used GA to extract classification rules from a MLP NN (specifically, they used the activation functions of the hidden nodes to extract the rules). The genetic algorithm was utilized to cluster the activation functions and hence to extract the rules. Krasniewicz (Krasniewicz, et al., 2000) investigated different methods in constructing a GA/MLP NN in a distributed environment. It is a known fact that training a NN could take a long time, and as such combining NNs with GAs amplifies this problem (since many NNs are needed to be trained when GAs are involved). This problem triggered the idea of using distributed (parallel) system, in Karsniewicz, et al., 2000). Santos (Santos, et al., 2000) used GAs to extract comprehensible rules (IF-Then Statements) from MLP NN's. In his work the GA was used to find the NN that provides us with the best rules. Yen (Yen, et al., 2000) used a hierarchical GA to construct an MLP NN. This method, as Yen claimed, solved the Network Feasibility problem, as well as the problem of mapping multiple phenotypes by genotypes, that appears when using GAs to construct NNs. Fieldsend (Fieldsend, et al., 2005) proposed a new method called Pareto ENN (evolutionary Neural Network) to construct multi-objective optimized NN that is capable of time series forecasting. The idea of his approach was to keep a set of F of chromosomes (NNs) at each generation, each one of which is the best performer regarding a specific measure. In the next generation if some chromosomes are better than their counter parts in F then we replace the chromosomes in F. This process continues until a maximum number of generations has passed. Manic (Manic, et al., 2002) used a combination of GAs with gradient descent to find the best set of weights. In his approach the GA is used to find a sub-optimal set of weights, and then the gradient descent method is used to fine tune this set. Leung (Leung, et al., 2003) proposed a modified GA and a modified BP-NN and used the new GA method to tune the parameters and the structure of the NN. The GA generates four off-springs from each couple of parents, while the MLP NN uses switches for its links.

1.2.2 Using GAs with Other NN Models (other than MLP-NNs)

Various authors used other NN models with GAs. Xiuju (Xiuju,et al., 2002) proposed a new RBF NN (Radial Basis Function) that uses GA to select class-dependent features (i.e. different hidden nodes are connected to different features based on the output of the GA). Liang-Hsuan (Liang-Hsuan, et al., 2002) proposed a new Intelligent Control System ICS, based on a multi-objective genetic algorithm. The idea is to use the GAs to find optimal or suboptimal actions when needed. Based on the principle of the nearest neighbor algorithm and NN, Ishibuchi (Ishibuchi, et al., 1997) proposed a method to construct a rule-based classification system using GA. His goal was to generate a set of fuzzy rules that minimize the error, rejection ratio and the number of rules. To create a set of rules every training pattern is used as separate rule. In the case were the training patterns are many, the designers usually use other means to compact the number of training patterns (rule's centers). Ghosh (Ghosh, et al., 2002) presented a new method to construct Error BackProp Neural Networks EBP NN. The idea is to use a mix of Genetic Algorithm and the Least Square Method to tune the weights and the number of hidden nodes of the network. Rovithakis (Rovithakis, et al., 2004) presented a procedural method to construct a HONN (High-Order Neural Network) using GAs. This method was then used to construct HONN for function approximation purposes. Chia-Feng (Chia-Feng, et al. 2004) used a new evolution method to train a recurrent neural network, a mixture of GA and PSO (practical swarming theory) was used to evolve the weights of the NN. The idea is to have the GA search for optimal solution through its crossover and mutation operators and then to use the PSO enhancement of the generation to create an even better generation of solutions.

1.2.3 Using GAs with ART NNs

There are only a handful of articles that used ART NNs with GAs. One of the methods of combining NNs with GAs is to use the NN as a fitness function evaluator for the

GA, Burton (Burton, et al., 1997) used an Adaptive Resonance Theory (ART) NN as a fitness evaluator. The authors used a GA to compose new musical rhythms, to see how similar these newly created ones are to the existing ones the authors used an ART network to classify the new rhythms.

It is a well known fact that the larger the number of features in a specific problem, the more complex and time consuming the classification process becomes. Quite often a large number of features compromises the accuracy of the classifier, as well. Palaniappan (Palaniappan, et al., 2002) developed a new method that uses Genetic Algorithms to select features that are then used as an input to a Fuzzy ARTMAP classifier. This method was applied on a real world problem.

Hui (Hui, tl al., 2003) claimed in their paper that the effect of different features is different from one to another and hence multiplying some features by a certain factor (impulsive force) could improve the generalization of the network. Hence, Hui proposed a new ART architecture called Impulsive Fuzzy ART (IFART), and used a GA to find the right impulsive forces.

In all the articles presented above (section 1.2.1 to 1.2.3), there is one common fact. Using GAs to evolve NNs yields optimal or sub-optimal neural network structures. It was also common that the results of using GAs with MLP-NNs were very successful, and when compared to the original network the genetic model always gave better results.

1.3 Motivation

After a careful study of the literature presented above and in the next chapter, it is very evident that ARTMAP architectures suffer from the *category proliferation,* problem. It is also evident that using GAs to evolve NNs has been a very successful approach to find

optimal topologies and weight sets. For the above reasons, we decided to investigate the use of GA's to evolve ART architectures.

1.4 Research Overview

In this dissertation the work was divided into five major efforts, presented in the following five sections.

1.4.1 Using GAs to Evolve ART Architectures

In this project, GAs were successfully used to simultaneously evolve the weights, as well as the topology of three ART neural networks, namely: FAM, EAM and GAM. But in contrast to the feed-forward neural networks that have been extensively evolved, ART neural networks have a number of topological constraints, such as (a) they consist of one hidden layer of nodes, and (b) every interconnection weight value from every node of the input layer to a node in the hidden layer is important (representing the minimum or the maximum of the values of input patterns across every dimension that were encoded by this node). Consequently, the only element of an ART topology that can be evolved is the number of nodes in the hidden layer. Furthermore, although we could start from an initial population of randomly chosen number and values of the weights, in our application we start with a population of *trained* ART networks, whose number of nodes in the hidden layer and the values of the interconnection weights converging to these nodes are fully determined (at the beginning of the evolution) by the specific ART training rules. To this initial population of networks, GA's are applied to modify these trained network's architectures (number of nodes in the hidden layer, and values of the interconnection weights) in a way that encourages better generalization and smaller size architectures.

It is worth reminding the reader that as with many neural network architectures, the knowledge in ART networks is stored in their interconnection weights that have a very

interesting geometrical interpretation (see Anagnostopoulos, et al., 2001). For example, the interconnection weights in FAM (converging to the nodes in the hidden layer) represent the lower and upper end-points of hyper-rectangles (referred to as categories) that enclose within their boundaries clusters of data that are mapped to the same label.

Eventually, the evolution of these trained networks produces an ART architecture, referred to as *GFAM, GEAM or GGAM* , and extracted from the last generation as the network that attained the highest fitness value. The GFAM, GEAM or GGAM network attained better generalization performance and smaller size than the networks that we started with in the initial GA population.

It is apparent that in evolving neural network architectures one has to decide on the genotype representation scheme for the neural network architecture under consideration, on the genetic operators used to evolve these neural network architectures and on the fitness function used to guide this evolution. In this dissertation we address these issues in a manner that fits the characteristics of the ART neural networks and our ultimate objective of reducing category proliferation in ART, while we preserve a good (sometimes optimal) generalization performance.

1.4.2 Universal ART (UART)

In the first section of this introduction, we related the category proliferation problem in part to the fact that any ART or ARTMAP module, introduced into the literature, uses only one category representation to cover the input space of the problem. In this effort a new ARTMAP architecture is proposed. Universal ART (UART) is a new architecture that has the potential of combining multiple category representations in one network. The current version combined both EAM and FAM to create a network that covers the input space with hyper-rectangles and/or hyper-ellipsoids when needed. This architecture benefits from the use of GAs to select the best categories for a specific problem.

UART trains half the of the GA population as FAM networks and the other half as EAM networks. Through a process called shuffling, the categories of all the networks are then redistributed amongst the chromosomes to ensure fair fitness evaluation of the initial population. UART then uses standard GA steps to find an optimal or sub-optimal network, that consists from FAM categories only, EAM categories only or possibly a combination of both types. The selected categories rely heavily on the nature of the problem at hand and on the random seeds used during the GA search.

1.4.3 Experiments and Comparisons

Extensive experimentation was conducted with GFAM, GEAM, GGAM and UART. The goal of this experimentation was to show the superiority of these models versus their existing ART counterpart architectures. The comparison was based on the accuracy of the architectures and size of the architectures produced by these techniques, as well as the computational effort involved in producing these architectures. Another goal of the experimentation was to discover a good, default set of GA parameter values to evolve the ART neural networks with.

1.4.4 Analysis

This effort is actually divided into two separate sub-efforts: **a.** An order of search analysis of the UART architecture, and **b**. A time complexity analysis of the *genetic* approach in finding an optimal network and the traditional approach of finding the best network which we refer to as exhaustive search.

In effort a, we proved four theorems that explain the order according to which a FAM versus an EAM category are searched. These theorems explain which category will represent an input pattern if the input pattern was 1. Inside both categories, 2. Inside the FAM category but outside the EAM category, 3. Inside the EAM category but outside the FAM category, 4. Outside both categories. Then we drew twelve results based on these theorems.

In effort b, we present the pseudo code for the genetic approach and then analyze its time complexity. Then we present the exhaustive search pseudo code and analyze its time complexity. From this comparison it is discovered that the genetic approach time complexity of $O(N^4)$ is much better than that of the exhaustive search of $O(N^7)$.

1.4.5 User Interface Development

A major effort through out this research was devoted to the design, implementation and testing of the user interface (UI). In fact, four different programs were developed namely GFAM UI, GEAM UI, GGAM UI and UART UI. The following are few of the many requirements these programs had to have:

- Capable of creating a variable number of ARTMAP networks (chromosomes).
- Capable of coding ARTMAP networks to chromosomes and vice versa.
- Capable applying genetic algorithms on the created chromosomes.
- Can run one or multiple generations at a time (user defined).
- Can display a 2D graphs of the categories and the input patterns (2D problems only).
- Can display a 2D graph of the classification borders for a specific network (2D problems only).
- Allows the user to insert and delete specific categories from a network.
- Can log different levels of details to log files.

The above requirements and others were successfully designed, implemented and tested for all of the four architectures.

The organization of this dissertation is as follows: In chapter 2 we present background information related to FAM, EAM and GAM architectures, and we also present the evolutionary computation (EC) concept, its variations and applicability. In chapter 3, we describe all the necessary elements of evolving FAM architectures, as well as the experiments

and results. In chapter 4, we introduce GEAM and GGAM, along with their results and associated comparisons. In chapter 5, we propose the new architecture UART. In chapter 6, we present the time complexity and the order of search analysis. In chapter 7, we go through the details of developing the UI programs used to evolve the different ART architectures. In chapter 8, we summarize our contributions, and we provide directions for future research.

2. BACKGROUND

In this chapter the main building blocks of this research are presented in detail. In particular, the following sections give the reader a thorough understanding of FAM, EAM, GAM and EA. It is worth mentioning, that there are semi-supervised versions of the above ART modules namely: ssFAM, ssEAM and ssGAM. ssFAM, ssEAM and ssGAM performance was compared with the performance of genetically engineered ART networks. The only difference between the two versions FAM and ssFAM, or EAM and ssEAM, or GAM and ssGAM is that the semi-supervised versions allow the categories to encode patterns that go to a mixture of labels, provided that the majority label is above a certain threshold. This semi-supervised feature reduced the size of the ART network and increased its generalization accuracy (see Anagnostopoulos, et al, 2003). In our comparisons of the evolved ART models, we used semi-supervised models ART to compare to.

2.1 Fuzzy ARTMAP (FAM)

The Fuzzy ARTMAP architecture consists of three layers or fields of nodes (see Figure 2-1).

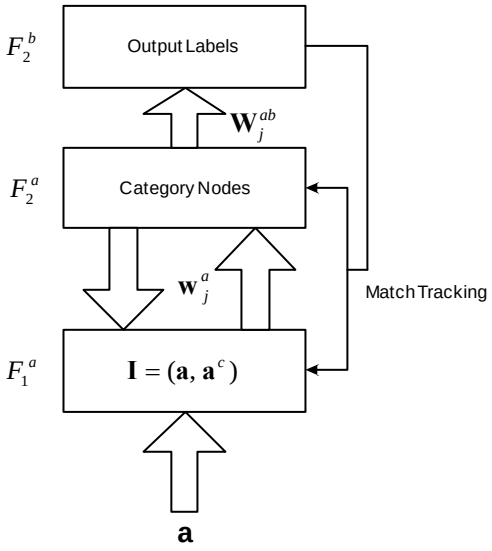

Figure 2-1: Simple FAM Architecture

These layers are *input layer* (F_1^a), the *category representation layer* (F_2^a), and the *output layer* (F_2^b). The input layer of FAM is the layer where inputs are applied. An input applied to F_1^a is a vector $\mathbf{I}$ of dimensionality $2M_a$ of the following form,

$$\mathbf{I} = (\mathbf{a}, \mathbf{a}^c) = (a_1,...,a_{M_a}, a_1^c,...,a_1^c,...,a_{M_a}^c); \quad a_i^c = 1 - a_i, \quad 1 \le i \le M_a \qquad 2\text{-}1$$

where $\mathbf{a}$ is a vector whose components lie in the interval $[0,1]$. Thus, layer F_1^a is a layer that contains $2M_a$ nodes, one node for each component of the input pattern $\mathbf{I}$. The index *i* $(1 \le i \le 2M_a)$ designates a generic node in layer F_1^a . The layer F_2^a of FAM is referred to as the *category representation layer*, because this is where categories (or groups) of input patterns are formed. Finally, the output layer (layer F_2^b) is the layer that produces the outputs of the network. Every node in the output layer of FAM represents one of the labels of the pattern recognition task. The index *k* $(1 \le k \le N_b)$ designates a generic node in F_2^b ; N_b represents the highest index needed to represent all the labels of the pattern classification task at hand.

FAM stores the learned knowledge in its interconnections weights. There are two vectors of FAM weights that are worth mentioning: (a) The vector of weights $\mathbf{w}_j^a = (w_{j1}^a, w_{j2}^a,..., w_{j,2M_a}^a)$, called a *template*, whose components emanate from node *j* in F_2^a and converge to all the nodes in F_1^a ; $\mathbf{w}_j^a$ represents the group of input patterns that chose node j in the category representation layer of FAM as their representative node and this node encoded them, and (b) the vector of weights, denoted by $\mathbf{W}_j^{ab} = (W_{j1}^a, W_{j2}^a,..., W_{j,N_b}^a)$, emanating from every node *j* in the F_2^a layer of FAM and converging , to all the nodes in F_2^b . In FAM, if all the components of $\mathbf{W}_j^{ab}$ are equal to 0, except component W_{jk}^{ab} , this is an indication that node j in F_2^a is mapped to label *k* in F_2^b .

2.1.1 FAM Category Geometrical Representation

It is very important to point out that the templates in FAM (i.e., the $\mathbf{w}_j^a$'s) have an interesting geometrical interpretation. That is every template in FAM can be thought of as a hyper-rectangle, whose boundaries enclose all the input patterns that were encoded by the template during FAM's training phase. For instance, in Figure 2-2, the 2D hyper-rectangle R_j^a of template $\mathbf{w}_j^a$ is shown as including within its boundaries 7 input patterns encoded by it. As it can be seen from Figure 2-2, R_j^a is completely defined by its two endpoints (i.e. $\mathbf{u}_j^a$, $\mathbf{v}_j^a$), and it can be shown that $\mathbf{w}_j^a = (\mathbf{u}_j^a, (\mathbf{v}_j^a)^c)$.

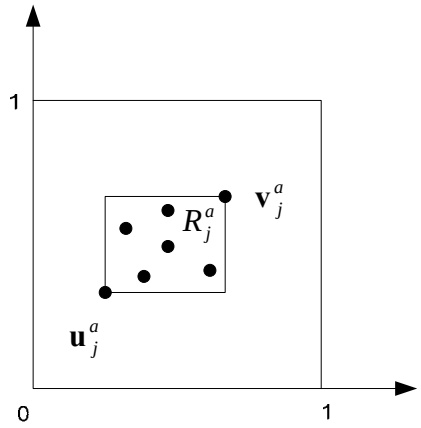

Figure 2-2: A rectangle representation of a FAM category that learned seven input patterns

To geometrically describe the FAM equations we need the concepts of the "size of a hyper-rectangle", and the distance of an input pattern $\mathbf{I}$ from the hyper-rectangle R_j^a. The size of the hyper-rectangle R_j^a (denoted by $s(\mathbf{w}_j^a)$) is equal to the L_1 norm of the vector $\mathbf{v}_j^a - \mathbf{u}_j^a$ (or in other words the sum of the lengths of all its sides). The distance of an input pattern $\mathbf{I} = (\mathbf{a}, \mathbf{a}^c)$ from a rectangle R_j^a (denoted by $dis(\mathbf{I}, \mathbf{w}_j^a)$) is equal to the minimum L_1 distance of $\mathbf{a}$ from any point of the rectangle R_j^a. Please refer to Figure 2-3 for an illustration of the size of a rectangle and the distance of an input pattern from this rectangle in the case where $M_a = 2$.

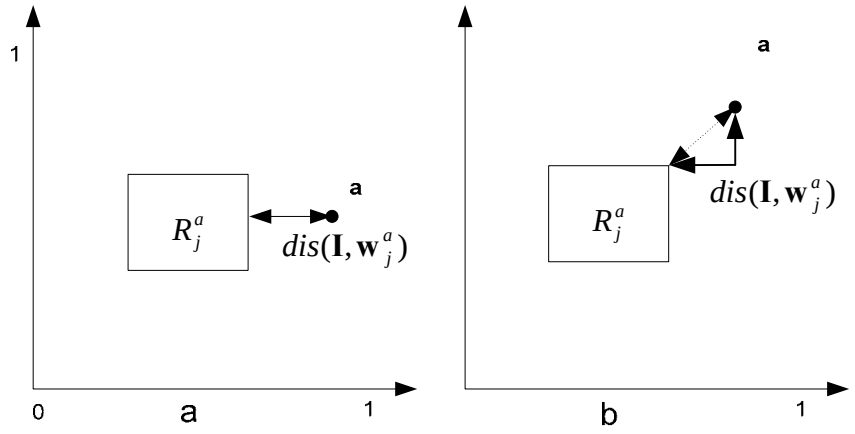

Figure 2-3: The distance of an input pattern a from the rectangle R_j^a is the minimum distance of the pattern a from the border of the rectangle R_j^a

2.1.2 FAM Operations and Parameters

FAM can operate in two distinct phases: the *training phase* and the *performance phase*. In the training phase of FAM a list of input patterns/output labels, for example, $\{(\mathbf{I}^1, O(\mathbf{I}^1)),...,(\mathbf{I}^r, O(\mathbf{I}^r)),...,(\mathbf{I}^{PT}, O(\mathbf{I}^{PT}))\}$, is repeatedly presented to FAM until FAM learns the required mapping. The task is considered accomplished (i.e. learning is complete) when the weights do not change during a list presentation or when a maximum number of list presentations is reached. The performance phase of FAM works as follows: Given a list of input patterns, such as $\tilde{\mathbf{I}}^1, \tilde{\mathbf{I}}^2,..., \tilde{\mathbf{I}}^{PS}$, we want to find the FAM output produced when each one of the aforementioned test patterns is presented at its F_1^a layer. In order to achieve the aforementioned goal we present the test list to the trained FAM architecture and we observe the network's output (i.e., label). (Appendix B gives a step-by-step procedure for FAM training and performance)

The operation of FAM is affected by two user defined network parameters, the choice parameter β_a, and the baseline vigilance parameter $\bar{\rho}_a$. The choice parameter β_a takes values in the interval $(0, \infty)$, while the baseline vigilance parameter $\bar{\rho}_a$ assumes values in the interval $[0,1]$. Both of these parameters affect the number of nodes created in the category

representation layer of FAM. The parameter β_a controls the order according to which nodes will be accessed in Fuzzy ARTMAP. At the same time the parameter β_a has an effect on how many nodes will be created in the category representation layer of Fuzzy ARTMAP during FAM's training phase (larger values of β_a tend to produce larger number of category nodes in FAM). The parameter $\overline{\rho}_a$ also has an effect on the number of nodes created in the category representation layer of Fuzzy ARTMAP (larger values of $\overline{\rho}_a$ produce larger number of nodes in the category representation layer). There are two other network parameter values in FAM that are controlled by the algorithm, namely, the vigilance parameter ρ_a, and the number of nodes N_a in the category representation layer of FAM. The vigilance parameter ρ_a takes value in the interval $[\overline{\rho}_a, 1]$ and its initial value is set to be equal to $\overline{\rho}_a$. The number of nodes N_a in the category representation layer of FAM corresponds to the number of committed nodes (nodes that have established a connection with nodes in the F_2^b layer) in FAM plus one uncommitted node. Prior to initiating the training phase of FAM, the top-down weights (the w_{ji}^a's) are chosen equal to 1, and the inter-ART weights (the W_{jk}^{ab}'s) are chosen equal to 0. There are three major operations that take place during the presentation of a training input/output pair (e.g., $(\mathbf{I}^r, \mathbf{O}^r)$) to Fuzzy ARTMAP.

Operation 1: Calculating the Category Choice Function (CCF) value

In this operation FAM calculates the category choice function (CCF) value (i.e. $T(j \mid \mathbf{I})$) for every category j in its category representation layer F_2^a, as follows:

$$T(j \mid \mathbf{I}) = \frac{M_a - s(\mathbf{w}_j^a) - dis(\mathbf{I}, \mathbf{w}_j^a)}{\beta_a + M_a - s(\mathbf{w}_j^a)} \qquad 2\text{-}2$$

After calculating the choice function values, the node *J* with the maximum choice function value proceeds to operation 2.

Operation 2: Calculating the Category Match Function (CMF) Value

The node J with the largest CCF value is examined to determine whether it passes the vigilance criterion. A node J (category) passes the vigilance criterion if its category (node) match function value (i.e., ($\rho(J \mid \mathbf{I})$) exceeds the vigilance parameter value ρ_a , that is if

$$\rho(J \mid \mathbf{I}) = \frac{M_a - s(\mathbf{w}_J^a) - dis(\mathbf{I}, \mathbf{w}_j^a)}{M_a} \geq \rho_a \qquad 2\text{-}3$$

If the vigilance criterion is passed we proceed with operation 3. Otherwise, node J is disqualified and we find the next in sequence node in F_2^a that maximizes the CCF value. Eventually we will end up with a node J that maximizes the CCF value and satisfies the vigilance criterion (notice that this could be an uncommitted node, and hence N_a (number of committed nodes in FAM) get increased by 1).

Operation 3: Match Tracking Mechanism/Change of the Weights

This operation is implemented only after we have found a node J that maximizes the CCF value of the remaining (in the competition) F_2^a nodes and passes the vigilance criterion. Operation 3 determines whether this node J passes the prediction test. The prediction test checks if the inter-ART weight vector emanating from node J (i.e. $\mathbf{W}_J^{ab} = (W_{J1}^a, W_{J2}^a, ..., W_{J,N_b}^a)$) matches exactly the desired output vector $\mathbf{O}$ (note that $\mathbf{O}$ is the output pattern that the input pattern **I** is supposed to be mapped to). If the prediction is satisfied then we say that the node "passed the prediction test". If the node does not pass the prediction test, the vigilance parameter ρ_a is increased to the level of $\frac{M_a - s(\mathbf{w}_J^a)}{M_a}$, node J is disqualified, and the next in sequence node J that maximizes the CCF value and passes the vigilance is chosen (this action is referred to as the match tracking mechanism). If node J though passes the prediction test, the weights $\mathbf{w}_J^a$ in FAM are modified as Figure 2-4 demonstrates, and according to the following equation:

$$\mathbf{w}_j^{a,new} = \mathbf{w}_j^{a,old} \wedge \mathbf{I} \qquad 2\text{-}4$$

where $\wedge$ is the fuzzy min operator, which outputs a vector whose components are equal to the minimum of the corresponding components of its arguments. And $\mathbf{W}_{j_{\max}}^{ab} = \mathbf{O}$.

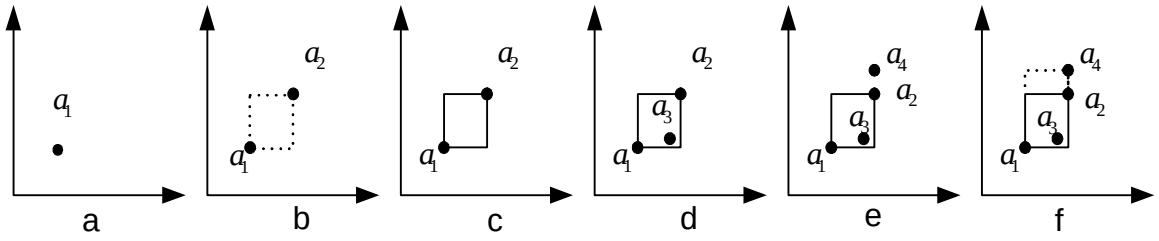

Figure 2-4: FAM Learning, a. A category with 0 size; b. Introducing a new pattern a2; c. The category expands to include a2; d. Since a3 is inside the category, it doesn't change its size; e. Pattern a4 is presented; f. Since a4 is outside the category, the category is expanded to include a4, within its boundaries.

Figure 2-4 covers all the possible learning scenarios in FAM, that is when a category learns the first input pattern, then as it learns a second input pattern, and then as it learns a third input pattern for the case where the third pattern is inside the rectangle and for the case where the third input pattern is outside the rectangle that the category defines.

FAM training is considered complete if and only if after repeated presentations of all training input/output pairs to FAM, where Operations 1-3 are recursively applied for every input/output pair, we find ourselves in a situation where a complete cycle through all the input/output pairs produced no weight changes, or if we reached a maximum number of list presentation, *through out all experiments we used 10 for this number*. In the performance phase of FAM only Operations 1 and 2 are implemented for every input pattern presented to FAM. By registering the network output to every test input presented to FAM, and by comparing it to the desired output we can calculate the network's performance (i.e. Percent Correct Classification or PCC)

2.2 Ellipsoidal ARTMAP (EAM)

Ellipsoidal ARTMAP (EAM) architecture is very similar to that of FAM. The major difference between EAM and FAM is that EAM covers the space of the input patterns with ellipsoids instead of rectangles (hyper-ellipsoids in problems with more than two dimensions). The EAM architecture, like the FAM architecture, consists of three layers or fields of nodes (Figure 2-5).

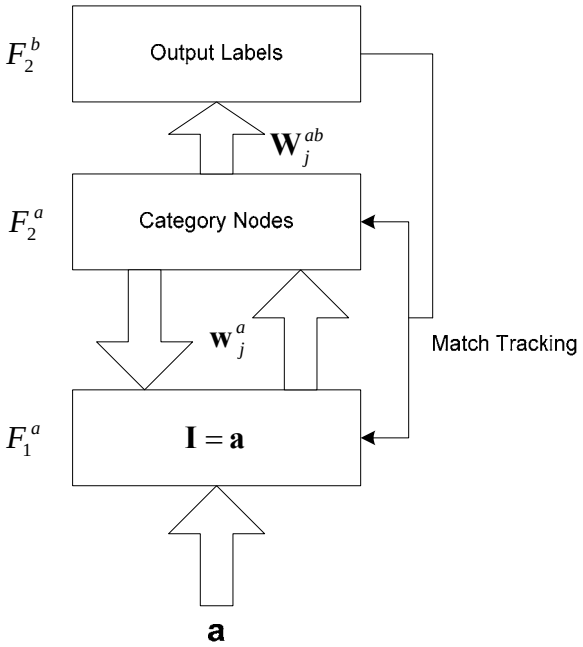

Figure 2-5: Simple EAM Architecture

These are the *input layer* (F_1^a), the *category representation layer* (F_2^a), and the *output layer* (F_2^b). The input layer of EAM is the layer where inputs are applied. An input applied to F_1^a is a vector $\mathbf{I}$ of dimensionality M_a of the following form,

$$\mathbf{I} = \mathbf{a} = (a_1,..a_i.,a_{M_a}); \quad 1 \le i \le M_a \qquad 2\text{-}5$$

where $\mathbf{a}$ is a vector whose components lie in the interval $(-\infty,+\infty)$. Thus, layer F_1^a is a layer that contains M_a nodes, one node for each component of the input pattern $\mathbf{I}$. The index i $(1 \le i \le M_a)$ designates a generic node in layer F_1^a. The layer F_2^a of EAM is referred to as the *category representation layer*, this layer contains all the categories (or groups) of input

patterns in the network. Finally, the output layer (layer F_2^b) is the layer where the outputs of the network can be found. Nodes in the output layer of EAM represent labels or classes of the pattern recognition task. The index k $(1 \le k \le N_b)$ designates a generic node in F_2^b; N_b is the number of all classes in the problem domain and so is the highest index in this classification task.

EAM as in FAM, stores the learned knowledge in its interconnection weights: (a) The vector of weights $\mathbf{w}_j^a = (\mathbf{m}_j, \mathbf{d}_j, r_j)$, called a *template*, where $\mathbf{m}_j, \mathbf{d}_j, r$ represent the center, direction, and radius of the major axis of the ellipsoid that node *j* creates, whose components emanate from node *j* in F_2^a and converge to all the nodes in F_1^a; $\mathbf{w}_j^a$ encompasses the group of input patterns that were selected by this node *j* in the category representation layer of EAM, and (b) the vector of weights, denoted by $\mathbf{W}_j^{ab} = (W_{j1}^a, W_{j2}^a, ..., W_{j,N_b}^a)$, emanating from every node *j* in the F_2^a layer of EAM and converging to all the nodes in F_2^b. In EAM, if all the components of $\mathbf{W}_j^{ab}$ are equal to 0, except component W_{jk}^{ab}, it is an indication that node *j* in F_2^a is mapped to label *k* in F_2^b.

2.2.1 EAM Category Geometrical Representation

As it was pointed out earlier, the major difference between FAM and EAM lies in the shape of their categories (FAM uses hyper-rectangles, while EAM uses hyper-ellipsoids). In both cases though the structure created that corresponds to a category (in FAM or EAM), encloses within its boundaries all the input patterns that used and were encoded by this category. In figure 2-6 a 2D hyper-ellipsoid category is shown as including within its boundaries 3 input patterns that it had encoded already. It is very obvious that the ellipsoid grows to include the patterns that it encodes.

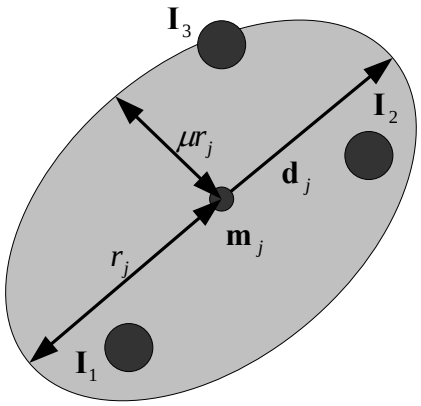

Figure 2-6: An EAM category that encodes 3 patterns

The size of an EAM category is defined as the maximum Mahalanobis distance between two patterns inside the representation region of the category, since this distance equals $2r_j$, then the size of category j is $s(\mathbf{w}_j) = 2r_j$. The distance in EAM is defined as the minimum distance between the pattern $\mathbf{I}$ and the boundaries of the category if the pattern is outside the category, otherwise the distance equals zero, and hence, the distance of an input pattern $\mathbf{I}$ from a category *j* represented by $\mathbf{w}_j^a$ is

$$dis(\mathbf{I}, \mathbf{w}_j^a) = \max\{\| \mathbf{I} - \mathbf{m}_j \|_{\mathbf{C}_j}, r_j\} - r_j \qquad 2\text{-}6$$

$$\text{and } \| \mathbf{I} - \mathbf{m}_j \|_{\mathbf{C}_j} = \frac{1}{\mu}\sqrt{\| \mathbf{I} - \mathbf{m}_j \|_2^2 - (1-\mu^2)\left[\mathbf{d}_j{}^T (\mathbf{I} - \mathbf{m}_j)\right]^2} \qquad 2\text{-}7$$

where $\mathbf{C}_j$ is the shape matrix of an EAM category j, μ is the ratio of the minor axis of the hyper-ellipsoid to its major axis and $\|.\|_2$ is the Euclidian (L_2) norm of its argument vector.

2.2.2 EAM Operations and Parameters

EAM can operate in two distinct phases: the *training phase* and the *performance phase*. In the training phase of EAM/EAM a list of input patterns/output labels, for example, $\{(\mathbf{I}^1, O(\mathbf{I}^1)),...,(\mathbf{I}^r, O(\mathbf{I}^r)),...,(\mathbf{I}^{PT}, O(\mathbf{I}^{PT}))\}$, is repeatedly presented to EAM until EAM learns the required mapping. The task is considered accomplished (i.e., the learning is

complete) when the weights do not change during a list presentation or a user defined maximum number of list presentations is reached. The performance phase of EAM works as follows: Given a list of input patterns, such as $\tilde{\mathbf{I}}^1, \tilde{\mathbf{I}}^2, ..., \tilde{\mathbf{I}}^{PS}$, we would like to find the EAM output produced when each one of the aforementioned test patterns is presented at its F_1^a layer. By presenting the test list to the trained EAM architecture and observing the network's output (i.e., label) the aforementioned goal could be reached.

Few parameters affect the operation of EAM. Among these parameters are (β_a (choice parameter) and $\overline{\rho}_a$ (baseline vigilance parameter)) that affect the performance of FAM as well. The other two are: μ, which is the minor-to-major axis length ratio (common for every EAM category), and *D* which corresponds to FAM's M_a variable and it should be greater than zero. The choice and vigilance parameters affect the number of nodes created in the category representation layer of FAM. The parameter β_a controls the order according to which nodes will be accessed in EAM. At the same time the parameter β_a has an effect on how many nodes will be created in the category representation layer of EAM during EAM's training phase (larger values of β_a tend to produce larger number of category nodes in EAM). The parameter $\overline{\rho}_a$ also has an effect on the number of nodes created in the category representation layer of Ellipsoidal (larger values of $\overline{\rho}_a$ produce larger number of nodes in the category representation layer). The parameter μ ranges from 0 to 1, and finally D is chosen equal to $\frac{\sqrt{M_a}}{\mu}$. Note, that the parameter D also affects the number of nodes created in the category representation layer of EAM (smaller values of *D* tend to produce more nodes in the category representation layer of EAM, and consequently result in less compression of the input patterns presented in EAM). There are two other network parameters in EAM that the networking uses internally, these are the vigilance parameter ρ_a, and the number of nodes

N_a in the category representation layer of EAM. The vigilance parameter ρ_a takes value in the interval $[\overline{\rho}_a, 1]$ and its initial value is set to be equal to $\overline{\rho}_a$. The number of nodes N_a in the category representation layer of EAM corresponds to the number of committed nodes in EAM plus one uncommitted node. Prior to initiating the training phase of EAM, $\mathbf{m}_j, \mathbf{d}_j, r_j$ are chosen equal to 0, and the inter-ART weights (the W_{jk}^{ab}'s) are chosen equal to 0 too. There are three major operations that take place during the training phase of Ellipsoid ARTMAP.

Operation 1: Calculating the Category Choice Function (CCF) value

To select the wining category the network should calculate the category (node) choice function (CCF) value (i.e., $T(j \mid \mathbf{I})$) for every node (category) j in F_2^a, which is performed as follows:

$$T(j \mid \mathbf{I}) = \frac{D - s(\mathbf{w}_j^a) - dis(\mathbf{I}, \mathbf{w}_j^a)}{\beta_a + D - s(\mathbf{w}_j^a)} \qquad 2\text{-}8$$

After calculating the choice function values for every node *j* in the category representation layer of EAM, the node *J* with the maximum choice function value is chosen, and EAM proceeds with operation 2.

Operation 2: Calculating the Category Match Function (CMF) Value

After finding the node J with the largest CCF value, this node is examined to determine whether it passes the vigilance criterion. A node *J* (category) passes the vigilance criterion if its category (node) match function value (i.e., ($\rho(J \mid \mathbf{I})$) exceeds the vigilance parameter value ρ_a, that is if

$$\rho(j \mid \mathbf{I}) = \frac{D - s(\mathbf{w}_j^a) - dis(\mathbf{I}, \mathbf{w}_j^a)}{D} > \rho_a \qquad 2\text{-}9$$

If the node passes the vigilance criterion the third operation starts. Otherwise, node J is disqualified and we find the next in sequence node in F_2^a that maximizes the CCF value. Eventually we will end up with a node J that maximizes the CCF value and satisfies the vigilance criterion, which could be an uncommitted node.

Operation 3: Match Tracking Mechanism/Change of the Weights

Operation 3 determines whether the chosen node J passes the prediction test. The prediction test is simply a comparison between the inter-ART weight vector emanating from node J (i.e. $\mathbf{W}_J^{ab} = (W_{J1}^a, W_{J2}^a, ..., W_{J,N_b}^a)$) and the desired output vector $\mathbf{O}$. If they are equal, this is referred to as the node "passing the prediction test". If the node does not pass the prediction test, the vigilance parameter ρ_a is increased to the level of $\frac{D - s(\mathbf{w}_J^a)}{D}$, node J is disqualified, a new search for the next node J that maximizes the CCF value and passes the vigilance starts (this action is referred to as the match tracking mechanism). If node J passes the prediction test, the weights $\mathbf{w}_J^a$ in EAM are modified according to the following equations and explained pictorially in figure 2-7.

$$\mathbf{m}_j^{new} = \mathbf{m}_j^{old} + \frac{\gamma}{2}\left(1 - \frac{\min\left\{r_j^{old}, \| \mathbf{I} - \mathbf{m}_j^{old} \|_{C_j^{old}}\right\}}{\| \mathbf{I} - \mathbf{m}_j^{old} \|_{C_j^{old}}}\right)(\mathbf{I} - \mathbf{m}_j^{old}) \qquad 2\text{-}10$$

and

$$r_j^{new} = r_j^{old} + \frac{\gamma}{2}\left(\max\left\{r_j^{old}, \| \mathbf{I} - \mathbf{m}_j^{old} \|_{C_j^{old}}\right\} - r_j^{old}\right) \qquad 2\text{-}11$$

Where γ is the learning factor, in our experiments we used, fast learning (i.e. γ =1).

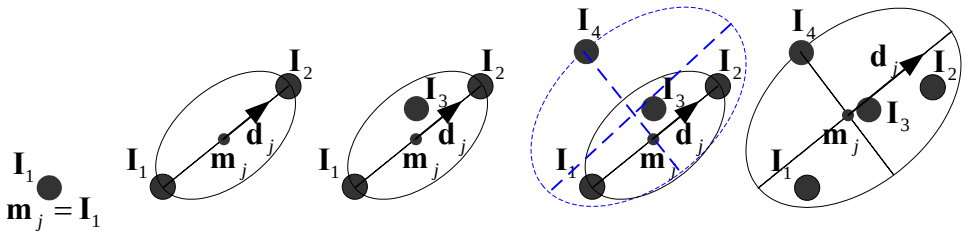

Figure 2-7: Creation and expansion of an EAM category

Figure 2-7 covers all the possible scenarios when node *J* first learns the first input pattern, then as it learns a second input pattern, and then as it learns a third input pattern, for the case where the third pattern is inside the ellipsoid, the ellipsoid does not get updated and hence stays the same.

EAM training is considered complete if and only if after repeated presentations of all training input/output pairs to EAM, where Operations 1-3 are recursively applied for every input/output pair, we find ourselves in a situation where a complete cycle through all the input/output pairs produced no weight changes. As in FAM, in the performance phase of EAM only Operations 1 and 2 are implemented for every input pattern presented to EAM. By registering the network output to every test input presented to EAM, and by comparing it to the desired output we can calculate the network's performance (i.e. Percent Correct Classification or PCC)

2.3 Gaussian ARTMAP (GAM)

Similar to FAM and EAM, the GAM architecture consists of three layers or fields of nodes (see Figure 2-5), The *input layer* (F_1^a), the *category representation layer* (F_2^a), and the *output layer* (F_2^b). The input layer of GAM is the layer where inputs are applied. An input pattern applied to F_1^a is a vector $\mathbf{I}$ (of dimensionality M_a) of the following form,

$$\mathbf{I} = \mathbf{a} = (a_1, ..a_i ., a_{M_a}); \quad 1 \le i \le M_a \qquad 2\text{-}12$$

Where **a** is a vector whose components lie in the interval $(-\infty,+\infty)$. Thus, layer F_1^a is a layer that contains M_a nodes, one node for each component of the input pattern **I**. The index *i* $(1 \le i \le M_a)$ designates a generic node in layer F_1^a. The layer F_2^a of GAM is referred to as the *category representation layer*, because this is where categories (or groups) of input patterns are formed. Finally, the output layer (layer F_2^b) is the layer that produces the outputs of the network. Every node in the output layer of GAM represents one of the labels of the pattern recognition task. The index *k* $(1 \le k \le N_b)$ designates a generic node in F_2^b; N_b represents the number of the different classes in the problem, and is the highest index needed to represent all the labels of the pattern classification task at hand.

Using its interconnections weights, GAM stores the learned knowledge. There are two types of GAM weights: (a) The vector of weights $\mathbf{w}_j^a = (\boldsymbol{\mu}_j, \boldsymbol{\sigma}_j, n_j)$, called a *template*, whose components emanate from node *j* in F_2^a and converge to all the nodes in F_1^a. The vector $\boldsymbol{\mu}_j$ is an M_a-dimensional vector, with components equal to the average values of the components of the input patterns that accessed and were encoded by category *j*. The vector $\boldsymbol{\sigma}_j$ is an M_a-dimensional vector, with components equal to the standard deviation of the components of the input patterns that accessed and were encoded by category *j*. Finally, n_j is a scalar that is equal to the number of input patterns that accessed and were encoded by category *j* in GAM, and (b) the vector of weights, denoted by $\mathbf{W}_j^{ab} = (W_{j1}^a, W_{j2}^a, ..., W_{j,N_b}^a)$, emanating from every node *j* in the F_2^a layer of GAM and converging , to all the nodes in F_2^b. In GAM, if all the components of $\mathbf{W}_j^{ab}$ are equal to 0, except component W_{jk}^{ab}, it is an indication that node j in F_2^a is mapped to label *k* in F_2^b.

Contrary to FAM and EAM, GAM does not create enclosed structures (such as hyper-rectangles or hyper-ellipsoids) that contain within their boundaries all the input patterns that

were encoded by these structures. In GAM a category is represented by a Gaussian (bell-shaped) curve, whose mean vector and the standard deviation vector corresponds to the mean and standard deviation vector of all the input patterns that chose and were encoded by this category in GAM's training phase. Furthermore, every GAM category has another parameter n_j, associated with it, that is equal to the number of patterns that were encoded by this category, and as such defines how important this bell-shaped curve is in representing the input patterns that are presented to GAM. For instance, see Figure 2-8, where a few input patterns are shown (in one dimension) and the Gaussian curve that they define is depicted.

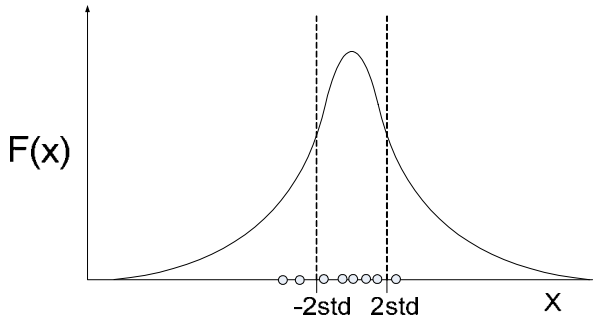

Figure 2-8: A 2D GAM category that encodes 5 patterns within 2 standard deviations

2.3.1 GAM Operations and Parameters

In the training phase of GAM a list of input patterns/output labels, for example, $\{(\mathbf{I}^1, O(\mathbf{I}^1)),...,(\mathbf{I}^r, O(\mathbf{I}^r)),...,(\mathbf{I}^{PT}, O(\mathbf{I}^{PT}))\}$, is repeatedly presented to GAM until GAM learns the required mapping. The task is considered done (i.e., the learning is complete) when a user defined maximum number of presentations is reached. The performance phase of GAM works as follows: We have a list of input patterns, such as $\tilde{\mathbf{I}}^1, \tilde{\mathbf{I}}^2,..., \tilde{\mathbf{I}}^{PS}$, and we want to find the GAM output produced when each one of the aforementioned test patterns is presented at its F_1^a layer of GAM. To attain the aforementioned goal we present the test list to the trained GAM architecture and we observe the network's output (i.e., label).

The operation of GAM is affected by two network parameters, the initial standard deviation parameter γ, and the baseline vigilance parameter $\bar{\rho}_a$. When a GAM category

encodes an input pattern for the first time its standard deviation vector of the Gaussian curve that it defines has components that are all equal to the initial standard deviation parameter γ. Both parameters γ and the baseline vigilance parameter $\overline{\rho}_a$ assume values in the interval [0, 1]. Both of these parameters affect the number of nodes created in the category representation layer of GAM. Higher values of $\overline{\rho}_a$ create more nodes in the category representation layer of Gaussian ARTMAP, and consequently produce less compression of the input patterns. There are two other network parameter values in GAM that are worth mentioning, such as the vigilance parameter ρ_a, and the number of nodes N_a in the category representation layer of GAM. The vigilance parameter ρ_a takes value in the interval $[\overline{\rho}_a, 1]$ and its initial value is set to be equal to $\overline{\rho}_a$. The number of nodes N_a in the category representation layer of GAM corresponds to the number of committed nodes in GAM plus one uncommitted node. Prior to initiating the training phase of GAM μ_j is set to 0's and σ_j is chosen equal to γ, and the inter-ART weights (the W_{jk}^{ab}'s) are chosen equal to 0. There are three major operations that take place during the presentation of a training input/output pair (e.g., $(\mathbf{I}^r, \mathbf{O}^r)$) to Gaussian ARTMAP.

Operation 1: Calculating the Category Choice Function (CCF) value

Calculation of the category (node) choice function value (i.e., $T(J \mid \mathbf{I})$) for every node (category) j in F_2^a, is as follows:

$$P(j \mid I) = \frac{P(I / j)P(j)}{P(I)} \qquad 2\text{-}13$$

where *P(I|j)* is the conditional density of I given j equals

$$P(I \mid j) = \frac{1}{(2\pi)^{M/2} \prod_{t=1}^{M} \sigma_{ji}} \exp\left(-\frac{1}{2} \sum_{i=1}^{M} \left(\frac{\mu_{ji} - I_i}{\sigma_{ji}} \right)^2 \right) \qquad 2\text{-}14$$

and the priori probability of j is

$$P(j) = \frac{n_j}{\sum_{j=1}^{N} n_j} \qquad 2\text{-}15$$

where N is the number of categories in the system. Since $P(I)$ is the same for all categories it is ignored and hence the equation of $T(j \mid \mathbf{I})$ is

$$T(j \mid \mathbf{I}) = \log\left((2\pi)^{M/2} P(I \mid j) P(j)\right) = -\frac{1}{2}\sum_{i=1}^{M}\left(\frac{\mu_{ji} - I_i}{\sigma_{ji}}\right)^2 - \log\left(\prod_{i=1}^{M}\sigma_{ji}\right) + \log(P(j)) \qquad 2\text{-}16$$

After calculation of the choice commitment function values the node J with the maximum choice commitment function value is chosen.

Operation 2: Calculating the Category Match Function (CMF) Value

The node J with the largest CCF value is examined to determine whether it passes the vigilance criterion. A node J (category) passes the vigilance criterion if its category (node) match function value (i.e., ($\rho(J \mid \mathbf{I})$) exceeds the vigilance parameter value ρ_a, that is if

$$\rho(j \mid \mathbf{I}) = \log\left((2\pi)^{M/2}\left(\prod_{i=1}^{M}\sigma_{ji}\right)P(I \mid j)\right) > \rho_a \qquad 2\text{-}17$$

If the vigilance criterion is passed we proceed with operation 3. Otherwise, node J is disqualified and we find the next in sequence node in F_2^a that maximizes the CCF value. Eventually we will end up with a node J that maximizes the CCF value and satisfies the vigilance criterion.

Operation 3: Match Tracking Mechanism/Change of the Weights

This operation is implemented only after we have found a node J that maximizes the CCF value of the remaining (in the competition) F_2^a nodes and passes the vigilance criterion. Operation 3 determines whether this node J passes the prediction test. The prediction test checks if the inter-ART weight vector emanating from node J (i.e., $\mathbf{W}_J^{ab} = (W_{J1}^a, W_{J2}^a, ..., W_{J,N_b}^a)$) matches exactly the desired output vector $\mathbf{O}$ (if it does, this is

referred to as the node "passing the prediction test"). If the node does not pass the prediction test, the vigilance parameter ρ_a is increased to the level of $g_J^{'}(I)$, node J is disqualified, and the next in sequence node J that maximizes the CCF value and passes the vigilance is chosen (this action is referred to as match tracking mechanism). If node J though passes the prediction test, the weights $\mathbf{w}_J^a$ in GAM are modified in a way that includes this pattern in the category. The algebraic equations that define the learning that $\mathbf{w}_J^a$ undergoes are as follows:

$$n_J := n_J + 1 \qquad \text{2-18}$$

$$\mu_J := (1 - n_J^{-1})\mu_J + n_J^{-1} I \qquad \text{2-19}$$

$$\sigma_{Ji} := \begin{cases} \sqrt{(1 - n_J^{-1})\sigma_{Ji}^2 + n_J^{-1}(\mu_{Ji} - I_i)^2} & \text{if } n_J > 1 \\ \gamma & \text{otherwise} \end{cases} \qquad \text{2-20}$$

GAM training is considered complete if and only if after repeated presentations of all training input/output pairs to GAM, where Operations 1-3 are recursively applied for every input/output pair, we reach the maximum number of list presentations defined by the user. In the performance phase of GAM only Operations 1 and 2 are implemented for every input pattern presented to GAM. By registering the network output to every test input presented to GAM, and by comparing it to the desired output we can calculate the network's performance (i.e. Percent Correct Classification or PCC)

2.4 Genetic Algorithms

One of the most widely used EA's techniques is the Genetic Algorithm. Experiments have showed that GAs are very powerful searching techniques. They were used successfully in many areas, such as optimization (Annie S. Wu, et al., 2004), scheduling (Annie Wu, Han Yu, et al., 2004), pattern recognition (Auwatanamongkol, S., 2000), robot control (Davidor

Y., 1990), and many others. The implementation of a GA follows the same steps of the general framework for EA's. A GA has the following components:

- **Population of Chromosomes:** Chromosomes represent solutions to the problem at hand. Chromosomes can be encoded in many different ways the most famous of which is the binary encoding.
- **Fitness Function:** Used to evaluate the chromosomes, where each chromosome's evaluation determines how close or far this solution is from the optimal.
- **Selection Function:** Used to select parent chromosomes (exploit partial solutions).
- **Genetic Operators:** Exchange and modify knowledge amongst the selected parent chromosomes to create new offspring (explores the search space).

2.4.1 Chromosome Representation

Many different chromosome representation approaches have been introduced in the literature, of which the following are the most common (Mitchell T., 1997):

- Binary: Uses a string of bits to represent the solution, each gene is represented by its equivalent binary code and all genes are then arranged in one string of bits.
- Location Independent: a. Messy: uses a string of cells, each cell consists of the location and the value of the bit. b. Floating Representation: uses building blocks that consist of a tag and a body where the tag is a building block identifier and the body is the actual building block (a group of bits adjacent to the tag).
- Redundant: Every building block is repeated many times in the chromosome.
- Non-Coding: In this scheme a chromosome consists of building blocks along with a set of non-coding blocks.
- Integer: Uses integer instead of binary numbers.
- Floating Point: Uses real numbers instead of binary numbers.

- Problem Specific: Uses structures that are problem dependent, and it could use any of the above representations or it might use a mixture of any of them.

All of the above representations have advantages and disadvantages (the interested reader could refer to (Mitchell T., 1997) to get more information about the pros and cons of each representation method).

2.4.2 Genetic Operators

The purpose of genetic operators is to explore new regions in the solution space by discovering new solutions that preserve some of the current knowledge. Two well known operators are used in GA:

a. **Crossover:** A method to exchange information among two (or more) parent chromosomes. In order to produce an offspring (or more) from the parents, it uses random numbers to select crossing points. Three different types of crossover are common:

   - **One-point:** Selects a crossing point at random from each parent and exchanges the subsections of both chromosomes after the crossover point.

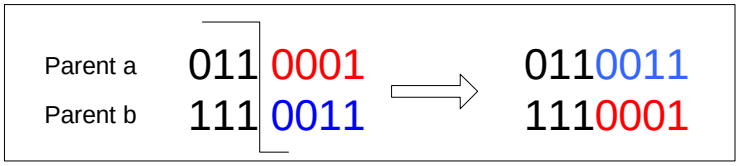

Figure 2-9: One-point crossover

   - **Two-point/multi-point:** Selects two/more random crossover points and exchanges subsections of both chromosomes (see Example in Figure 2-10). If more than two crossover points are chosen the same principle applies.

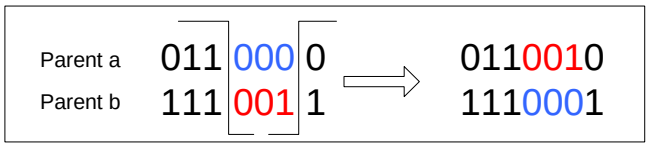

Figure 2-10: Two-point crossover

- **Uniform:** This method uses a pre-specified probability that each bit will be swapped, which is similar to using a mask of bits where 0 means no swap and 1 means swap or vice versa.

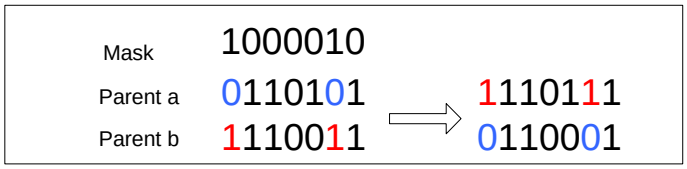

Figure 2-11: Uniform crossover

Each of the above types has advantages and disadvantages (for more information see Mitchell T., 1997).

b. **Mutation:** Mutation is a method of creating new offspring by modifying the parent. Binary mutation is quite simple, it is done by flipping a bit from 0 to 1, or the other way around, according to a specific probability. Floating point mutation can be accomplished in more ways than one, one of them is to add a randomly selected number (could be from a Gaussian Distribution) to a randomly selected gene. It is important to mention that low mutation rate results in less exploration, while high mutation rate could be disruptive.

The crossover and mutation operators have many variations reported in the literature (see T.Ba¨ck, et al., 1997, and T. Mitchell, 1997). Also it is important to mention that some other genetic operators, beyond crossover and mutation, were used in the GA literature (for an example see Liang-Hsuan, et al., 2002).

2.4.3 Selection

Selection is the process of choosing parents to create new offspring from. This process directs the search toward the promising areas of the solution surface and is usually based on the fitness of the individual chromosomes. Many selection methods were introduced in the literature, some of which are reported below:

- Fitness Proportional Selection: every individual gets a probability of being selected based on the ratio of its fitness to the average fitness of all individuals.
- Stochastic Universal Sampling: divides a wheel spin into N equally spaced markers, and then uses only one randomly generated number to select the parent.
- Sigma Scaling: The value of the fitness is modified according to the following equation $f' = f - (\bar{f} - c * \sigma)$ (where f is the original fitness value, $\bar{f}$ is the mean, σ is the standard deviation and c is a constant) to either eliminate a large difference in the fitness values or to show an un clear difference in the fitness values, and hence, maintains selection pressure over the length of a run, thus minimizing the affects of convergence on reproductive selection.
- Rank Selection: Ranks the individuals according to their fitness and then calculate the number of offspring based on the rank rather than the fitness.
- Tournament Selection: Selects at random two individuals, and then generates another random number. If this random number is greater than a specific value choose the individual with higher fitness, otherwise choose the individual with less fitness. In this approach more than two individuals could be selected.
- Elitism: Preserve the best performers from the current generation, and then uses another selection method to generate the rest of the individuals.

The interested reader can find more information in J. H. Holland, 1975, Baker, J., 1987 and (Mitchell T., 1997 about these and other selection strategies.

3. GENETIC FUZZY ARTMAP (GFAM)

GFAM (Genetic Fuzzy ARTMAP) is an evolved FAM network that is produced by applying, repeatedly, genetic operators on an initial population of trained FAM networks. GFAM uses tournament selection with elitism, as well as genetic operators, including crossover and mutation. In addition, GFAM uses two special operators, Cat_{add} and Cat_{del}.

To better understand how GFAM is designed we resort to a step-by-step description of this design. It is instructive though to first introduce some terminology that is included in Appendix A. The design of GFAM can be articulated through a sequence of steps, defined succinctly below, and explained in detail later.

Step 1: Initialize Pop_{size} number of FAM networks, each one of them operating with a different value for the baseline vigilance parameter $\overline{\rho}_a$, and different orders of training pattern presentation.

Step 2: Train each one of the Pop_{size} initialized FAM networks, using the training set for a maximum number of iterations.

Step 3: Convert the Pop_{size} trained FAM networks into chromosomes. Crop all chromosomes so that no one-point categories exist.

Step 4: Evolve the chromosomes of the current generation by executing the following sub-steps:

Sub-Step 4a: Calculate the fitness for all chromosomes of the current generation.

Sub-Step 4b: Initialize an empty generation (referred to as *temporary generation*).

Sub-Step 4c: Move the NC_{best} best chromosomes from the current generation to the temporary generation.

Sub-Step 4d: Select chromosomes for crossover from the current generation to populate the remainder of the temporary generation.

Sub-Step 4e: With a probability $P(Cat_{add})$ apply the Cat_{add} operator to every individual generated in sub-step 4d.

Sub-Step 4f: With a probability $P(Cat_{del})$ apply the Cat_{del} operator to every individual generated in sub-step 4e.

Sub-Step 4g: With a probability $P(Mut)$ apply the mutation operator to every individual generated in sub-step 4f.

Sub-Step 4h: Replace the current generation with the members of the temporary generation

Step 5: If evolution has reached the maximum number of iterations, $Gen_{\max}$, then calculate the performance of the best-fitness FAM network on the test set and report classification accuracy and number of categories that this best-fitness FAM network possesses. If the maximum number of iterations has not been reached yet, go to step 4 to evolve one more population of chromosomes

Each one of the aforementioned steps of the algorithm is now described in more detail, as needed.

Step 1 (More Details): The algorithm starts by training Pop_{size} FAM networks, each one of them trained with a different value of the baseline vigilance parameter $\overline{\rho}_a$. In particular, we first define $\overline{\rho}_a^{inc} = \dfrac{\overline{\rho}_a^{\max} - \overline{\rho}_a^{\min}}{Pop_{size} - 1}$, and then the baseline vigilance parameter of every network is determined by the equation $\overline{\rho}_a^{\min} + i * \overline{\rho}_a^{inc}$, where $i \in \{0, 1, ..., Pop_{size} - 1\}$. In our experiments with GFAM we chose $\overline{\rho}_a^{\min} = 0.1$, and $\overline{\rho}_a^{\max} = 0.95$. Meanwhile, GFAM allows the user to change the order of pattern presentation automatically and randomly.

Step 2 (More Details): We assume that the reader is familiar with how training of FAM networks is accomplished, and thus the details here are omitted.

Step 3 (More Details): Once the Pop_{size} networks are trained, they need to be converted to chromosomes so that they can be manipulated by the genetic operators. GFAM uses a real number representation to encode the networks. Each FAM chromosome consists of two levels, level 1 containing all the categories of the FAM network, and level 2 containing the lower and upper endpoints of every category in level 1, as well as the label of that category (see Figure 3-1). We denote the category of a trained FAM network with index p $(1 \le p \le Pop_{size})$ by $\mathbf{w}_j^a(p)$, where $\mathbf{w}_j^a(p) = (\mathbf{u}_j^a(p), (\mathbf{v}_j^a(p)^c)$ and the label of this category by $l_j(p)$ for $1 \le j \le N_a(p)$.

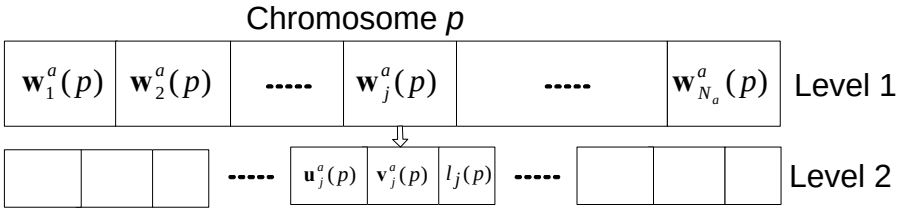

Figure 3-1: GFAM chromosome structure

In this step, we eliminate all single-point categories in the trained FAM networks, referred to as cropping the chromosomes. Since our ultimate objective is to design a FAM network that reduces the network size and improves generalization we discourage at this stage the creation of single-point categories. Our experiments have shown that cropping single-point categories is beneficial because it speeds-up the convergence of the GA.

Step 4 (More Details): In this step the GFAM applies a GA to the population of trained FAMs.

Sub-step 4a (More Details): Calculate the fitness of each chromosome (trained FAM). This is accomplished by feeding into each trained FAM the validation set and by calculating the percentage of correct classification exhibited by each one of these trained FAM networks. In particular, if $PCC(p)$ designates the percentage of correct classification, exhibited by the p-th

FAM, and this FAM network possesses $N_a(p)$ nodes in its category representation layer, then its fitness function value is defined by:

$$Fit(p) = \frac{(Cat_{\max} - N_a(p)) \cdot PCC^2(p)}{\frac{100}{Cat_{\min}} - \frac{PCC(p)}{N_a(p)} + \varepsilon} \qquad 3\text{-}1$$

where, $Cat_{\min}$ and $Cat_{\max}$ are the minimum and maximum number of categories that a FAM network is allowed to have during the evolutionary process ($Cat_{\min}$ is chosen equal to 1, or equal to the number of classes in the classification problem under consideration, while $Cat_{\max}$ is chosen to be a relatively large number for the classification problem at hand). The constant ε in the denominator of the above equation is a small positive constant and it is needed to make sure that the denominator would not be zero in the case when $N_a(p) = Cat_{\min}$ and $PCC(p) = 100$.

Sub-step 4b (More Details): Obvious, no further explanations are needed.

Sub-step 4c (More Details): The algorithm searches for the best NC_{best} chromosomes from the current generation and copies them to the temporary generation.

Sub-step 4d (More Details): The remaining $Pop_{size} - NC_{best}$ chromosomes in the temporary generation are created by crossing over pairs of parents from the current generation. The parents are chosen using a deterministic tournament selection, as follows: Randomly select two groups of four chromosomes each from the current generation, and use as a parent, from each group, the chromosome with the best fitness value in the group. If it happens that from both groups the same chromosome is chosen then we choose from one of the groups the chromosome with the second best fitness value. If two parents with indices p, p' are crossed over two random numbers n, n' are generated from the index sets $\{1, 2, ..., N_a(p)\}$ and $\{1, 2, ..., N_a(p')\}$, respectively. Then, all the categories with index greater than index n' in the chromosome with index p' and all the categories with index less

than index n in the chromosome with index p are moved into an empty chromosome within the temporary generation. Notice that crossover is done on level 1 of the chromosome. This operation is pictorially illustrated in Figure 3-2.

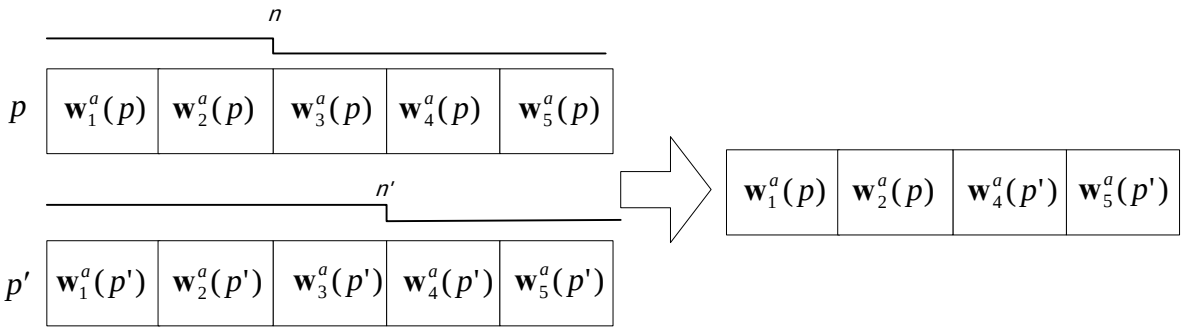

Figure 3-2: Crossover implementation

Sub-step 4e (More Details): The operator Cat_{add} adds a new category to every chromosome created in step 4d with probability $P(Cat_{add})$. The new category has lower and upper endpoints **u**, **v** that are randomly generated as follows: For every dimension of the input feature space (M_a dimensions total) we generate two random numbers uniformly distributed in the interval [0, 1]; the smallest of the two random numbers is associated with the **u** coordinate along this dimension., while the largest of these numbers is associated with the **v** coordinate along this dimension. The label of this newly created category is chosen randomly amongst the N_b categories of the pattern classification task under consideration. A chromosome does not add a category if the addition of this category causes the number of categories for this chromosome that exceed the designated maximum number of categories, $Cat_{\max}$.

Sub-step 4f (More Details): The operator Cat_{del} deletes one of the categories of every chromosome created in step 4e with probability $P(Cat_{del})$. A chromosome does not delete a

category if the deletion of this category results in the number of categories for this chromosome to fall below the designated minimum number of categories $Cat_{\min}$.

Sub-Step 4g (More Details): In GFAM, every chromosome created by step 4f gets mutated as follows: with probability $P(mut)$ every category is mutated. If a category is chosen to be mutated, either its **u** or **v** endpoints is selected randomly (50% probability) and then every component of this selected vector gets mutated by adding to it a small number. This number is drawn from a Gaussian distribution with mean 0 and standard deviation 0.01. If the component of the chosen vector becomes smaller than 0 or greater than 1 (after mutation), it is set back to 0 or 1, respectively. Notice that mutation is applied to level 2 of the chromosome structure. The label of the chromosome is not mutated because our initial GA population consists of trained FAMs, and consequently we have a lot of confidence in the labels of the categories that these trained FAMs have discovered through the FAM training process.

Sub-Step 4h (More Details): Obvious, no more details are needed.

Step 5 (More Details): Obvious, no more details are needed.

3.1 Justification of the Evolutionary Choices for GFAM

3.1.1 Justification of the Fitness Function Choice for GFAM

We have chosen to use a fitness function that is provided by equation 3-1:

$$Fit(p) = \frac{(Cat_{\max} - N_a(p)) \cdot PCC^2(p)}{\frac{100}{Cat_{\min}} - \frac{PCC(p)}{N_a(p)} + \varepsilon}$$

As a reminder, $Cat_{\max}$ is the maximum number of categories that an evolved FAM can have, and $Cat_{\min}$ is the minimum number of categories that an evolved FAM can have. The parameter $Cat_{\min}$ is chosen to be equal to 1. It seems that a more natural choice would have been to choose $Cat_{\min}$ equal to the number of different classes in the problem at hand;

however this choice, although it makes GFAM converge faster to a solution, it occasionally compromises the generalization. The parameter $PCC(p)$ is the percentage of correct classification of an evolved FAM on the validation set, and $N_a(p)$ is the actual number of categories of the evolved FAM. Finally, ε is a small positive number.

The chosen fitness function has a number of good properties. First, it depends on both measures of performance, size of the FAM network and accuracy on the validation set. It depends on the accuracy in a way that higher accuracy leads us to larger fitness values, as figures 3-3a and 3-3b to 3-3e demonstrate. It depends on size in a way that small size leads us to larger fitness values, everything else kept fixed, as figures 3-3a and 3-3f to 3-3i demonstrate. Note that if the size decreases the numerator of the fitness increases and the denominator decreases, provided that everything else is kept fixed. Similarly, if the accuracy increases the numerator increases and the denominator decreases, provided that everything else is kept fixed. It is also worth noting that when the size is equal to the minimum size and the accuracy is equal to the highest accuracy the denominator of the fitness function practically approaches zero and the fitness function assumes a very high value as plots 3-3b through 3-3e illustrate. Hence, the fitness function shows a strong preference towards the creation of minimum size and highest accuracy networks, as it should. Finally, in addition to the 2-d plots in figures 3-3f to 3-3i (plots of fitness versus accuracy for different size networks), and the 2-d plots in figures 3-3f to 3-3i (plots of fitness versus size for different accuracy networks), a 3-d plot that shows the fitness values as both size and accuracy are changing is provided in Figure 3-3a.

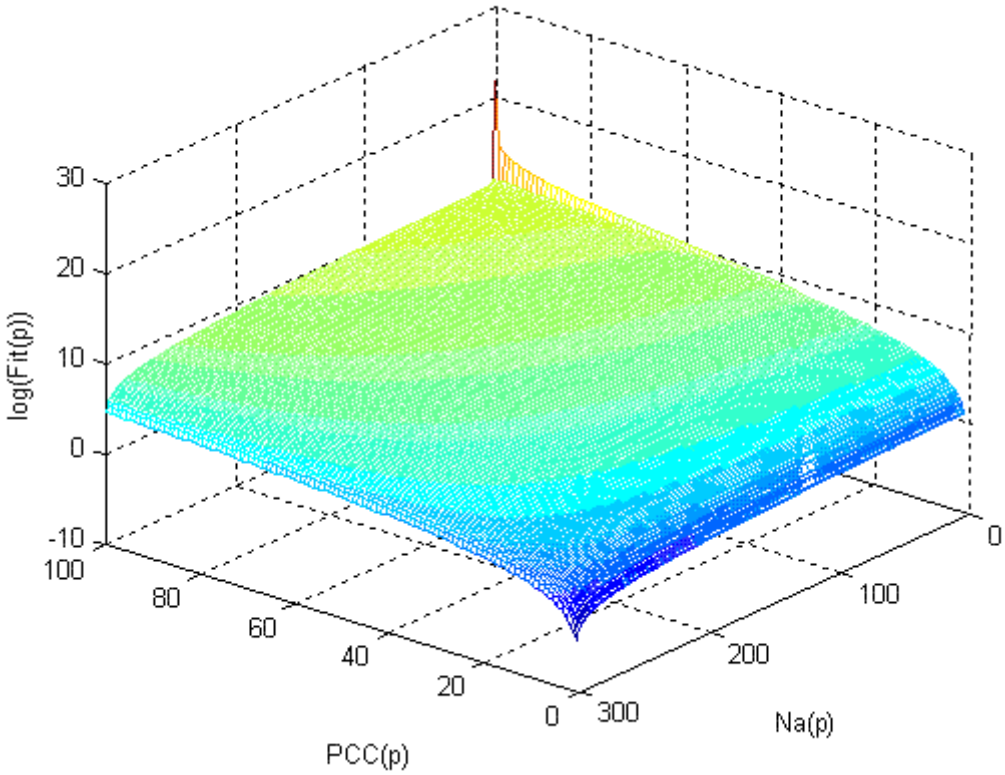

Figure3-3a: 3D plot of log(fit(p))

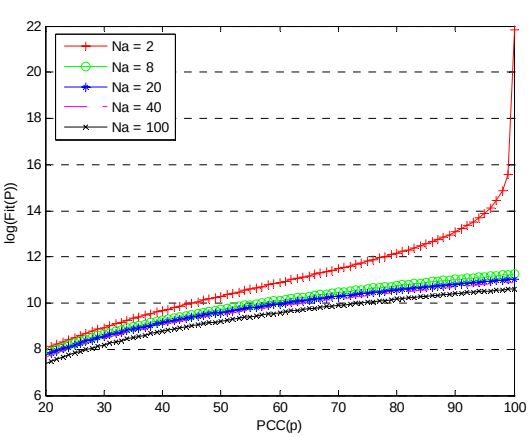

Figure 3-3b: CATmin = 2

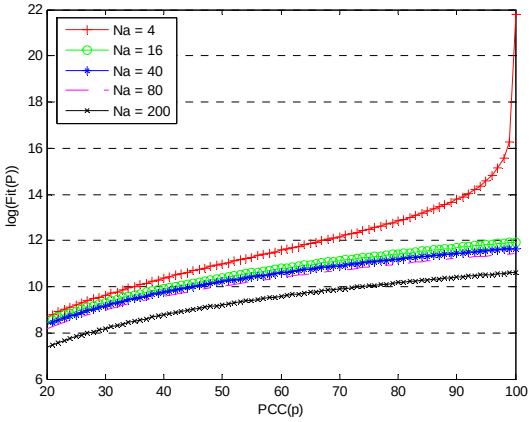

Figure 3-3c: CATmin = 4

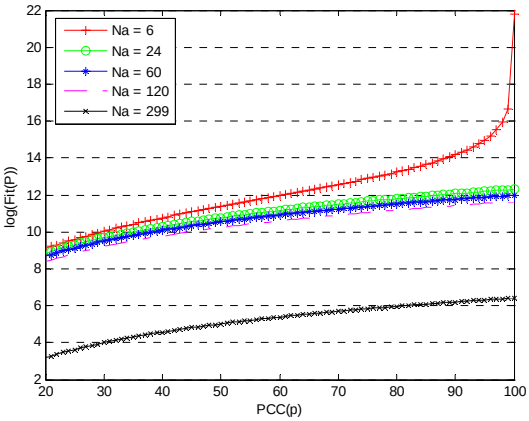

Figure 3-3d: CATmin = 6

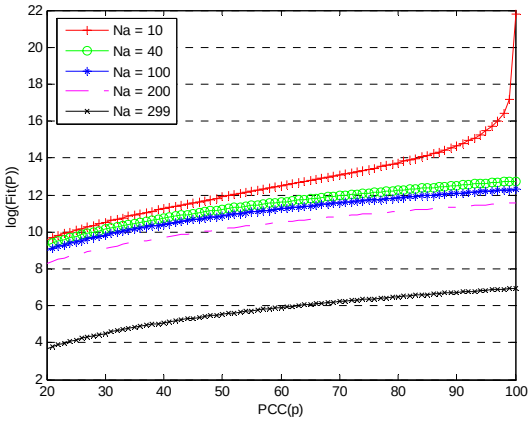

Figure 3-3e: CATmin = 10

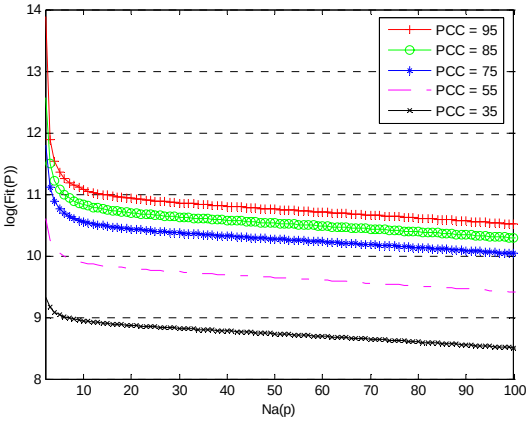

Figure 3-3f: CATmin = 2

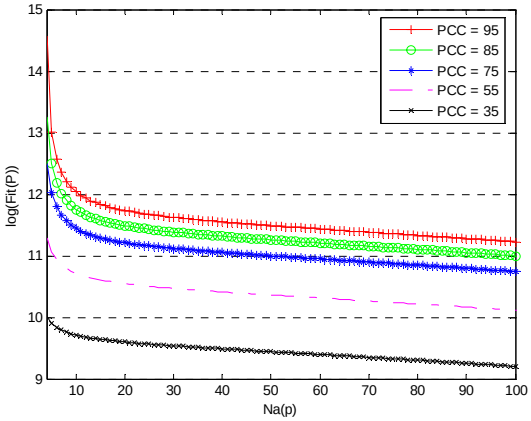

Figure 3-3g: CATmin = 4

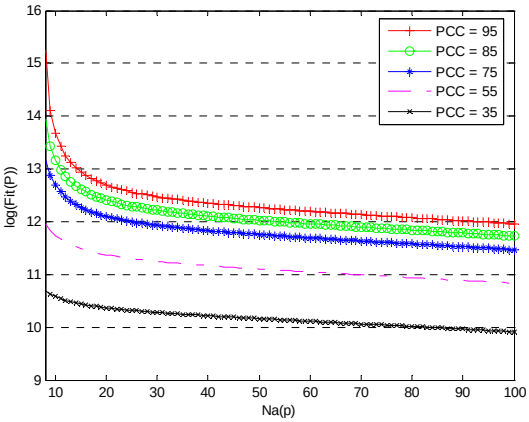

Figure 3-3h: CATmin = 8

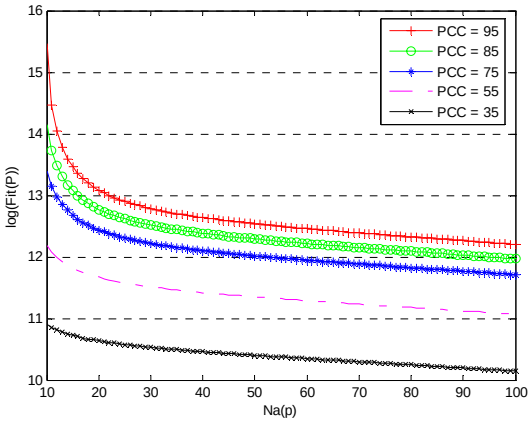

Figure 3-3i: CATmin = 10

3.1.2 Justification of the Genetic Operators Choices for GFAM

In the previous section, we have introduced a number of typical genetic operators, such as mutation and crossover. We have also introduced two genetic operators, Cat_{add} and Cat_{del}, that are more pertinent to the type of problem on which we focusing (co-optimize the

number of categories and generalization performance of the GFAM network that the evolution of the trained FAM architectures produces). These operators were explained in the previous section. In this section, we are focusing on the justification of why these special genetic operators (Cat_{add} and Cat_{del}) are needed for the evolution of FAM architectures. We also provide good default values for the $\Pr(Cat_{add})$, $\Pr(Cat_{del})$, $\Pr(mut)$ probabilities.

Our approach can be summarized as follows. We have chosen three classification problems to work with that are described below (we refer to these problems as *Problems 1, 2, and 3*). For each one of these problems we generated a number of trained FAM architectures and we evolved these architectures for a number of generations. In particular, we generated 20 trained FAMs and evolved them for 500 generations (Experiment 1), we also generated 40 trained FAMs and we evolved them for 250 generations (Experiment 2), and we finally generated 100 trained FAMs and we evolved them for 100 generations (Experiment 3). Hence in all of these experiments the product of trained FAMs and number of generations that these trained FAMs were evolved was a constant (i.e., $Pop_{size} \cdot Gen_{\max} = constant = 10{,}000$). Note that each one of the experiments 1, 2, and 3 was run 50 times with a different random seed each time, and the average fitness value of the best FAM trained network over these 50 runs was reported. For each one of these (problem, experiment) pair, we used three different values for each of the probabilities $P(mut)$, $P(Cat_{add})$ and $P(Cat_{del})$. More specifically, for the mutation probability, we used the following three values: 0, $1/N_a$, $\min(5/N_a, 1)$, where N_a represents the number of categories that the FAM network possesses. The mutation probability of $5/N_a$ seems to be large, especially when N_a is small, but the mutation effect on the categories is fairly small. This is due to the fact that 95% of the random numbers, indicating how much a category should be mutated, lie within an interval whose endpoints are 0.01 away from zero. For $P(Cat_{add})$, we used the following

three values: 0, 0.1 and 0.3. For the $P(Cat_{del})$, we used the following three values: 0, 0.1, and 0.3, It is obvious that if $P(Cat_{del}) = 0$ then the Cat_{del} operator is not used in the evolution of FAMs. Similarly, if $P(Cat_{add}) = 0$ then the Cat_{add} operator is not used in the evolution of FAMs. The values used for the mutation probability, category add probability and category delete probability are depicted in Table 3-1. In Table 3-2, we are depicting (in a tabular form) the number of simulation runs, the number of generations per simulation run, the number of FAMs in the initial population of each of the simulation runs, and the number of combinations of probabilities for mutation, delete category and add category that were tested for each one of the simulation runs. Note that for each set of $P(mut)$, $P(Cat_{add})$ and $P(Cat_{del})$ values, we performed 50 simulation runs for (a) $Gen_{\max} = 500$, $Pop_{size} = 20$, (b) $Gen_{\max} = 250$, $Pop_{size} = 40$, and (c) $Gen_{\max} = 100$, $Pop_{size} = 100$. Hence, we evolved the trained FAM networks a total of 1, 350 (=50 x 27) times for each one of the experiments mentioned above, and 4,050 for each one of the problems mentioned below.

Table 3-1: The values of the probabilities for mutation, category add, and category delete used in the experiments to determine good values for the GA parameters

Value	$P(mut)$	$P(Cat_{add})$	$P(Cat_{del})$
Not Selected	0	0	0
Low Level	1/Na	0.1	0.1
High Level	5/Na	0.3	0.3
Note: these best values are based on previous experiments			

Table 3-2: For each problem (database) we ran 3 experiments. For each experiment we used the depicted combinations of number of generations, and population size (3 combinations). We evolved the trained Fuzzy ARTMAPs 50 different times (50 random seeds), and for each time we used the combinations of probability values, shown in Table 3-1. Hence, the FAMs were evolved 4050 times for each problem, or a total of 12150 times for all the problems.

	$Gen_{\max}$	Pop_{size}	# Random Seeds	($P(Cat_{del})$, $P(mut)$, $P(Cat_{add})$) combinations	# of Runs
Experiment1	500	20	50	27	27 * 50
Experiment2	250	40	50	27	27 * 50
Experiment3	100	100	50	27	27 * 50

The three problems that we chose to experiment with are: ***Problem 1 (Four squares in a square problem)***: In this problem we have four squares (smaller squares) symmetrically located within a square (larger square), as Figure 3-4a demonstrates. The probability that a data-point falls inside each one of the smaller squares or inside the larger square but outside the smaller squares is equal to 0.2. Once a point is chosen to lie within a region, its location within the region is chosen according to a uniform distribution. ***Problem 2 (Varying size squares within a square problem)***: In this problem we have seven squares, all enclosed within the square ([0, 1]x[0,1]), as Figure 3-4b demonstrates. Some of the smaller squares are completely or partially overlapping with the bigger squares. The probability that a point falls in any of the smaller squares is 1/7. The probability that a point falls in any of the bigger squares but not in the region defined by the overlapping smaller squares is also 1/7. Once a point is chosen to lie within a region, its location within the region is chosen according to a uniform distribution. ***Problem 3 (Two circles in a square problem)***: In this problem we have two circles (of different size) within a square ([0, 1]x[0, 1]), as Figure 3-4c demonstrates. The probability that a data-point falls inside the small circle, inside the large circle, or inside the square but outside the circles is equal to 0.2, 0.3, and 0.5, respectively. Once a point is chosen to lie within a region, its location within the region is chosen according to a uniform distribution.

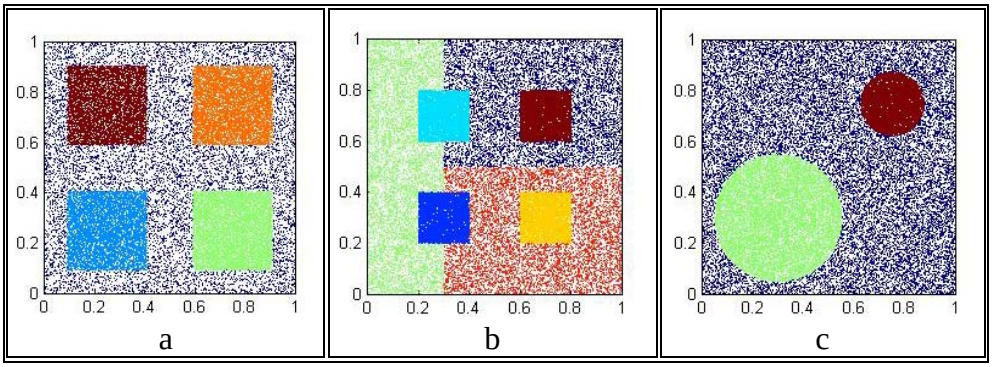

Figure 3-4: a: Problem 1 (Four squares in a square problem), b: (Asymmetric squares within a square problem), c: Problem 3 (Two circles in a square problem)

These problems without necessarily being representative of the type of classification problems that we encounter in real-world applications have the advantage that they deal with 2-D data (for which visualization of the results is easy); they are multi-class classification problems where the number of classes ranges from 3 (Problem 3) to 5 (Problem 1) to 7 (Problem 2); they correspond to problems that have symmetry (Problem 1), or not (Problems 2, 3); they are problems for which the class boundaries have different structures (Problems 1 and 2 has classes with rectangular boundaries, while Problem 3 has classes with circular boundaries); they are problems where one class's boundary is completely or partially included within another class's boundary (thus making the problems non-trivial); and finally they correspond to problems whose optimal decision boundaries and classification accuracy is known.

In Figure 3-5, we show the average fitness value (average over the 50 runs) of the best FAM network produced by GFAM for Problem 1. The vertical axis in Figure 3-5 is showing the average fitness value, while the horizontal axis has discrete ticks that correspond to all possible combinations of the $P(Cat_{add})$ and $P(Cat_{del})$ probabilities that we chose to experiment with (e.g., one of the ticks is 0.0, 0.1 indicating $P(Cat_{del})$ and $P(Cat_{add})$ probabilities equal to 0.0 and 0.1, respectively). Furthermore, there are three curves depicted in Figure 3-5 (with different colors (markers)) that correspond to the three different values of the mutation probability ($P(mut)$. In a similar fashion, in Figure 3-6, we depict the average fitness value (averaged over 50 runs) of the best FAM network produced by GFAM for Problem 2. The philosophy of presenting the results in Figure 3-6 is the same as the philosophy adopted for Figure 3-5. Finally, in Figure 3-7 we are showing the average fitness value (average over the 50 runs) of the best FAM network produced by GFAM for Problem 3. The philosophy of presenting the results in Figure 3-7 is the same as the philosophy adopted for Figure 3-5 and 3-6.

Two obvious observations that can be extracted from the three figures (Figures 3-5, 3-6 and 3-7) are: (a) a zero value of $P(mut)$ is a non-optimal choice, (b) zero values for both $P(Cat_{add})$ and $P(Cat_{del})$, is also a non-optimal choice. If we exclude the value of zero mutation probability and compute the average of the fitness values for the remaining two competing mutation probabilities (mutation probabilities of $1/N_a$ and $5/N_a$) we end up with the curves depicted in Figure 3-8. From Figure 3-8, it is obvious that $P(mut)$ of $5/N_a$ gave better results for the following combinations of $P(Cat_{add})$ and $P(Cat_{del})$: (0, 0) (0, 0.1) (0, 0.3) (0.1, 0) (0.1, 0.1). On the other hand it is also obvious that $(P(mut))$ of $1/N_a$ gave better results for the following combinations of $P(Cat_{add})$ and $P(Cat_{del})$: (0.1, 0.3) (0.3, 0) (0.3, 0.1) (0.3, 0.3). However, both of these values of mutation probabilities produced good results when $P(Cat_{add})$ and $P(Cat_{del})$ were equal to 0.1 and 0.1, respectively. Hence, for our future experiments with GFAM we chose to experiment only with values of $P(mut)$, $P(Cat_{add})$ and $P(Cat_{del})$ equal to $5/N_a$, 0.1 and 0.1, respectively.

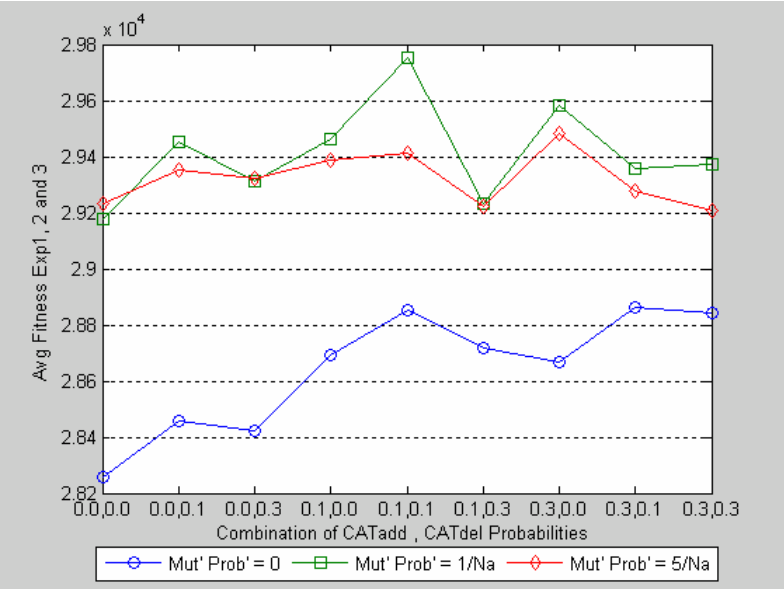

Figure 3-5: Average Fitness value of the Best FAM produced by GFAM for Problem 1. The average is computed over the 50 runs. The average fitness values are shown with respect to all pairs of category add and category delete probabilities. The different colored curves correspond to the three different values of the mutation probability.

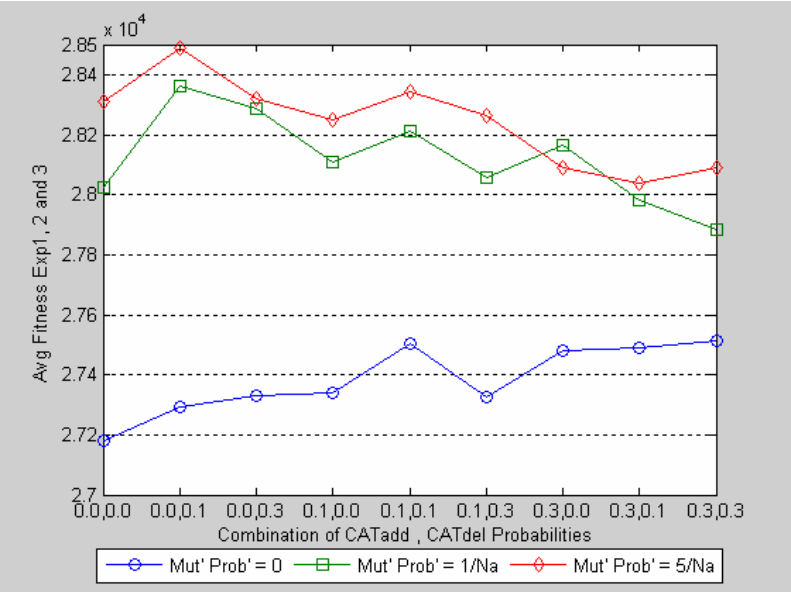

Figure 3-6: Average Fitness value of the Best FAM produced by GFAM for Problem 2. The average is computed over the 50 runs. The average fitness values are shown with respect to all pairs of category add and category delete probabilities. The different colored curves correspond to the three different values of the mutation probability.

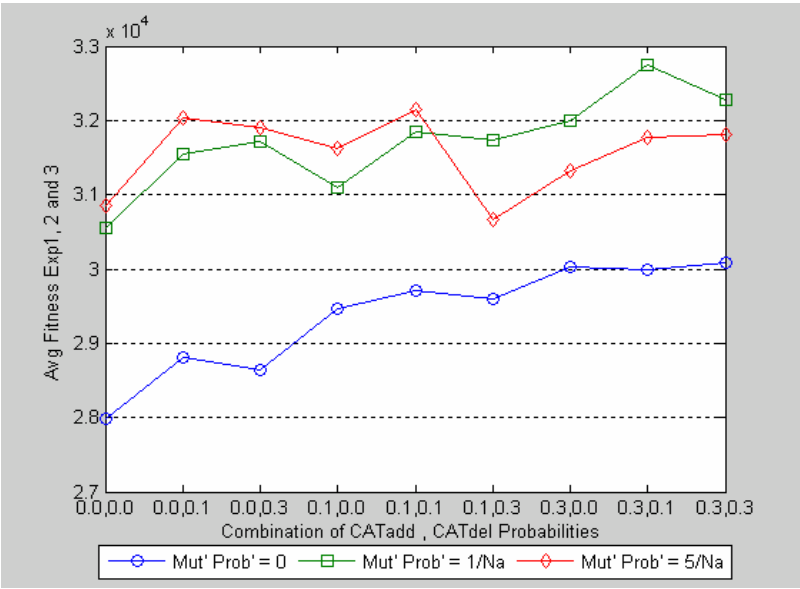

Figure 3-7: Average Fitness value of the Best FAM produced by GFAM for Problem 2. The average is computed over the 50 runs. The average fitness values are shown for all pairs of category add and category delete probabilities. The different colored curves correspond to the three different values of the mutation probability

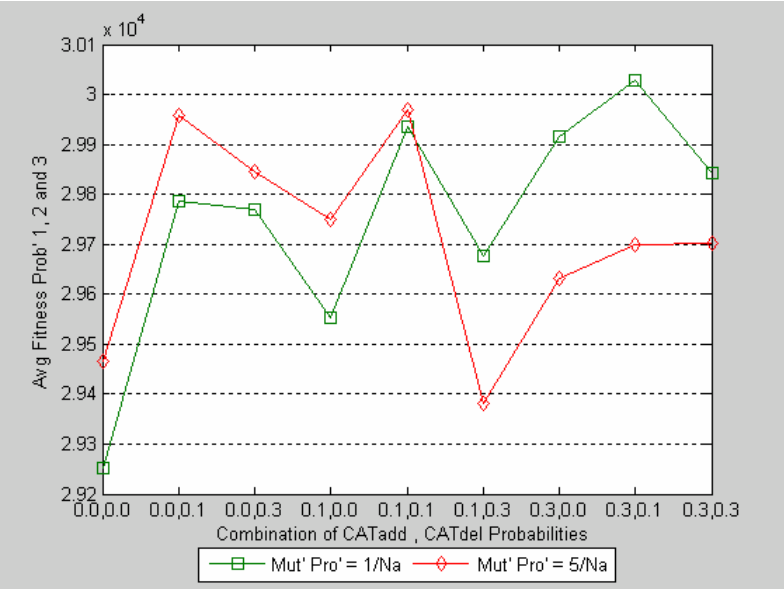

Figure 3-8: Average Fitness value of the Best FAM produced by GFAM for Problems 1, 2 and 3. The average is computed over the 50 runs. The average fitness values are shown for all pairs of category add and category delete probabilities. The different colored curves correspond to the two different non-zero values of the mutation probability

3.2 Experiments with GFAM

We have performed a number of experiments with GFAM. The purpose of these experiments was two-fold: First to examine the performance of GFAM on a variety of classification problems (some of them simulated, some of them real) with respect to the resulting network accuracy (generalization performance) and with respect to resulting network size. Secondly, to compare the performance of GFAM with other ART network classifiers that have been proposed in the literature with the intent of addressing the category proliferation problem in FAM. We undertake the first task (performance of GFAM) in this section and the second task (comparisons of GFAM and other ART classifiers) in the following section.

3.2.1 Databases

We experimented with both artificial and real databases. The specifics of these databases are given in Table 3-3.

1. Gaussian Databases: These are artificial databases, where we created 2-dimensional data sets, Gaussianly distributed, belonging to 2-class, 4-class, and 6-class problems. In each one of these databases we varied the amount of overlap of data belonging to different classes. In particular, we considered 5%, 15%, 25%, and 40% overlap. Note that 5% overlap means the optimal Bayesian Classifier would have 5% misclassification rate on the Gaussianly distributed data. There are a total of 3×4=12 Gaussian databases. We name the databases as "G#c-##" where the first number is the number of classes and the second number is the class overlap. For example, G2c-05 means the Gaussian database is a 2-class and 5% overlap database.
2. Structures within a Structure databases: These are artificial databases that were inspired by the circle (structure) – in the – square (structure) problem. This problem has been extensively examined in the ART, and other than ART neural network literature. Eight different datasets were generated by changing the structures (type, number and probability) that we were dealing with. The data-points within each structure of these artificial datasets are either uniformly distributed within the structure. The number of points within each structure is chosen in a way that the probability of finding a point within this structure is equal to a pre-specified number.
   - 4Ci/Sq: This is a 4 circle in a square problem, obviously a five class classification problem. The probability of finding a data point within a circle or inside the square and outside the circles is equal to 1/5.
   - 4Sq/Sq: This is a 4 square (inside squares) in a square (outside square) problem, obviously a five class classification problem. The probability of finding a data point within an inside square or outside the inside squares and inside the outside square is equal to 1/5.

- 7Sq: This is a seven class problem, with four squares and 3 rectangle-like shapes. The probability of finding a data point within any of the seven squares is equal to 1/7.
- 1Ci/Sq: This is a 1 circle in a square problem, obviously a two class classification problem. The probability of finding a data point within a circle or inside the square and outside the circle is equal to 1/2. The sizes of the areas in the circle and outside the circle and inside the square are the same. This is the benchmark circle in the square problem.
- 1Ci/Sq/0.3:0.7: This is a 1 circle in a square problem, obviously a two class classification problem. The probability of finding a data point within a circle or inside the square and outside the circle is equal to 0.3 and 0.7, respectively. The sizes of the areas in the circle and outside the circle and inside the square are 0.3 and 0.7 respectively.
- 5Ci/Sq: This is 5 concentric circles in a square problem, obviously a six class classification problem. The probability of finding a data point within each one of the co-centric circles, or inside the square and outside the circles is equal to 1/6.
- 2Ci/Sq/5:25:70: This is two circles in a square problem, obviously a three class classification problem. One of the circles is smaller than the other. The probability of finding a data point within the small circle, the large circle, and outside the circles and inside the square is 0.05, 0.25, and 0.7, respectively.
- 2Ci/Sq/20:30:50: This is two circles in a square problem, obviously a three class classification problem. One of the circles is smaller than the other. The probability of finding a data point within the small circle, the large circle, and outside the circles and inside the square is 0.2, 0.3, and 0.5, respectively.

In Figure 3-9 and 3-10 we show plots of the simulated databases.

3. Modified Iris Database (MOD-IRIS): In this database we started from the IRIS dataset (Hettich et al. [16]) of the 150 3-class problem. We eliminated the data corresponding to the class that is linearly separable from the others. Thus, we ended up with 100 data-points. From the four input attributes of this IRIS dataset we focused on only two attributes (attribute 3 and 4) because they seem to have enough discriminatory power to separate the 2-class data. Finally, in order to create a reasonable size dataset from these 100 points (so we can reliably perform cross-validation to identify the optimal ART, GFAM networks) we created noisy data around each one of these 100 data-points (the noise was Gaussian of zero mean and small variance) to end up with approximately 10,000 points. We named this database Modified Iris.
4. Modified Abalone Database (ABALONE): This database is originally used for prediction of the age of an abalone (Hettich et al. [16]). It contains 4177 instances, each with 7 numerical attributes, 1 categorical attribute, and 1 numerical target output (age). We discarded the categorical attribute in our experiments, and grouped the target output values into 3 classes: 8 and lower (class 1), 9-10 (class 2), 11 and greater (class 3). This grouping of output values has been reported in the literature before.
5. Page Blocks Database (PAGE): This database represents the problem of classifying the blocks of the page layout in a document (Hettich et al. [16]). It contains 5473 examples coming from 54 distinct documents. Each example has 10 numerical attributes (e.g., height of the block, length of the block, eccentricity of the block, etc.,) and one target (output) attribute, representing the type of the block (text, horizontal line, graphic, vertical line, and picture). One of the noteworthy points about this database is that its major class (text) has a high probability of occurring (above 80%). This dataset has five classes, four of them make only 9% of the total instances.

The data in each one of the above databases was split into a training set, a validation set, and a test set. The percentage of classes in each one of these subsets resembled the percentage of classes in the original dataset. The summarized specifics of each one of these databases are depicted in Table 3-3. The training set was used to train the FAM networks that were used to initialize the population in GFAM, the validation test was used to assess the performance of the FAM networks during their evolution, and the test set was used to report the performance of the best FAM network (called GFAM) at the completion of the evolutionary process.

Table 3-3: Databases used in the Genetic ARTMAP experiments

	Database Name	# Training Instances	# Validation Instances	# Test Instances	# Numerical Attributes	# Classes (N_b)	% Major Class (A_0)
1	G2c-05	500	5000	5000	2	2	1/2
2	G2c-15	500	5000	5000	2	2	1/2
3	G2c-25	500	5000	5000	2	2	1/2
4	G2c-40	500	5000	5000	2	2	1/2
5	G4c-05	500	5000	5000	2	4	1/4
6	G4c-15	500	5000	5000	2	4	1/4
7	G4c-25	500	5000	5000	2	4	1/4
8	G4c-40	500	5000	5000	2	4	1/4
9	G6c-05	504	5004	5004	2	6	1/6
10	G6c-15	504	5004	5004	2	6	1/6
11	G6c-25	504	5004	5004	2	6	1/6
12	G6c-40	504	5004	5004	2	6	1/6
13	4Ci/Sq	2000	5000	3000	2	5	0.2
14	4Sq/Sq	2000	5000	3000	2	5	0.2
15	7Sq	2000	5000	3000	2	7	1/7
16	1Ci/Sq	2000	5000	3000	2	2	0.5
17	1Ci/Sq/0.3:0.7	2000	5000	3000	2	2	0.7
18	5Ci/Sq	2000	5000	3000	2	6	1/6
19	2Ci/Sq/5:25:70	2000	5000	3000	2	3	0.7
20	2Ci/Sq/20:30:50	2000	5000	3000	2	3	0.5
20	7SqWN	504	5004	5004	2	6	1/7
21	5Ci/SqWN	504	5004	5004	2	6	1//6
22	MOD-IRIS	500	4800	4800	2	2	1/2
23	ABALONE	501	1838	1838	7	3	1/3
24	PAGE	500	2486	2487	10	5	0.832

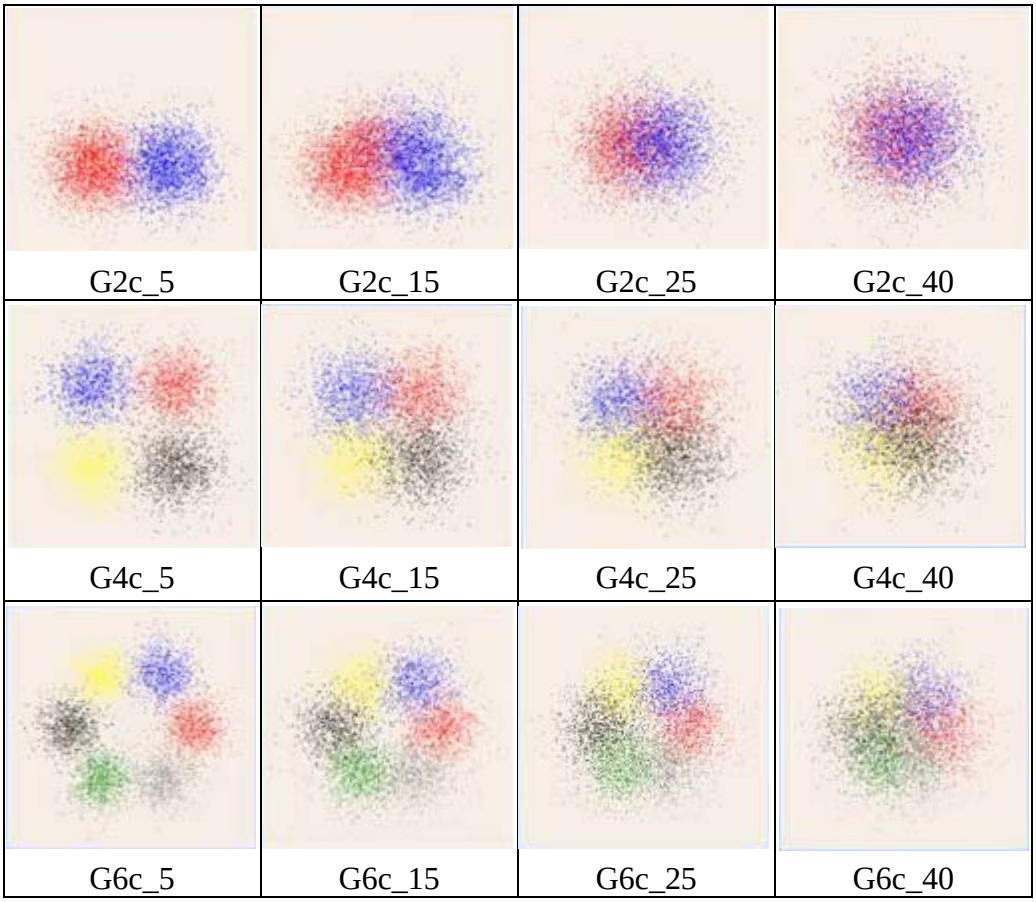

Figure 3-9: Gaussian Databases (2-dimensional, 2, 4 or 6 class, 5, 15, 25 and 40 % of overlap)

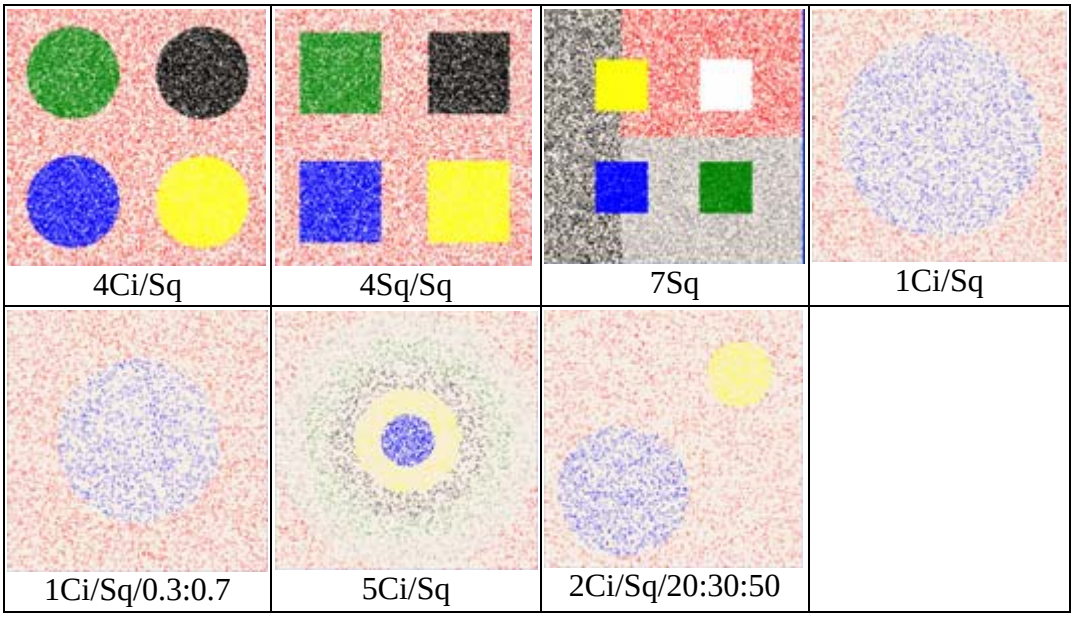

Figure 3-10: Structures within Structure Databases

3.2.2 Experimental Procedure – Experimental Results

In section 3.1.1.2, we have experimented extensively with GFAM to identify a good initialization of the GA process and to specify a good set of parameters for the evolution of trained FAMs. From this point on, the GFAM is produced by first initializing a population of 20 trained FAM networks (they were trained with different values of the baseline vigilance parameter and different orders of training pattern presentations), and by evolving them for 500 generations. In particular, the GA parameters used for the creation of GFAM were: $\rho_a^{\min} = 0.1,\ \rho_a^{\max} = 0.95,\ \beta_a = 0.1,\ Pop_{size} = 20,\ Gen_{\max} = 500,\ NC_{best} = 3,\ Cat_{\min} = 1,\ Cat_{\max} = 300,\ P(Cat_{add}) = 0.1, P(Cat_{del}) = 0.1, P(mut) = 5/Na$. GFAM is the FAM network that attains the highest value of the fitness function at generation 500 of the evolutionary process.

3.3 GFAM Performance

In this section we are reporting the performance of GFAM on each one of the database problems, described in Section 3.2.1. The performance of GFAM is assessed by reporting the size of GFAM and the accuracy attained by GFAM on the test set. The results are reported in Table 3-4. In Table 3-4, the first column is the index of the database that we are experimenting with. The second column is the actual database name, as reported in section 3.2.1. Columns 3 and 4 contain the accuracy and size of the GFAM network for the designated database. The performance of GFAM, as it is evidenced by the results in Table 3-4, is verified by some obvious observations. For instance, GFAM's performance on databases 1-12 (Gaussian datasets of known amount of overlap) is nearly optimal; for example the best performance on the G6c-15 problem (6 class Gaussian dataset of 15% overlap) is a classifier with 6 categories and 85% correct classification, and GFAM is a classifier with 6 categories and 84.71% of correct classification. Similarly, in the 7Square problem the optimal classifier would require 7 categories and attain a 100% correct classification; GFAM is a 7 category classifier exhibiting a 97.2% of correct classification. Finally, two of the real problems

reported here, MOD-IRIS and PAGE, also gave very good results almost 95% percentage of correct classification, by creating only two categories. Notice how GFAM ignored one class in the abalone dataset (abalone is a 3 class problem), since the number of samples from this class is very small compared to the number of samples pertaining to the other two classes. Hence, GFAM's fitness function was higher for a network with 2 categories and 58.73% generalization accuracy than for a network with 3 categories and a slightly higher generalization accuracy.

3.3.1 Performance Comparisons of GFAM and other ART Networks

We compared GFAM's performance with the performance of the following networks: ssFAM, ssEAM, ssGAM, and safe micro-ARTMAP. We chose these networks for a reason. Each one of these ART networks at the time of their introduction into the literature emphasized that they were addressing the category proliferation problem in ART. More details about the specifics of each one of these networks can be found in their associated references (provide references here). For the purposes of this thesis it suffices to know that ssEAM covers the space of the input patterns with ellipsoids, while ssGAM covers the space of the input patterns with bell-shaped curves. Furthermore ssFAM, ssEAM, and ssGAM allow a category (hyper-rectangle or ellipsoid or hyper-dimensional bell shaped curve) to encode patterns of different labels provided that the plurality label of a category exceeds a certain, user-specified, threshold. Finally, micro-ARTMAP allows the encoding of patterns of different labels by a single category, provided that the entropy of the category does not exceed a certain, user-defined threshold.

The comparisons of GFAM and the aforementioned ART networks are depicted in Table 3-4 where the first column is the index of the database that we are experimenting with. The second column is the actual database name, as reported in earlier sections. Columns 3-10 of Table 3-4 contain the performance of the designated ART networks. The performance

reported includes the accuracy of the "best" ART network on the test set. The performance also includes the number of categories created of the designated ART network. The reported numbers of accuracy and size of the "best" network correspond to the ART network that attained the highest value of the fitness function (this value was computed based on the accuracy of the ART network on the cross-validation set, and on the size of the ART network). Note, that for networks, other than GFAM, the "best" ART network was determined after extensive experimentation with the ART network's parameter values (e.g., in ssFAM the best network was determined after training ssFAM networks with different values of the choice parameter, vigilance parameter, order of pattern presentation, and amount of mixture of labels allowed within a category; a total of 20,000 ssFAM networks were trained and their performance was examined). On the other hand, the performance of the GFAM is the one calculated after the evolution of 20 FAM trained networks for 500 generations with GA values as indicated in Section 5.3.

According to the results in Table 3-4, in all instances (except minor exceptions) the accuracy of GFAM (generalization performance) is higher than the accuracy of the other ART network (where ART is ssFAM, ssEAM, ssGAM or safe micro-ARTMAP). According to the results in Table 3-4, in all instances (with no exceptions) the size of GFAM is smaller than the size of the other ART network (where ART is ssFAM, ssEAM, ssGAM or safe micro-ARTMAP), sometimes even by a factor of 15. For example, the generalization performance of GFAM can be as 13% better than the generalization performance of ssFAM, while its size can be by a factor of 4 times smaller than the size of ssFAM. Also, the generalization performance of GFAM can be as 13% better than the generalization performance of ssEAM, while its size can be by a factor of 4.5 times smaller than the size of ssEAM. Furthermore, the generalization performance of GFAM can be as 6% better than the generalization performance of ssGAM, while its size can be by a factor of 15 times smaller

than the size of ssGAM. Finally, the generalization performance of GFAM can be as 10% better than the generalization performance of safe micro-ARTMAP, while its size can be by a factor of 3 times smaller than the size of safe micro-ARTMAP.

The comparison results between GFAM and the other ART networks are also pictorially depicted in figures 3-11a to 3-11d. In each one of these figures we are showing the accuracy of GFAM, and that of one other network (e.g., ssFAM). In the same figure we are also showing the size of the GFAM and that of one other ART network. This way the one-to-one comparison of the GFAM and the other ART network can be quickly assessed.

What is most worth pointing out is that the better performance of GFAM is attained with reduced computations compared with the computations needed by the alternate methods (ssFAM, ssEAM, ssGAM, safe micro-ARTMAP). Specifically, the performance attained by ssFAM, ssEAM, ssGAM and the safe micro-ARTMAP required training these networks for a large number of network parameter settings (at least 20,000 experiments) and then choosing the network that achieved the higher value for the fitness function that we introduced earlier in Section 4a. Of course, one can argue that such an extensive experimentation with these networks might not be needed, especially if one is familiar with the functionality of these networks and chooses to experiment only with a limited set of network parameter values. However, the practitioner in the field might lack the expertise to carefully choose the network parameters to experiment with, and consequently might need to experiment extensively to come up with a good network. In Appendix B, we show, in more detail, how more computationally efficient GFAM is compared to ssFAM, ssEAM, ssGAM and safe micro-ARTMAP. The comparison is based under the assumption that extensive parameter experimentation with the network parameters of ssFAM, ssEAM, ssGAM or safe micro-ARTMAP is needed to obtain a good performing ssFAM, ssEAM, ssGAM or safe micro-ARTMAP network, respectively.

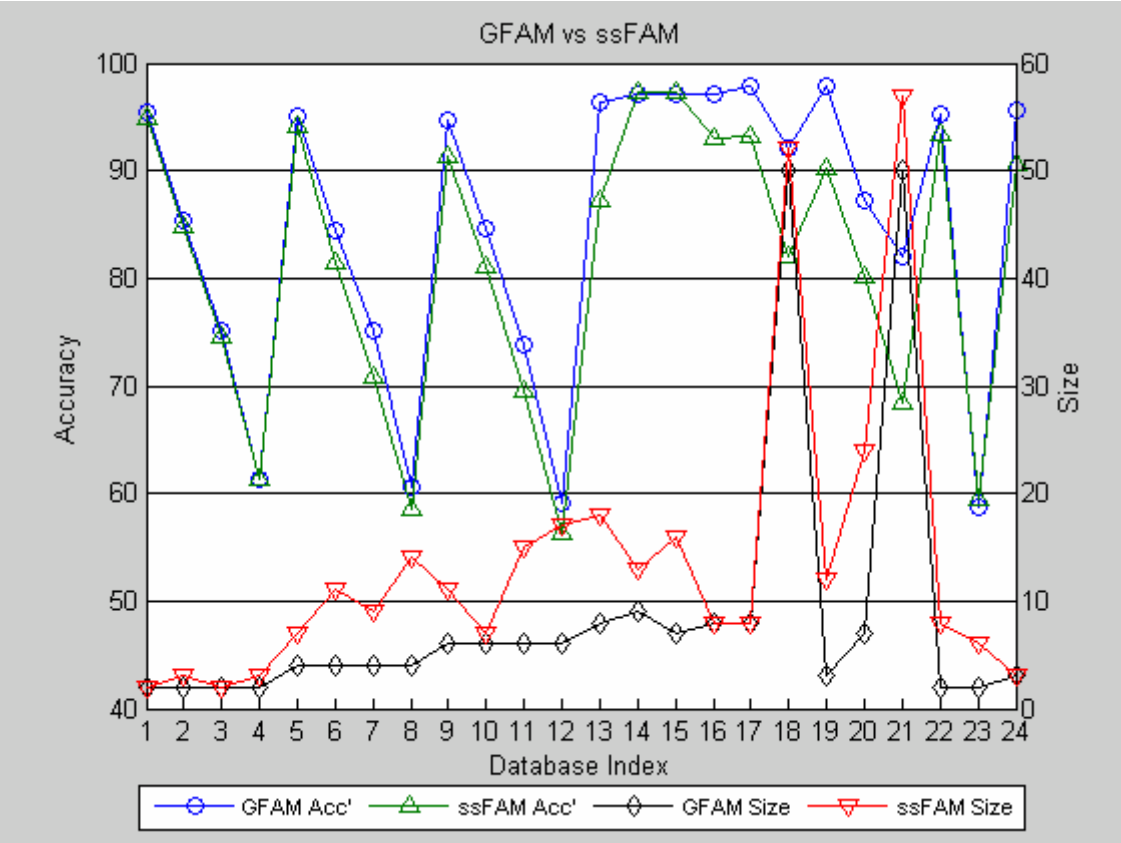

Figure 3-11a: Performance and Size comparison of GFAM vs ssFAM

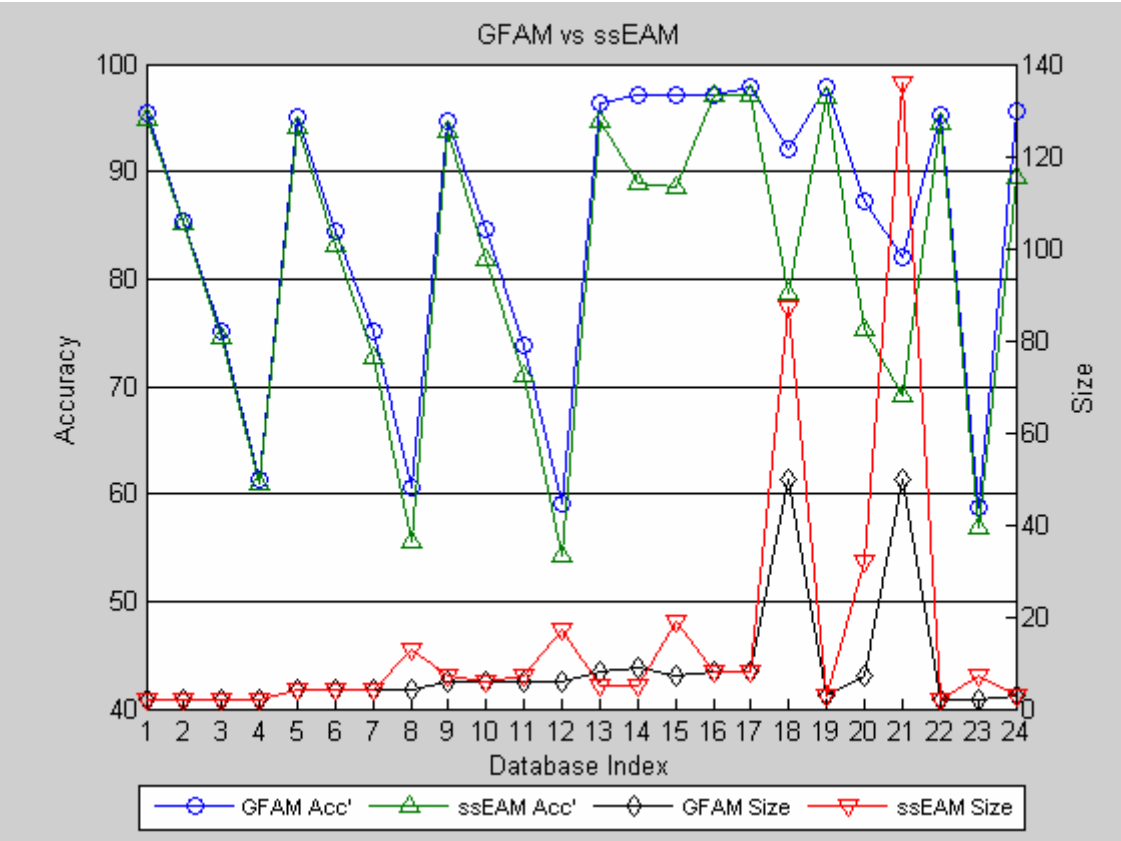

Figure 3-11b: Performance and Size comparison of GFAM vs ssEAM

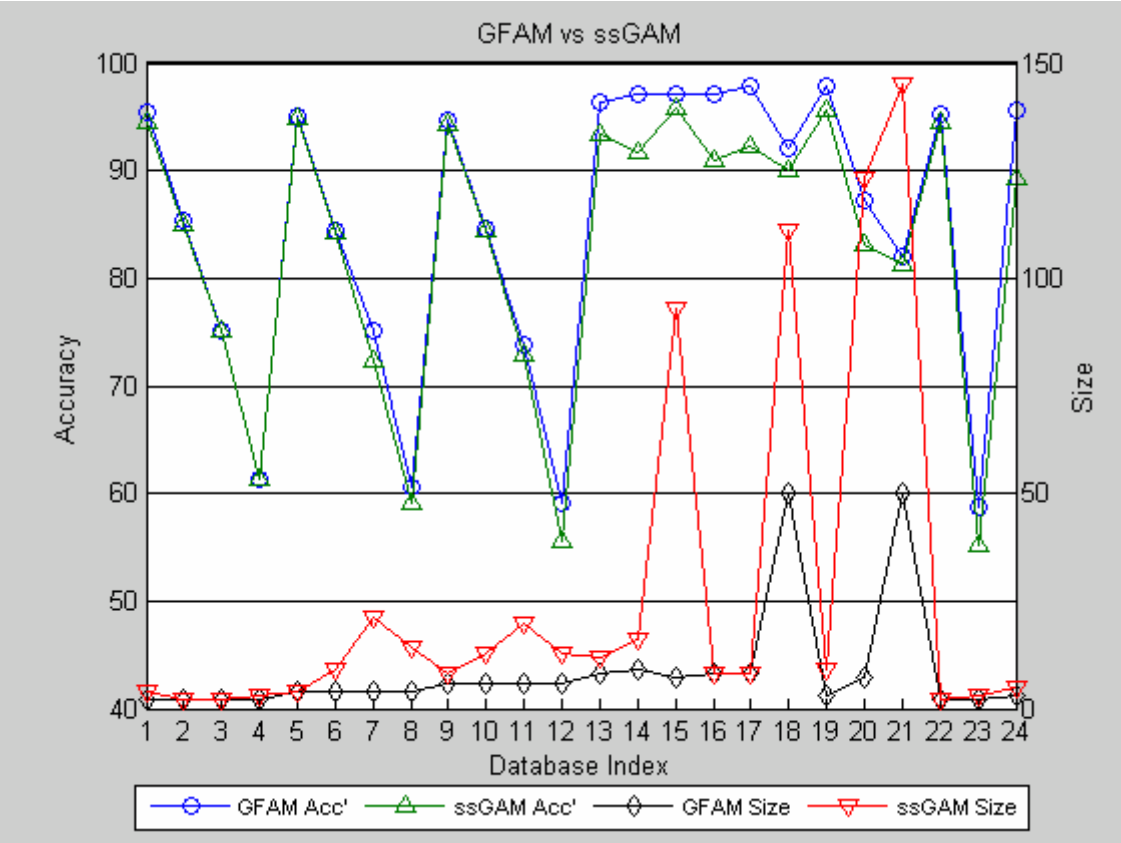

Figure 3-11b: Performance and Size comparison of GFAM vs ssGAM

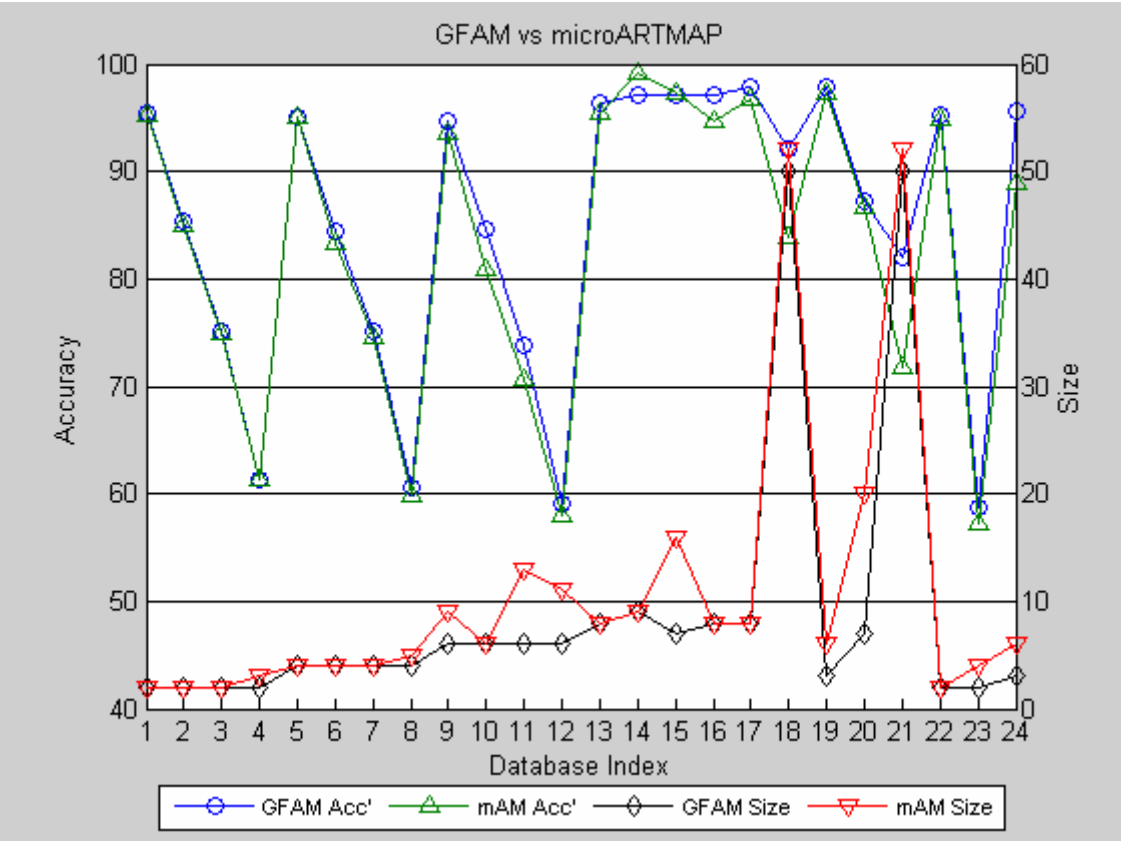

Figure 3-11d: Performance and Size comparison of GFAM vs microARTMAP

Table 3-4: Accuracy and size results achieved by GFAM and other ART networks. Note that:Safe uAM: Safe microARTMAP; FAM: Fuzzy ARTMAP; EAM: Ellipsoidal ARTMAP; GAM: Gaussian ARTMAP; ss*: semi-supervised version

	Database Name	GFAM		Safe μAM		ssFAM		ssEAM		ssGAM	
1	G2c-05	95.36	2	95.22	2	94.90	2	94.94	2	94.48	4
2	G2c-15	85.30	2	85.00	2	84.80	3	85.20	2	85.04	2
3	G2c-25	75.08	2	74.98	2	74.60	2	74.50	2	75.10	2
4	G2c-40	61.38	2	61.40	3	61.34	3	60.98	2	61.30	3
5	G4c-05	95.02	4	95.04	4	94.10	7	94.14	4	94.80	4
6	G4c-15	84.46	4	83.28	4	81.40	11	83.20	4	84.24	9
7	G4c-25	75.20	4	74.50	4	70.80	9	72.72	4	72.32	21
8	G4c-40	60.60	4	59.76	5	58.48	14	55.62	13	59.10	14
9	G6c-05	94.68	6	93.57	9	91.42	11	93.80	7	94.40	8
10	G6c-15	84.71	6	80.92	6	81.11	7	81.80	6	84.35	13
11	G6c-25	73.90	6	70.74	13	69.62	15	71.10	7	72.86	20
12	G6c-40	59.19	6	58.03	11	56.35	17	54.21	17	55.65	13
13	4Ci/Sq	96.32	8	95.42	8	87.23	18	94.68	5	93.4	12
14	4Sq/Sq	97.12	9	99.12	9	97.24	13	88.89	5	91.78	16
15	7Sq	97.2	7	97.22	16	97.26	16	88.5	19	95.83	93
16	1Ci/Sq	97.2	8	94.76	8	92.97	8	97.02	8	91.02	8
17	1Ci/Sq/0.3:0.7	97.8	8	96.82	8	93.21	8	97.13	8	92.33	8
18	5Ci/Sq	92	50	83.83	52	81.95	52	78.68	87	90.02	111
19	2Ci/Sq/20:30:50	97.87	3	97.22	6	90.24	12	97.01	3	95.6	9
20	7SqWN	87.3	7	86.67	20	80.15	24	75.23	32	83.11	123
21	5Ci/SqWN	81.97	50	71.72	52	68.39	57	69.2	136	81.3	145
22	MOD-IRIS	95.31	2	94.92	2	93.41	8	94.54	2	94.54	2
23	ABALONE	58.73	2	57.18	4	59.52	6	56.80	7	55.10	3
24	PAGE	95.59	3	88.82	6	90.63	3	89.54	3	89.34	5

3.3.2 Performance Comparisons of GFAM and Other Neural Networks

The comparison of GFAM, and ssFAM, ssEAM, ssGAM, provided in the previous section is fair because it used the same databases and datasets/per database for training,

validation and testing of these architectures, and the same criterion for finding the best of these ART architectures (the criterion was to maximize the fitness function, defined in Section 3.1.1). However, some of the structures/in/a structure artificial databases, extensively examined above, have also been utilized to assess the performance of other ART architectures, such as the distributed Fuzzy ARTMAP (dFAM), FasART, and distributed FasART (see Parado-Hernandez, et al., 2003). Distributed Fuzzy ARTMAP differs by Fuzzy ARTMAP in the sense that more than one category is activated to represent an input pattern in ART's training phase. FasART uses a different activation function compared to the one used by Fuzzy ARTMAP. Finally, distributed FasART is the distributed version of FasART, in a similar manner as distributed Fuzzy ARTMAP is the distributed version of Fuzzy ARTMAP. More details about the functionality of these ART networks can be found in Parado-Hernandez, et al., 2003 and they are beyond the scope of this paper. We avoided the extensive comparison of GFAM with dFAM, FasART, and dFasART for a reason. Although some of the databases used to assess the performance of dFAM, FasART, and dFasART, in Parado-Hernandez, et al., 2003, are the same as the databases used to assess the performance of GFAM, the actual data used for training, and testing of GFAM are not the same used for the training and testing of dFAM, FasART, and dFasART. Furthermore, parameter network optimization with a validation set, such as to optimize a fitness function, was not conducted for FasART, dFAM, and dFasART. Actually, the results reported in Parado-Hernandez, et al., are averages of the performances of the dFAM, FasART, and dFASART on a test set of 5,000 points for a specific set of network parameter values (we tend to think that it was a good set of network parameter values). The averages correspond to the average performance attained by 100 different choices of training sets of size equal 2,000 points. The comparison between GFAM performance and dFAM, FasART, and dFasART performances can be deduced from the summarized numbers of Table 3-5. Based on this table, we can state that

the GFAM performance is better than the averages of the performances attained by dFAM, FasART and dFasART, at least for the databases contained in Table 3-5.

Table 3-5: Accuracy and size results achieved by GFAM and other ART networks. Note: dFAM: Distributed Fuzzy ARTMAP, FasART, dFasART : Distributed FasART, GFAM : Genetic Fuzzy ARTMAP

Database Index	Database Name	dFAM		FasART		dFasART		GFAM	
13	4Ci/Sq	87.19	33.38	92.76	22.30	87.95	22.38	96.32	8
14	4Sq/Sq	95.68	30.72	91.82	139.26	91.29	69.32	97.12	9
16	1Ci/Sq	88.90	12.34	96.30	63.42	92.78	30.50	97.2	8
17	1Ci/Sq/0.3:0.7	75.79	6.96	97.00	56.12	96.28	12.24	97.8	8
18	5Ci/Sq	80.85	125.14	95.95	278.7	73.39	179.74	92	50
19	2Ci/Sq/20:30:50	95.23	12.96	96.27	57.68	94.91	24.32	97.87	3

3.4 GFAM Summary and Conclusions

Adaptive Resonance Theory (ART) neural networks have been introduced into the literature by Carpenter, Grossberg and their colleagues at Boston University, as well as other researchers in the field. The consensus with ART networks is that they converge fast to a solution for arbitrary classification problems they can provide explanations for the answers that they produce, they can function in an on-line training mode, and they solve effectively a variety of classification problems. However, all these benefits sometimes come at the expense of unnecessarily creating too many categories to solve the problem at hand, referred to as the category proliferation problem in ART. This problem is more acute when ART is confronted with classification problems that deal with noisy or highly overlapping data. To alleviate this problem a number of researchers have proposed solutions, such as ssFAM, ssEAM, ssGAM (see Anagnostopoulos, et al., 2003, Verzi, et al., 2001), and safe micro-ARTMAP (see Gomez, et al., 2002), to mention only a few.

In this thesis, we have introduced, yet another, method of solving the category proliferation problem in ART. This method relies on evolving a population of trained ART

networks, and more specifically Fuzzy ARTMAP (FAM) neural networks. The evolution of trained FAMs creates an ART network, referred to as GFAM.

We have experimented with a number of databases that helped us identify good default parameter settings for the evolution of FAM. We defined a fitness function that gave emphasis to the creation of a small size FAM networks which exhibited good generalization. In the evolution of FAM trained networks we also defined and justified the usage of unique operators, such as the delete category and add category operators. The GFAM network identified at the end of the evolutionary process (last generation) was the FAM network that attained the highest fitness value. Our method for creating GFAM resulted in a FAM network that performed well on a number of classification problems.

In particular, GFAM was found superior to a number of other ART techniques (ssFAM, ssEAM, ssGAM, safe micro-ARTMAP) that have been introduced into the literature to address the category proliferation problem in FAM. More specifically, GFAM gave a better generalization performance (in almost all problems tested) and a smaller size network (in all problems tested), compared to these other ART techniques. What is also worth mentioning is that GFAM outperformed these other ART techniques by requiring only a fraction of the computations needed by these other networks.

Obviously, the introduced method to evolve trained FAMs can be extended to other ART architectures, such as EAM, and GAM, amongst others, without any significant changes in the approach followed, and that is the issue that we tackle in Sections 4 and 5 of this thesis.

4. GEAM AND GGAM

4.1 Genetic Ellipsoidal ARTMAP (GEAM)

GFAM (Genetic Ellipsoidal) is an evolved EAM network that is produced by applying, repeatedly, genetic operators on an initial population of trained EAM networks. GEAM uses tournament selection with elitism, as well as genetic operators, including crossover and mutation. In addition, GEAM uses two special operators, Cat_{add} and Cat_{del}.

To better understand how GEAM is designed we resort to a step-by-step description of this design. Please refer to the terminology introduced in Appendix A before you dwell in the ste-by-step description of EAM. The design of GEAM can be articulated through a sequence of steps, defined succinctly below, and explained in detail later.

Step 1: Initialize Pop_{size} number of EAM networks, each one of them operating with a different value for the baseline vigilance parameter $\overline{\rho}_a$, and possibly different orders of pattern presentation.

Step 2: Train each one of the Pop_{size} initialized EAM networks, using the training set for a maximum number of iterations ($Gen_{\max}$).

Step 3: Convert the Pop_{size} trained EAM networks into chromosomes. Crop all chromosomes so that no one-point categories exist.

Step 4: Evolve the chromosomes of the current generation by executing the following sub-steps:

Sub-Step 4a: Calculate fitness for all chromosomes of the current generation.

Sub-Step 4b: Initialize an empty generation (referred to as *temporary generation*).

Sub-Step 4c: Move the best (NC_{best}) chromosomes from the current generation to the temporary generation.

Sub-Step 4d: Select chromosomes for crossover from the current generation and thus further populate the temporary generation.

Sub-Step 4e: With a probability $P(Cat_{add})$ apply the Cat_{add} operator on every individual generated in sub-step 4d.

Sub-Step 4f: With a probability $P(Cat_{del})$ apply the Cat_{del} operator on every individual generated in sub-step 4e.

Sub-Step 4g: With a probability $P(Mut)$ apply the mutation operator on every individual generated in sub-step 4f.

Sub-Step 4h: Replace the current generation with the members of the temporary generation

Step 5: If evolution has reached the maximum number $Gen_{\max}$ of iterations, then calculate the performance of the best-Fitness EAM network on the test set and report classification accuracy and number of categories that this Best-Fitness EAM network possesses. If the maximum number of iterations has not been reached yet, go to step 4 to evolve one more population of chromosomes

Each one of the aforementioned steps of the algorithm is now described in more detail, as needed.

Step 1 (More Details): The algorithm starts by training Pop_{size} EAM networks, each one of them trained with a different value of the baseline vigilance parameter $\overline{\rho}_a$. In particular, we first define $\overline{\rho}_a^{inc} = \dfrac{\overline{\rho}_a^{\max} - \overline{\rho}_a^{\min}}{Pop_{size} - 1}$, and then the baseline vigilance parameter of every network is determined by the equation $\overline{\rho}_a^{\min} + i * \overline{\rho}_a^{inc}$, where $i \in \{0, 1, ..., Pop_{size} - 1\}$. In our experiments with GFAM we chose $\overline{\rho}_a^{\min} = 0.1$, and $\overline{\rho}_a^{\max} = 0.95$. Meanwhile, GEAM allows the user to change the order of pattern presentation automatically and randomly.

Step 2 (More Details): We assume that the reader is familiar of how training of EAM networks is accomplished, and thus the details here are omitted.

Step 3 (More Details): Once the Pop_{size} networks are trained they need to be converted to chromosomes, so that they can be manipulated by the genetic operators. GEAM uses a mix of real numbers representation to encode the networks. Each EAM chromosome consists of two levels, level 1 containing all the categories of the EAM network, and level 2 containing the center, the direction, the radius and the axis ratio, as well as the label of that category (see Figure 4-1). We denote the category of a trained EAM network with index p $(1 \le p \le Pop_{size})$ by $\mathbf{w}_j^a(p)$, where $\mathbf{w}_j^a(p) = (\mathbf{m}_j^a, \mathbf{d}_j^a, r_j^a)$, the axis ratio by $\mu_j^a(p)$, and the label of this category by $l_j(p)$ for $1 \le j \le N_a(p)$. In this step we are also eliminating single-point categories in the trained EAM networks, referred to as cropping the chromosomes. Since our ultimate objective is to design an EAM network that reduces the network size and improves generalization we are discouraging at this stage the creation of single-point categories.

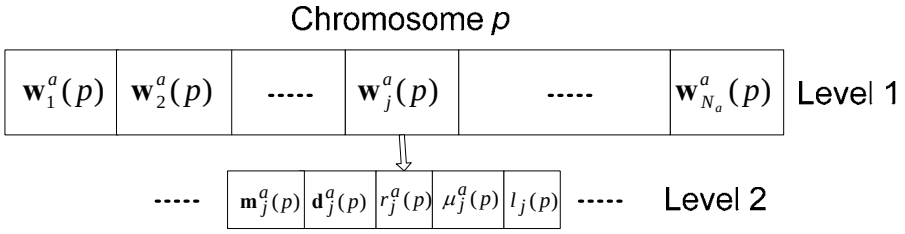

Figure 4-1: GEAM chromosome structure

Step 4 (More Details): In this step the GFAM applies a GA to the population of trained EAMs.

Sub-step 4a (More Details): Calculate the fitness of each chromosome (trained EAM). This is accomplished by feeding into each trained EAM the validation set and by calculating the percentage of correct classification exhibited by each one of these trained EAM networks. In

particular, if $PCC(p)$ designates the percentage of correct classification, exhibited by the *p*-th EAM, and this EAM network possesses $N_a(p)$ nodes in its category representation layer, then its fitness function value is defined by:

$$Fit(p) = \frac{(Cat_{\max} - N_a(p)) \cdot PCC^2(p)}{\frac{100}{Cat_{\min}} - \frac{PCC(p)}{N_a(p)} + \varepsilon} \qquad 4\text{-}1$$

where, $Cat_{\min}$ and $Cat_{\max}$ are the minimum and maximum number of categories that a FAM network is allowed to have during the evolutionary process ($Cat_{\min}$ is chosen equal to 1, or equal to the number of classes in the classification problem under consideration, while $Cat_{\max}$ is chosen to be a relatively large number for the classification problem at hand). The constant ε in the denominator of the above equation is a small positive constant and it is needed to make sure that the denominator would not be zero in the case when $N_a(p) = Cat_{\min}$ and $PCC(p) = 100$.

Sub-step 4b (More Details): Obvious, no further explanations are needed.

Sub-step 4c (More Details): The algorithm searches for the best NC_{best} chromosomes from the current generation and copies them to the temporary generation.

Sub-step 4d (More Details): The remaining $Pop_{size} - NC_{best}$ chromosomes in the temporary generation are created by crossing over pairs of parents from the current generation. The parents are chosen using a deterministic tournament selection, as follows: Randomly select two groups of four chromosomes each from the current generation, and use as a parent, from each group, the chromosome with the best fitness value in the group. If it happens that from both groups the same chromosome is chosen then we choose from one of the groups the chromosome with the second best fitness value. If two parents with indices p, p' are crossed over two random numbers n, n' are generated from the index sets $\{1, 2, ..., N_a(p)\}$ and $\{1, 2, ..., N_a(p')\}$, respectively. Then, all the categories with index

greater than index n' in the chromosome with index p' and all the categories with index less than index n in the chromosome with index p are moved into an empty chromosome within the temporary generation. Notice that crossover is done on level 1 of the chromosome. This operation is pictorially illustrated in Figure 3-2.

Sub-step 4e (More Details): The operator Cat_{add} adds a new category to every chromosome created in step 4d with probability $P(Cat_{add})$. The new category has a center $\mathbf{m}$, a direction vector $\mathbf{d}$, a radius r, an axis ratio μ, and label l that are partially randomly generated as follows: For every dimension of the input feature space (M_a dimensions total) we generate a random number uniformly distributed in the interval [0, 1]; we assign $\mathbf{m}$ these values, the direction vector $\mathbf{d}$ is assigned zeros (i.e. it will act as if it is a circle, although μ could be mutated to a value other than 1 in the next generations), the axis ratio μ is assigned 1, the radius r is given a random numbers in the interval [0, 1], and the label of this newly created category is chosen randomly amongst the N_b categories of the pattern classification task under consideration. A chromosome does not add a category if the addition of this category results in number of categories for this chromosome that exceeds the designated maximum number of categories $Cat_{\max}$.

Sub-step 4f (More Details): The operator Cat_{del} deletes one of the categories of every chromosome created in step 4e with probability $P(Cat_{del})$. A chromosome does not delete a category if the deletion of this category results in the number of categories for this chromosome to fall below the designated minimum number of categories $Cat_{\min}$.

Sub-Step 4g (More Details): In GEAM, every chromosome created by step 4f gets mutated as follows: with probability $P(mut)$ every category is mutated. If a category is chosen, then every component of $\mathbf{m}$ gets mutated by adding to it a small number. This number is drawn from a Gaussian distribution with mean 0 and standard deviation 0.01. If the component of

the chosen vector becomes smaller than 0 or greater than 1 (after mutation), it is set back to 0 or 1, respectively. Furthermore, the category's axis ratio μ or radius r is selected (50% probability); we then add a small number drawn from a Gaussian distribution to the selected item with the same rules as above, here though, if μ gets greater than 1 we set it to one, otherwise, if it becomes zero or less we set its value to 0.0001, also, if the radius r becomes zero or less wet it back to 0.0001. Notice that mutation is applied on level 2 of the chromosome structure, but the label of the chromosome is not mutated (the reason being that our initial GA population consists of trained EAMs, and consequently we have a lot of confidence in the labels of the categories that these trained EAMs have discovered through the EAM training process).

Step 5 (More Details): Obvious, no more details are needed.

4.1.1 GEAM Experiments and Results

We used the same default set of parameters used for GFAM to run all the experiments of GEAM and the results were very good. Hence, in GEAM's case we avoided the experimentation (applied to GFAM) to choose good default values for the GA parameters. Hence, GEAM is produced by first initializing a population of 20 trained EAM networks (they were trained with different values of the baseline vigilance parameter and different orders of training pattern presentations), and by evolving them for 500 generations. In particular, the GA parameters used for the creation of GEAM were: $\rho_a^{\min} = 0.1$, $\rho_a^{\max} = 0.95$, $\beta_a = 0.1$, $Pop_{size} = 20$, $Gen_{\max} = 500$, $NC_{best} = 3$, $Cat_{\min} = 1$, $Cat_{\max} = 300$, $P(Cat_{add}) = 0.1$, $P(Cat_{del}) = 0.1$, $P(mut) = 5/Na$. GEAM is the EAM network that attains the highest value of the fitness function at generation 500 of the evolutionary process.

4.1.1.1 GEAM Performance

In this section we are reporting the performance of GEAM on each one of the database problems, described in Section 3.1.2.1. Similar to that of GFAM, the performance of

GEAM is assessed by reporting the size of GEAM and the accuracy attained by GEAM on the test set. The results are reported in Table 4-1. In Table 4-1, the first column is the index of the database that we are experimenting with. The second column is the actual database name, as reported in section 3.1.2., and summarized in Table 3-3. Columns 3 and 4 contain the size and accuracy of the GEAM network for the designated database. The performance of GEAM, as it is evidenced by the results in Table 4-1, is verified by some obvious observations. For instance, GEAM's performance on databases 1-12 (Gaussian datasets of known amount of overlap) is nearly optimal; for example the best performance on the G6c-40 problem (6 class Gaussian dataset of 40% overlap) is a classifier with 6 categories and 60% correct classification, and GEAM is a classifier with 6 categories and 59.35% of correct classification. Similarly, in the CINS problem the optimal classifier would require 2 categories and attain a 100% correct classification; GEAM is a 2 category classifier exhibiting a 99.9% of correct classification. Finally, two of the real problems reported here, MOD-IRIS and PAGE, also gave very good results attaining 94.8% and 94.12% of correct classification, while creating only two and three categories, respectively.

4.1.1.2 Performance Comparisons of GEAM and other ART Networks

We compared GEAM's performance with the performance of other ART networks, such as ssEAM, ssEAM, ssGAM, and safe micro-ARTMAP. The comparisons of GEAM and the aforementioned ART networks are depicted in Table 4-1, where the first column is the index of the database that we are experimenting with. The second column is the actual database name, as reported in earlier sections. Columns 3-10 of Table 4-1 contain the performance of the designated ART networks. The performance reported includes the accuracy of the "best" ART network on the test set. The performance also includes the number of categories created of the designated ART network. The reported numbers of accuracy and size of the "best" network correspond to the ART network that attained the

highest value of the fitness function (this value was computed based on the accuracy of the ART network on the cross-validation set, and on the size of the ART network). Note, that for networks, other than GEAM, the "best" ART network was determined after extensive experimentation with the ART network's parameter values (e.g., in ssEAM the best network was determined after training ssEAM networks with different values of the choice parameter, vigilance parameter, order of pattern presentation, and amount of mixture of labels allowed within a category; a total of more than 20,000 ssEAM networks were trained and their performance was examined). On the other hand, the performance of the GEAM is the one calculated after the evolution of 20 EAM trained networks for 500 generations with GA values as indicated in Section 3.2.2.

According to the results in Table 4-1, in all instances (except minor exceptions) the accuracy of GEAM (generalization performance) is higher than the accuracy of the other ART network (where ART is ssEAM, ssEAM, ssGAM or safe micro-ARTMAP). According to the results in Table 4-1, in all instances (with no exceptions) the size of GEAM is smaller than the size of the other ART network (where ART is ssEAM, ssEAM, ssGAM or safe micro-ARTMAP), sometimes even by a factor of 12. For example, the generalization performance of GEAM can be as 13% better than the generalization performance of ssFAM, while its size can be by a factor of 4 times smaller than the size of ssFAM. Also, the generalization performance of GEAM can be as 15% better than the generalization performance of ssEAM, while its size can be by a factor of 6.5 times smaller than the size of ssEAM. Furthermore, the generalization performance of GEAM can be as 9% better than the generalization performance of ssGAM, while its size can be by a factor of 12 times smaller than the size of ssGAM. Finally, the generalization performance of GEAM can be as 10% better than the generalization performance of safe micro-ARTMAP, while its size can be by a factor of 4 times smaller than the size of safe micro-ARTMAP.

The comparison results between GEAM and the other ART networks are also pictorially depicted in figures 4-2a to 4-2d. In each one of these figures we are showing the accuracy of GEAM, and that of one other network (e.g., ssEAM). In the same figure we are also showing the size of the GEAM and that of one other ART network. This way the one-to-one comparison of the GEAM and the other ART network can be quickly assessed.

What is most worth pointing out is that the better performance of GEAM is attained with reduced computations compared with the computations needed by the alternate methods (ssEAM, ssEAM, ssGAM, safe micro-ARTMAP). Specifically, the performance attained by ssEAM, ssEAM, ssGAM and the safe micro-ARTMAP required training these networks for a large number of network parameter settings (at least 20,000 experiments) and then choosing the network that achieved the higher value for the fitness function that we introduced earlier in Section 3.2.2. Of course, one can argue that such an extensive experimentation with these networks might not be needed, especially if one is familiar with the functionality of these networks and chooses to experiment only with a limited set of network parameter values. However, the practitioner in the field might lack the expertise to carefully choose the network parameters to experiment with, and consequently might need to experiment extensively to come up with a good network. In Appendix B, we show, in more detail, how more computationally efficient GFAM is compared to ssEAM, ssEAM, ssGAM and safe micro-ARTMAP. The computational complexity of GEAM is given by similar equations as the GFAM computational complexity, calculated in Appendix B. The comparison between GFAM (and GEAM) and the rest of the ART networks is based under the assumption that extensive parameter experimentation with the network parameters of ssEAM, ssEAM, ssGAM or safe micro-ARTMAP is needed to obtain a good performing ssEAM, ssEAM, ssGAM or safe micro-ARTMAP network, respectively.

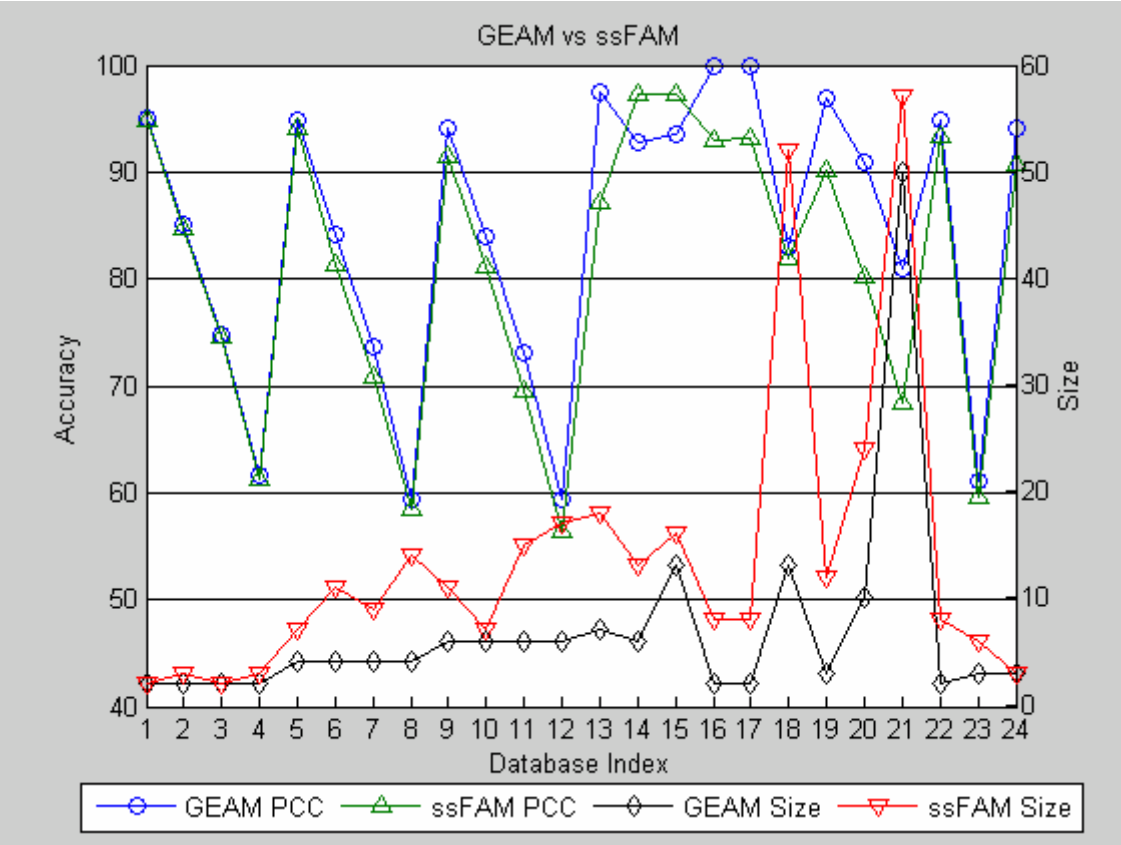

Figure 4-2a: Performance and Size comparison of GEAM vs ssFAM

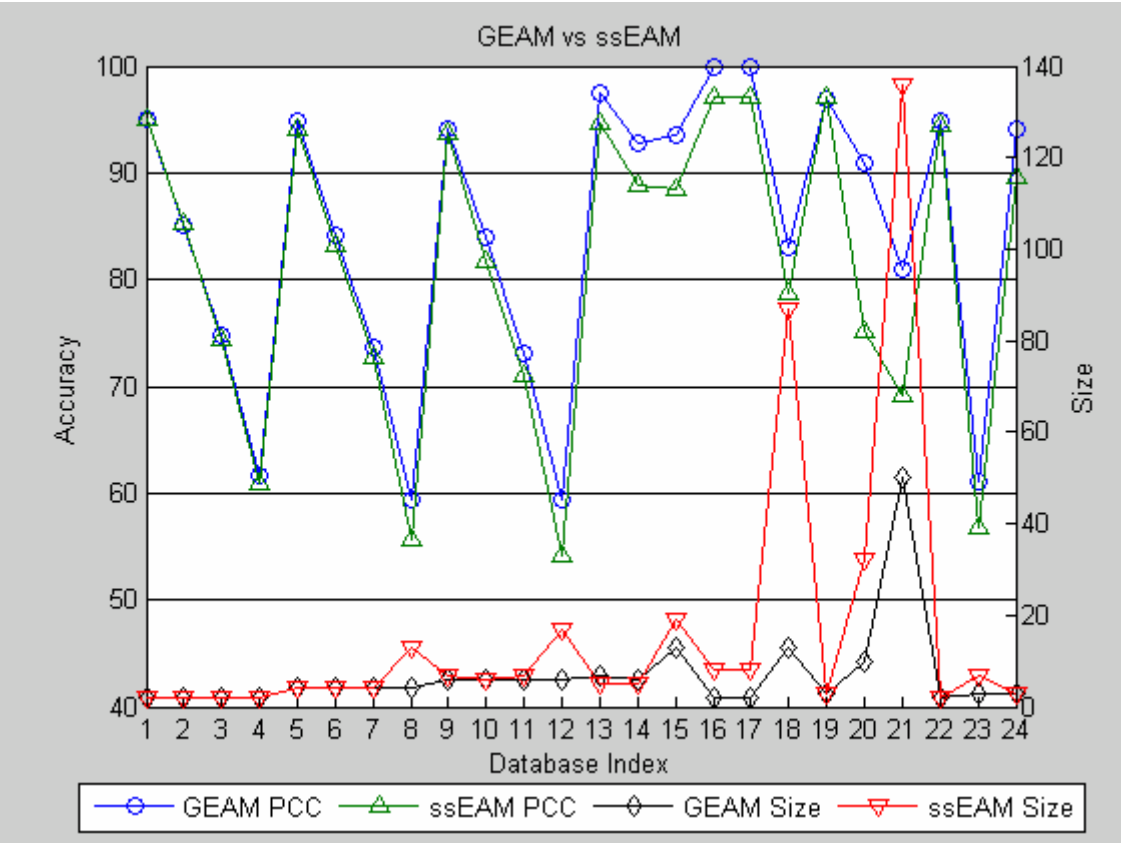

Figure 4-2b: Performance and Size comparison of GEAM vs ssEAM

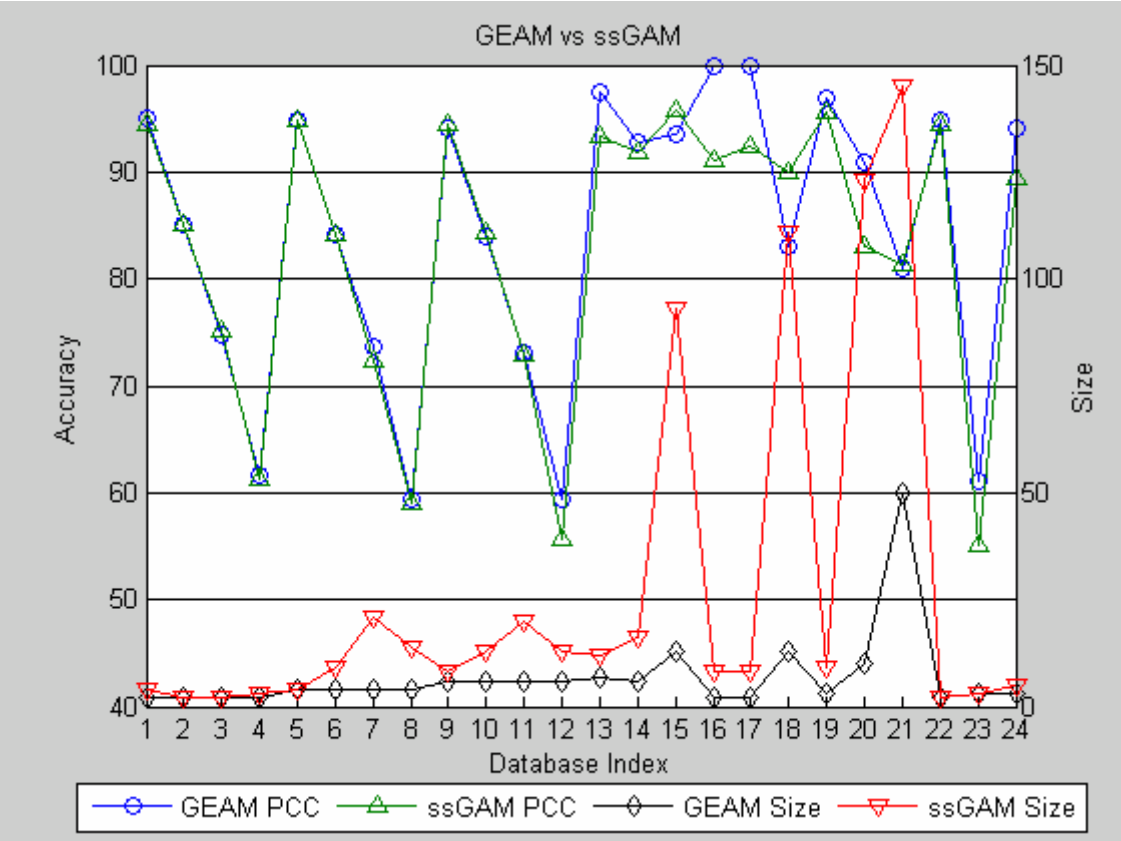

Figure 4-2c: Performance and Size comparison of GEAM vs ssGAM

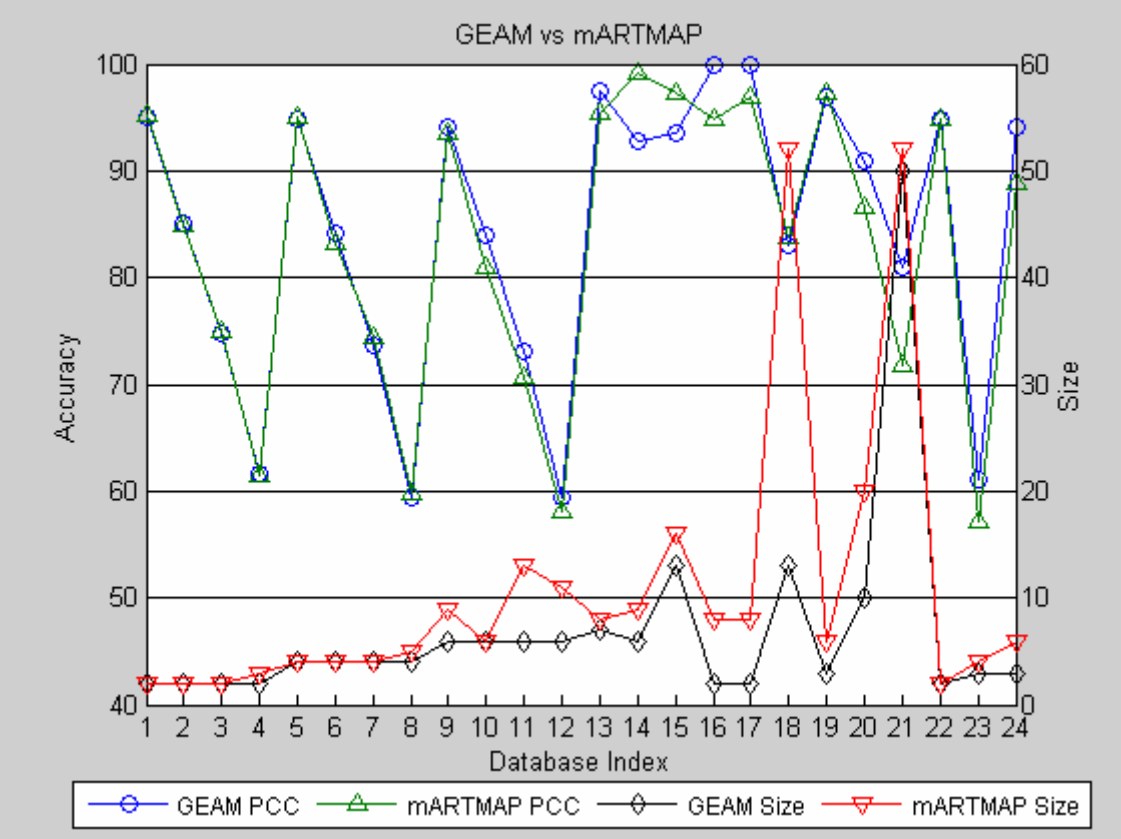

Figure 4-2d: Performance and Size comparison of GEAM vs microARTMAP

Table 4-1: Accuracy and size results achieved by GEAM and other ART networks. Note that:Safe uAM: Safe microARTMAP; FAM: Fuzzy ARTMAP; EAM: Ellipsoidal ARTMAP; GAM: Gaussian ARTMAP; ss*: semi-supervised version

	Database Name	GEAM		Safe μAM		ssFAM		ssEAM		ssGAM	
1	G2c-05	95	2	95.22	2	94.90	2	94.94	2	94.48	4
2	G2c-15	85.12	2	85.00	2	84.80	3	85.20	2	85.04	2
3	G2c-25	74.74	2	74.98	2	74.60	2	74.50	2	75.10	2
4	G2c-40	61.6	2	61.40	3	61.34	3	60.98	2	61.30	3
5	G4c-05	94.8	4	95.04	4	94.10	7	94.14	4	94.80	4
6	G4c-15	84.22	4	83.28	4	81.40	11	83.20	4	84.24	9
7	G4c-25	73.7	4	74.50	4	70.80	9	72.72	4	72.32	21
8	G4c-40	59.5	4	59.76	5	58.48	14	55.62	13	59.10	14
9	G6c-05	94.1247	6	93.57	9	91.42	11	93.80	7	94.40	8
10	G6c-15	84.0328	6	80.92	6	81.11	7	81.80	6	84.35	13
11	G6c-25	73.0416	6	70.74	13	69.62	15	71.10	7	72.86	20
12	G6c-40	59.3525	6	58.03	11	56.35	17	54.21	17	55.65	13
13	4Ci/Sq	97.4	7	95.42	8	87.23	18	94.68	5	93.4	12
14	4Sq/Sq	92.7667	6	99.12	9	97.24	13	88.89	5	91.78	16
15	7Sq	93.46	13	97.22	16	97.26	16	88.5	19	95.83	93
16	1Ci/Sq	99.9	2	94.76	8	92.97	8	97.02	8	91.02	8
17	1Ci/Sq/0.3:0.7	99.9	2	96.82	8	93.21	8	97.13	8	92.33	8
18	5Ci/Sq	83.0333	13	83.83	52	81.95	52	78.68	87	90.02	111
19	2Ci/Sq/20:30:50	96.8667	3	97.22	6	90.24	12	97.01	3	95.6	9
20	7SqWN	90.8273	10	86.67	20	80.15	24	75.23	32	83.11	123
21	5Ci/SqWN	81	50	71.72	52	68.39	57	69.2	136	81.3	145
22	MOD-IRIS	94.8125	2	94.92	2	93.41	8	94.54	2	94.54	2
23	ABALONE	61.0018	3	57.18	4	59.52	6	56.80	7	55.10	3
24	PAGE	94.1219	3	88.82	6	90.63	3	89.54	3	89.34	5

4.1.2 Summary/Conclusions

In this section, we have introduced, yet another, method of solving the category proliferation problem in ART. This method relies on evolving a population of trained ART

networks, and more specifically Ellipsoidal ARTMAP (EAM) neural networks. The evolution of trained EAMs creates an ART network, referred to as GEAM.

In chapter 3 we defined a methodology of evolving trained FAM networks, resulting in GFAM. This methodology was also applied successfully for the evolution of EAM networks, resulting in GEAM. In chapter 3 we experimented with a number of databases that helped us identify good default parameter settings for the evolution of FAM The same parameters and settings used in chapter 3 for the evolution of FAM networks (GFAM) were also used for the evolution of EAM networks (GEAM).

Our experiments with GEAM indicate that GEAM is superior to a number of other ART techniques (ssEAM, ssEAM, ssGAM, safe micro-ARTMAP) that have been introduced into the literature to address the category proliferation problem in EAM. More specifically, GEAM gave a better generalization performance (in almost all problems tested) and a smaller size network (in all problems tested), compared to these other ART techniques. What is also worth mentioning is that GEAM outperformed these other ART techniques by requiring only a fraction of the computations needed by these other networks.

4.2 Genetic Gaussian ARTMAP (GGAM)

GGAM (Genetic Gaussian ARTMAP) is an evolved GAM network that is produced by applying, repeatedly, genetic operators on an initial population of trained GAM networks. GGAM uses tournament selection with elitism, as well as genetic operators, including crossover and mutation. In addition, GGAM uses two special operators, Cat_{add} and Cat_{del}.

To better understand how GGAM is designed we resort to a step-by-step description of this design. Please refer to the terminology introduced in Appendix A before you dwell in the ste-by-step description of EAM. The design of GGAM can be articulated through a sequence of steps, defined succinctly below, and explained in detail later.

Step 1: Initialize Pop_{size} number of GAM networks, each one of them operating with a different value for the baseline vigilance parameter $\overline{\rho}_a$, and possibly different orders of pattern presentation.

Step 2: Train each one of the Pop_{size} initialized GAM networks, using the training set for a maximum number of iterations ($Gen_{\max}$).

Step 3: Convert the Pop_{size} trained GAM networks into chromosomes. Crop all chromosomes so that no one-point categories exist.

Step 4: Evolve the chromosomes of the current generation by executing the following sub-steps:

Sub-Step 4a: Calculate fitness for all chromosomes of the current generation.

Sub-Step 4b: Initialize an empty generation (referred to as *temporary generation*).

Sub-Step 4c: Move the best (NC_{best}) chromosomes from the current generation to the temporary generation.

Sub-Step 4d: Select chromosomes for crossover from the current generation and thus further populate the temporary generation.

Sub-Step 4e: With a probability $P(Cat_{add})$ apply the Cat_{add} operator on every individual generated in sub-step 4d.

Sub-Step 4f: With a probability $P(Cat_{del})$ apply the Cat_{del} operator on every individual generated in sub-step 4e.

Sub-Step 4g: With a probability $P(Mut)$ apply the mutation operator on every individual generated in sub-step 4f.

Sub-Step 4h: Replace the current generation with the members of the temporary generation

Step 5: If evolution has reached the maximum number $Gen_{\max}$ of iterations, then calculate the performance of the best-Fitness GAM network on the test set and report classification

accuracy and number of categories that this Best-Fitness GAM network possesses. If the maximum number of iterations has not been reached yet, go to step 4 to evolve one more population of chromosomes

Each one of the aforementioned steps of the algorithm is now described in more detail, as needed.

Step 1 (More Details): The algorithm starts by training Pop_{size} GAM networks, each one of them trained with a different value of the baseline vigilance parameter $\overline{\rho}_a$. In particular, we first define $\overline{\rho}_a^{inc} = \frac{\overline{\rho}_a^{\max} - \overline{\rho}_a^{\min}}{Pop_{size} - 1}$, and then the baseline vigilance parameter of every network is determined by the equation $\overline{\rho}_a^{\min} + i * \overline{\rho}_a^{inc}$, where $i \in \{0, 1, ..., Pop_{size} - 1\}$. In our experiments with GFAM we chose $\overline{\rho}_a^{\min} = 0.1$, and $\overline{\rho}_a^{\max} = 0.95$. Meanwhile, GGAM allows the user to change the order of pattern presentation automatically and randomly.

Step 2 (More Details): We assume that the reader is familiar of how training GAM networks is accomplished, and thus the details here are omitted.

Step 3 (More Details): Once the Pop_{size} networks are trained they need to be converted to chromosomes, so that they can be manipulated by the genetic operators. GGAM uses a mix of real numbers representation to encode the networks. Each GAM chromosome consists of two levels, level 1 containing all the categories of the GAM network, and level 2 containing the mean, the standard deviation and the number of encoded nodes (during training), as well as the label of that category (see Figure 4-3). We denote the category of a trained GAM network with index p $(1 \le p \le Pop_{size})$ by $\mathbf{w}_j^a(p)$, where $\mathbf{w}_j^a = (\boldsymbol{\mu}_j, \boldsymbol{\sigma}_j, n_j)$, and the label of this category by $l_j(p)$ for $1 \le j \le N_a(p)$. In this step we are also eliminating single-point categories in the trained GAM networks, referred to as cropping the chromosomes. Since our

ultimate objective is to design a GAM network that reduces the network size and improves generalization we are discouraging at this stage the creation of single-point categories.

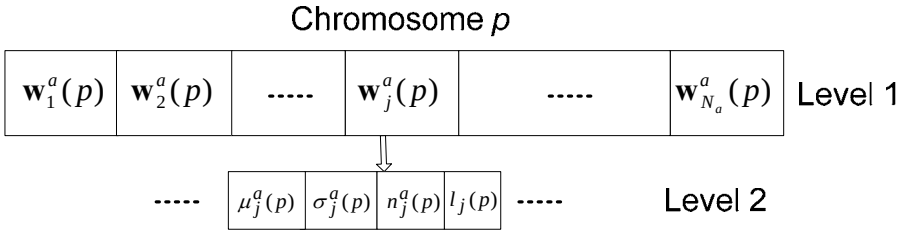

Figure 4-3: GGAM Chromosome Structure

Step 4 (More Details): In this step the GFAM applies a GA to the population of trained FAMs.

Sub-step 4a (More Details): Calculate the fitness of each chromosome (trained GAM). This is accomplished by feeding into each trained GAM the validation set and by calculating the percentage of correct classification exhibited by each one of these trained GAM networks. In particular, if $PCC(p)$ designates the percentage of correct classification, exhibited by the *p*-th GAM, and this GAM network possesses $N_a(p)$ nodes in its category representation layer, then its fitness function value is defined by:

$$Fit(p) = \frac{(Cat_{\max} - N_a(p)) \cdot PCC^2(p)}{\frac{100}{Cat_{\min}} - \frac{PCC(p)}{N_a(p)} + \varepsilon} \qquad 4\text{-}2$$

where, $Cat_{\min}$ and $Cat_{\max}$ are the minimum and maximum number of categories that a FAM network is allowed to have during the evolutionary process ($Cat_{\min}$ is chosen equal to 1, or equal to the number of classes in the classification problem under consideration, while $Cat_{\max}$ is chosen to be a relatively large number for the classification problem at hand). The constant ε in the denominator of the above equation is a small positive constant and it is needed to make sure that the denominator would not be zero in the case when $N_a(p) = Cat_{\min}$ and $PCC(p) = 100$.

Sub-step 4b (More Details): Obvious, no further explanations are needed.

Sub-step 4c (More Details): The algorithm searches for the best NC_{best} chromosomes from the current generation and copies them to the temporary generation.

Sub-step 4d (More Details): The remaining $Pop_{size} - NC_{best}$ chromosomes in the temporary generation are created by crossing over two parents from the current generation. The parents are chosen using the deterministic tournament selection method, as follows: Randomly select two groups of four chromosomes each from the current generation, and use as a parent from each group the chromosome with the best fitness value in the group. If it happens that from both groups the same chromosome is chosen then we choose from one of the groups the chromosome with the second best fitness value. If two parents with indices p, p' are crossed over two random numbers n, n' are generated from the index sets $\{1, 2, ..., N_a(p)\}$ and $\{1, 2, ..., N_a(p')\}$, respectively. Then, all the categories with index greater than index n' in chromosome with index p' and all the categories with index less than index n in the category with index p are moved into an empty chromosome within the temporary generation. Notice that crossover is done on level 1 of the chromosome. This operation is pictorially illustrated in the Figure 3-2.

Sub-step 4e (More Details): The operator Cat_{add} adds a new category to every chromosome created in step 4d with probability $P(Cat_{add})$. The new category has a mean vector $\boldsymbol{\mu}$, a standard deviation vector $\boldsymbol{\sigma}$, a probability (number of encoded patterns) n, and label l that are randomly generated as follows: For every dimension of the input feature space (M_a dimensions total) we generate a random number uniformly distributed in the interval [0, 1]; each one of these numbers is chosen to be one of the components of $\boldsymbol{\mu}$. In a similar fashion, we choose the components of the standard deviation vector $\boldsymbol{\sigma}$, whoever $\boldsymbol{\sigma}$'s values are chosen in the interval [0.1,0.9]. Furthermore, n is chosen to be a positive real random number, uniformly distributed, while the label of this newly created category is chosen

randomly amongst the N_b categories of the pattern classification task under consideration. A chromosome does not add a category if the addition of this category results in number of categories for this chromosome that exceeds the designated maximum number of categories $Cat_{\max}$.

Sub-step 4f (More Details): The operator Cat_{del} deletes one of the categories of every chromosome created in step 4e with probability $P(Cat_{del})$. A chromosome does not delete a category if the deletion of this category results in the number of categories for this chromosome to fall below the designated minimum number of categories $Cat_{\min}$.

Sub-Step 4g (More Details): In GGAM, every chromosome created by step 4f gets mutated as follows: with probability $P(mut)$ every category is mutated. If a category is chosen, its mean vector $\boldsymbol{\mu}$ or standard deviation vector $\boldsymbol{\sigma}$ is selected randomly (50% probability). Then every component of this selected vector gets mutated by adding to it a small number. This number is drawn from a Gaussian distribution with mean 0 and standard deviation 0.01. If the component of the chosen vector becomes smaller than 0 or greater than 1 (after mutation), it is set back to 0 or 1, respectively. Also the probability of the category n gets mutated by adding a small number drawn from a Gaussian distribution to the selected item with the same rules as above. Notice that mutation is applied on level 2 of the chromosome structure, but the label of the chromosome is not mutated (the reason being that our initial GA population consists of trained GAMs, and consequently we have a lot of confidence in the labels of the categories that these trained GAMs have discovered through the GAM training process).

Step 5 (More Details): Obvious, no more details are needed.

4.2.1 GGAM Experiments and Results

We used the same default set of parameters used for GFAM to run all the experiments of GGAM and the results were very good. Hence, in GGAM's case we avoided the

experimentation (applied to GFAM) to choose good default values for the GA parameters. Hence, GGAM is produced by first initializing a population of 20 trained GAM networks (they were trained with different values of the baseline vigilance parameter and different orders of training pattern presentations), and by evolving them for 500 generations. In particular, the GA parameters used for the creation of GGAM were: $\rho_a^{\min} = 0.1$, $\rho_a^{\max} = 0.75$, $\beta_a = 0.1$, $Pop_{size} = 20$, $Gen_{\max} = 500$, $NC_{best} = 3$, $Cat_{\min} = 1$, $Cat_{\max} = 300$, $P(Cat_{add}) = 0.1$, $P(Cat_{del}) = 0.1$, $P(mut) = 5/Na$. GGAM is the GAM network that attains the highest value of the fitness function at generation 500 of the evolutionary process.

4.2.1.1 GGAM Performance

In this section we are reporting the performance of GGAM on each one of the database problems, described in Section 3.1.2.1. Similar to that of GFAM, the performance of GGAM is assessed by reporting the size of GGAM and the accuracy attained by GGAM on the test set. The results are reported in Table 4-2. In Table 4-2, the first column is the index of the database that we are experimenting with. The second column is the actual database name, as reported in section 3.1.2. and summarized in Table 3-3. Columns 3 and 4 contain the size and accuracy of the GGAM network for the designated database. The performance of GGAM, as it is evidenced by the results in Table 4-2, is verified by some obvious observations. For instance, GGAM's performance on databases 1-12 (Gaussian datasets of known amount of overlap) is nearly optimal; for example the best performance on the G6c-40 problem (6 class Gaussian dataset of 40% overlap) is a classifier with 6 categories and 60% correct classification, and GGAM is a classifier with 6 categories and 59.43% of correct classification. Similarly, in the CINS problem the optimal classifier would require 2 categories and attain a 100% correct classification; GGAM is a 2 category classifier exhibiting a 99.77% of correct classification. Finally, two of the real problems reported here,

MOD-IRIS and PAGE, also gave very good results 94.83% and 95.02% of correct classification respectively, by creating two categories only.

4.2.1.2 Performance Comparisons of GGAM and other ART Networks

As it was the case with GFAM, we compared GGAM's performance with the performance of the following networks: ssEAM, ssEAM, ssGAM, and safe micro-ARTMAP. The comparisons of GGAM and the aforementioned ART networks are depicted in Table 4-2, where the first column is the index of the database that we are experimenting with. The second column is the actual database name, as reported in earlier sections. Columns 3-10 of Table 4-2 contain the performance of the designated ART networks. The performance reported includes the accuracy of the "best" ART network on the test set. The performance also includes the number of categories created of the designated ART network. The reported numbers of accuracy and size of the "best" network correspond to the ART network that attained the highest value of the fitness function (this value was computed based on the accuracy of the ART network on the cross-validation set, and on the size of the ART network). Note, that for networks, other than GGAM, the "best" ART network was determined after extensive experimentation with the ART network's parameter values (e.g., in ssFAM the best network was determined after training ssFAM networks with different values of the choice parameter, vigilance parameter, order of pattern presentation, and amount of mixture of labels allowed within a category; a total of 20,000 ssFAM networks were trained and their performance was examined). On the other hand, the performance of the GGAM is the one calculated after the evolution of 20 GAM trained networks for 500 generations with GA values as indicated in Section 3.2.2.

According to the results in Table 4-2, in all instances (except minor exceptions) the accuracy of GGAM (generalization performance) is higher than the accuracy of the other ART network (where ART is ssEAM, ssEAM, ssGAM or safe micro-ARTMAP). According

to the results in Table 4-2, in all instances (with no exceptions) the size of GGAM is smaller than the size of the other ART network (where ART is ssEAM, ssEAM, ssGAM or safe micro-ARTMAP), sometimes even by a factor of 12. For example, the generalization performance of GGAM can be as 15% better than the generalization performance of ssFAM, while its size can be by a factor of 4 times smaller than the size of ssFAM. Also, the generalization performance of GGAM can be as 14% better than the generalization performance of ssEAM, while its size can be by a factor of 4 times smaller than the size of ssEAM. Furthermore, the generalization performance of GGAM can be as 8% better than the generalization performance of ssGAM, while its size can be by a factor of 12 times smaller than the size of ssGAM. Finally, the generalization performance of GGAM can be as 10% better than the generalization performance of safe micro-ARTMAP, while its size can be by a factor of 4 times smaller than the size of safe micro-ARTMAP.

The comparison results between GGAM and the other ART networks are also pictorially depicted in figures 4-4a to 4-4d. In each one of these figures we are showing the accuracy of GGAM, and that of one other network (e.g., ssEAM). In the same figure we are also showing the size of the GGAM and that of one other ART network. This way the one-to-one comparison of the GGAM and the other ART networks can be quickly assessed.

What is most worth pointing out is that the better performance of GGAM is attained with reduced computations compared with the computations needed by the alternate methods (ssEAM, ssEAM, ssGAM, safe micro-ARTMAP). Specifically, the performance attained by ssEAM, ssEAM, ssGAM and the safe micro-ARTMAP required training these networks for a large number of network parameter settings (at least 20,000 experiments) and then choosing the network that achieved the higher value for the fitness function that we introduced earlier in Section 3.2.2. Of course, one can argue that such an extensive experimentation with these networks might not be needed, especially if one is familiar with the functionality of these

networks and chooses to experiment only with a limited set of network parameter values. However, the practitioner in the field might lack the expertise to carefully choose the network parameters to experiment with, and consequently might need to experiment extensively to come up with a good network. The computational complexity of GGAM is given by similar equations as the GFAM computational complexity, calculated in Appendix B. The comparison between the computational complexity GFAM (and GGAM) and the rest of the ART networks is based under the assumption that extensive parameter experimentation with the network parameters of ssEAM, ssEAM, ssGAM or safe micro-ARTMAP is needed to obtain a good performing ssEAM, ssEAM, ssGAM or safe micro-ARTMAP network, respectively.

Table 4-2: Accuracy and size results achieved by GGAM and other ART networks. Note that:Safe uAM: Safe microARTMAP; FAM: Fuzzy ARTMAP; EAM: Ellipsoidal ARTMAP; GAM: Gaussian ARTMAP; ss*: semi-supervised version

	Database Name	GGAM		Safe μAM		ssFAM		ssEAM		ssGAM	
1	G2c-05	95.2	2	95.22	2	94.90	2	94.94	2	94.48	4
2	G2c-15	85.2	2	85.00	2	84.80	3	85.20	2	85.04	2
3	G2c-25	75.06	2	74.98	2	74.60	2	74.50	2	75.10	2
4	G2c-40	61.64	2	61.40	3	61.34	3	60.98	2	61.30	3
5	G4c-05	94.82	4	95.04	4	94.10	7	94.14	4	94.80	4
6	G4c-15	84.28	4	83.28	4	81.40	11	83.20	4	84.24	9
7	G4c-25	74.74	4	74.50	4	70.80	9	72.72	4	72.32	21
8	G4c-40	59.82	4	59.76	5	58.48	14	55.62	13	59.10	14
9	G6c-05	94.4644	6	93.57	9	91.42	11	93.80	7	94.40	8
10	G6c-15	84.8122	6	80.92	6	81.11	7	81.80	6	84.35	13
11	G6c-25	74.0008	6	70.74	13	69.62	15	71.10	7	72.86	20
12	G6c-40	59.4325	6	58.03	11	56.35	17	54.21	17	55.65	13
13	4Ci/Sq	97.2	5	95.42	8	87.23	18	94.68	5	93.4	12
14	4Sq/Sq	93.0667	5	99.12	9	97.24	13	88.89	5	91.78	16
15	7Sq	95.1333	10	97.22	16	97.26	16	88.5	19	95.83	93
16	1Ci/Sq	99.7667	2	94.76	8	92.97	8	97.02	8	91.02	8
17	1Ci/Sq/0.3:0.7	99.9	2	96.82	8	93.21	8	97.13	8	92.33	8
18	5Ci/Sq	87.7667	39	83.83	52	81.95	52	78.68	87	90.02	111
19	2Ci/Sq/20:30:50	98.9333	3	97.22	6	90.24	12	97.01	3	95.6	9
20	7SqWN	89.83	10	86.67	20	80.15	24	75.23	32	83.11	123
21	5Ci/SqWN	81.34	42	71.72	52	68.39	57	69.2	136	81.3	145
22	MOD-IRIS	94.8333	2	94.92	2	93.41	8	94.54	2	94.54	2
23	ABALONE	61.5385	3	57.18	4	59.52	6	56.80	7	55.10	3
24	PAGE	95.0179	2	88.82	6	90.63	3	89.54	3	89.34	5

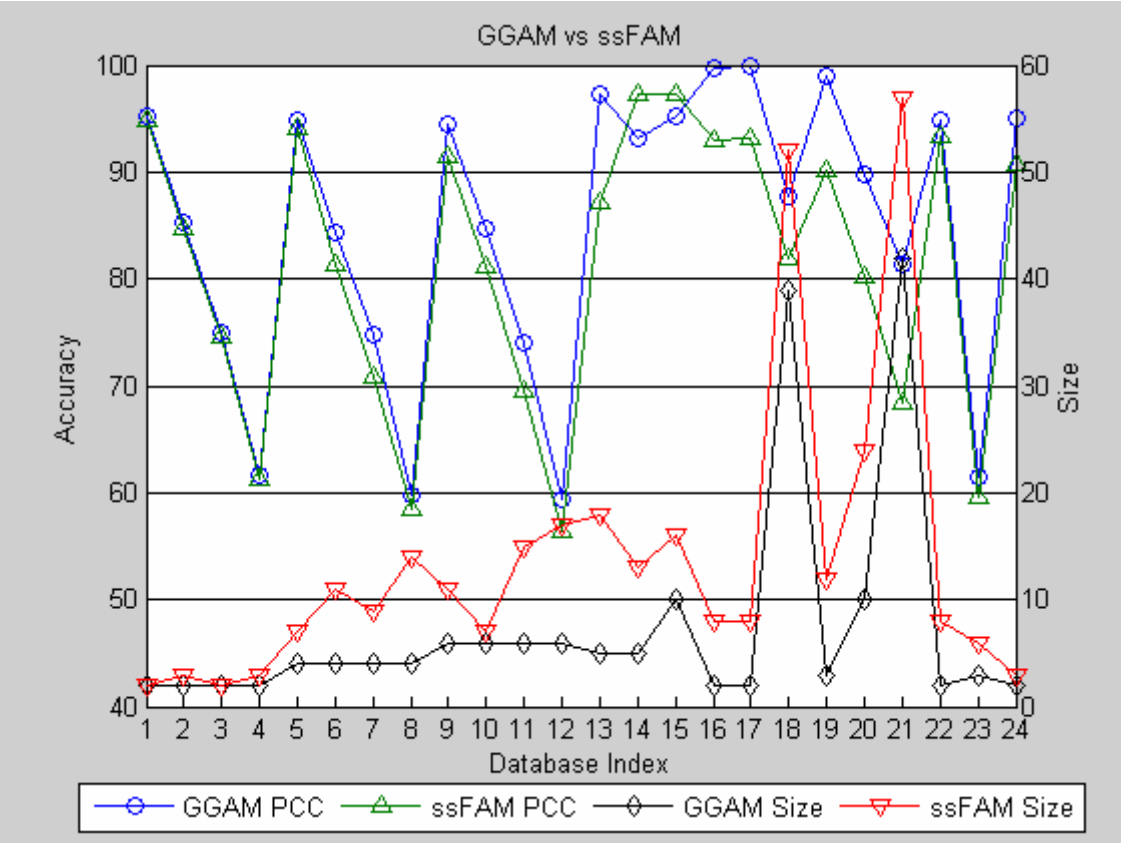

Figure 4-4a: Performance and Size comparison of GGAM vs ssFAM

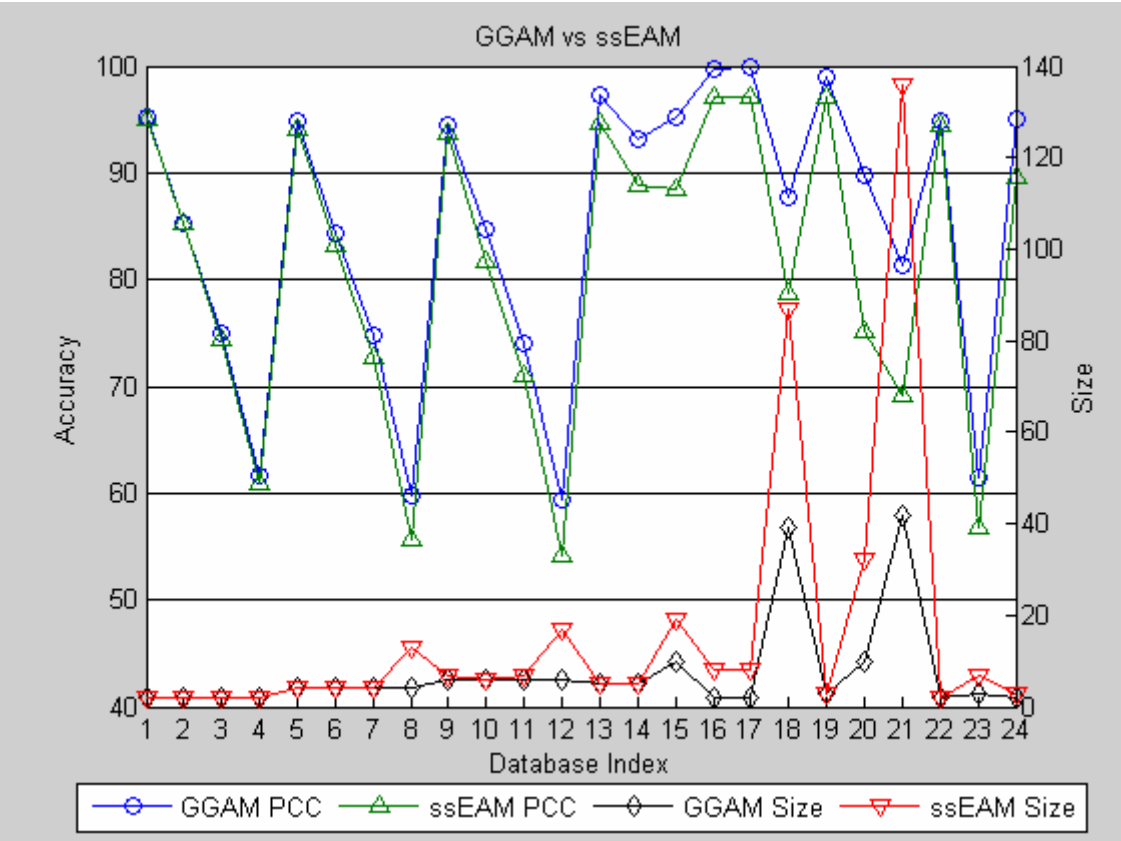

Figure 4-4b: Performance and Size comparison of GGAM vs ssEAM

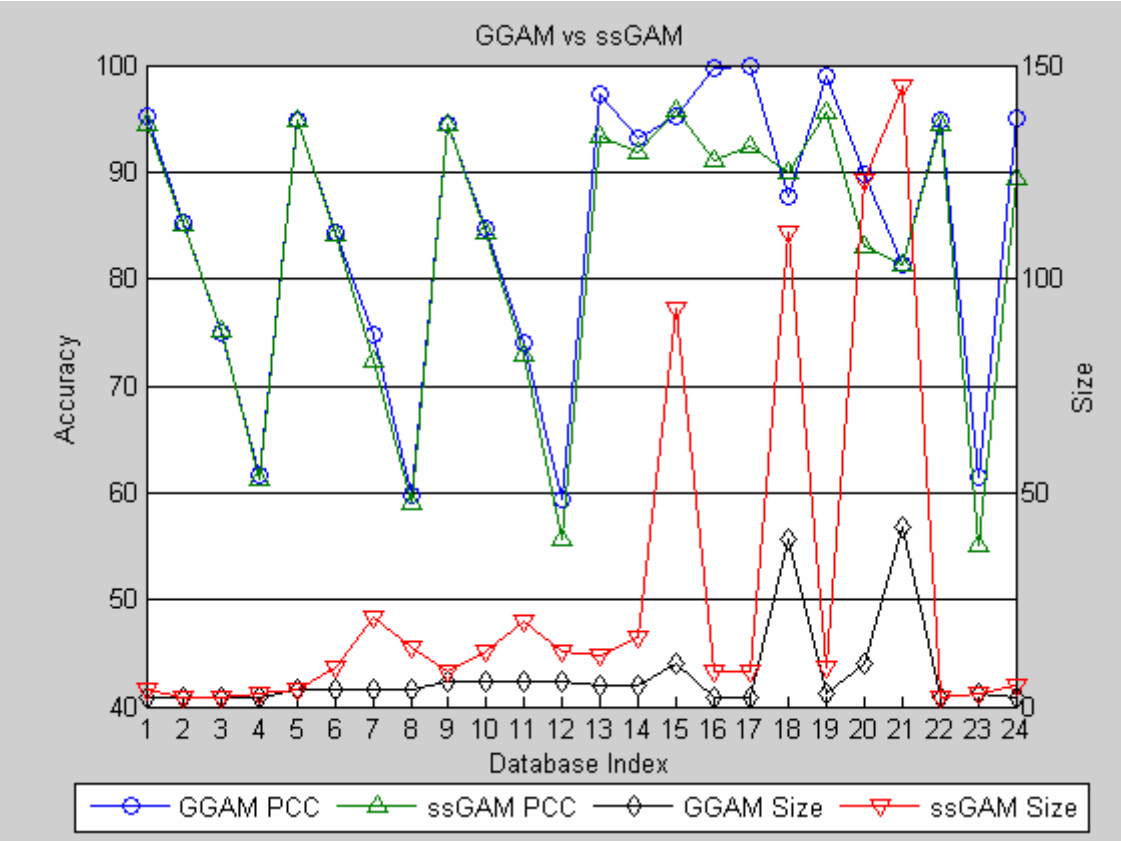

Figure 4-4c: Performance and Size comparison of GGAM vs ssGAM

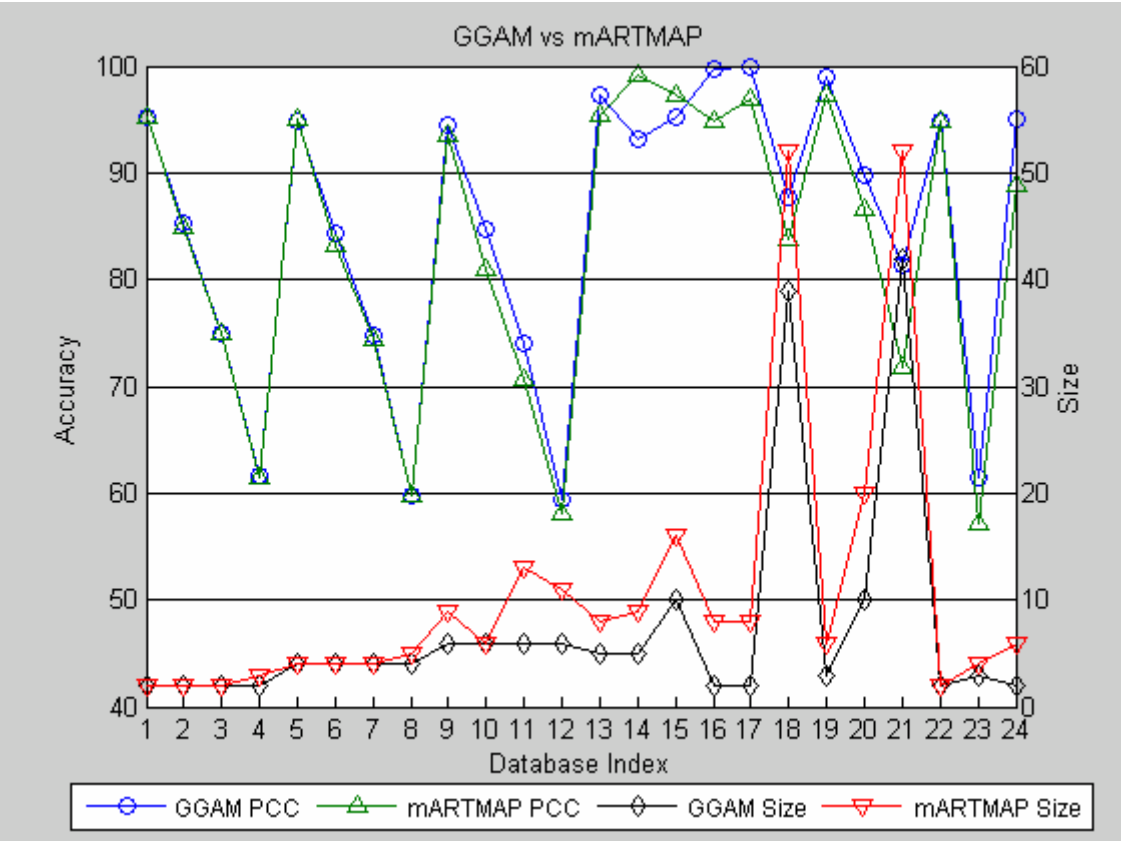

Figure 4-4d: Performance and Size comparison of GGAM vs microARTMAP

4.2.2 Summary/Conclusions

In this section, we have introduced, yet another, method of solving the category proliferation problem in ART. This method relies on evolving a population of trained ART networks, and more specifically Gaussian ARTMAP (GAM) neural networks. The evolution of trained GAMs creates an ART network, referred to as GGAM.

In chapter 3 we defined a methodology of evolving trained FAM networks, resulting in GFAM. This methodology was also applied successfully for the evolution of GAM networks, resulting in GGAM. In chapter 3 we experimented with a number of databases that helped us identify good default parameter settings for the evolution of FAM The same parameters and settings used in chapter x for the evolution of FAM networks (GFAM) were also used for the evolution of GAM networks (GGAM).

Our experiments with GGAM indicate that GGAM is superior to a number of other ART techniques (ssEAM, ssEAM, ssGAM, safe micro-ARTMAP) that have been introduced into the literature to address the category proliferation problem in GAM. More specifically, GGAM gave a better generalization performance (in almost all problems tested) and a smaller size network (in all problems tested), compared to these other ART techniques. What is also worth mentioning is that GGAM outperformed these other ART techniques by requiring only a fraction of the computations needed by these other networks.

5. UNIVERSAL ART (UART)

In all our previous genetically engineered ART architectures we only evolved one type of ART architectures, such as FAM, or EAM or GAM. It will be advantageous for some classification problem to be able to evolve a mixture of ART architectures, such as FAM and EAM, EAM and GAM, FAM and GAM, or FAM, EAM and GAM. The evolution of a mixture of ART architectures leads us to a genetically designed ART network that we call Universal ART (UART).

The motivation behind the creation of UART could be presented by a number of examples. For some datasets one of the FAM, EAM, GAM architectures will produce the best results, while for another dataset another architecture will do the best. Furthermore, it could also be the case that GAM is better at describing the input patterns in a portion of the input space, while FAM might be able to do a better job at another portion of the input space (and the same dataset). These observations gave birth to the idea of UART that does not a-priori determine which of is the best category structure (hyper-rectangle, hyper-ellisoid, Gaussian) that could best represent the data in various portions of the input pattern space.

It is reasonable to think that a problem space would not be suitable to be covered by only one of the geometrical shapes mentioned above, and this could explain the extra nodes created and the lack of accuracy attained by a specific ART architecture. The figure below supports our claim. In figure 5-1, there are three different classification problems represented in 2-dimentional space.

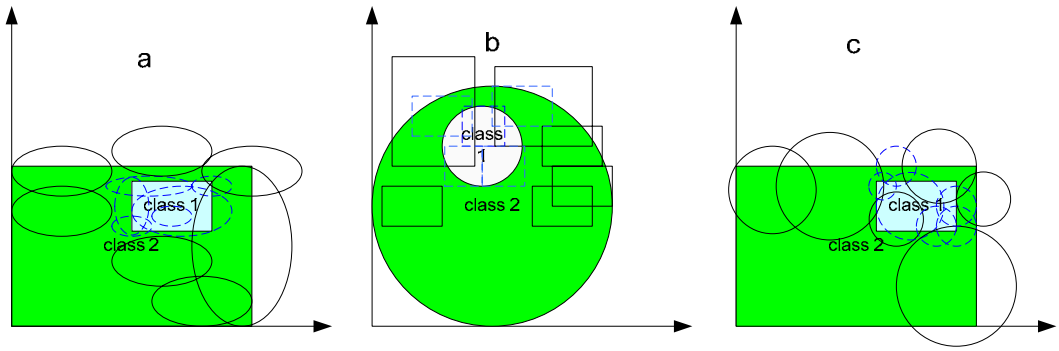

Figure 5-1: These figures show what happens when using unsuitable classifiers for a certain problem.

In figure 5-1 (a), we see a 2-class problem that is represented by two rectangles, for which the geometrical shapes created by FAM would be the best to solve, and would be wasteful to use EAM to solve it. In the other hand, figure 5-1 (b) shows an example of a 2-class problem for which the geometrical shapes created by EAM would be the best to solve, and it would be wasteful to use FAM to solve it, figure 5-1 (c) shows a similar scenario when using GAM to classify the two rectangle problem, (GAM categories are not really circles, but depicted here as such to show the point). Furthermore, there might be classification examples where one of the ART classifies will not be the best choices to use in every portion of the input space (see Figure 5-2).

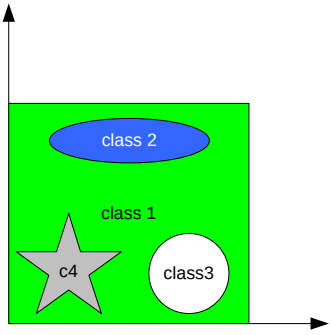

Figure 5-2: a classification problem where the boundaries can't be optimally covered by FAM, GAM or EAM's categories.

For this problem a genetically engineered ART architecture, such as UART, might perform well because it has the ability to create three different geometrical structures in the input space (see Figure 5-3). It is expected that if the input space is covered by the minimum

possible number of correct structures by an ART neural network the generalization performance of the network on unseen data might be improved too.

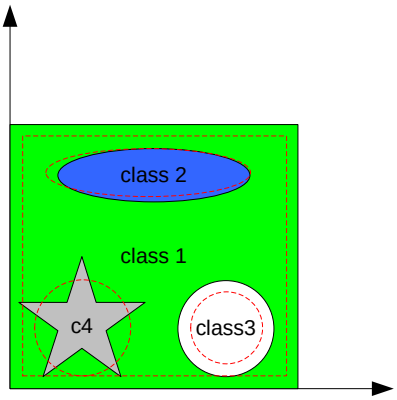

Figure 5-3: Using UART to solve the problem in figure 5-2, notice that parts of the problem space are not covered, but because UART encourages smaller size, it might sacrifice little accuracy to get optimal size.

5.1 UART Design

UART can operate in three distinct phases: *the training phase*, the *geometry selection phase* (or Genetic Phase) and the *performance phase*. In the training phase of UART, the user defines the number of the networks the system should generate, referred to as Pop_{size}. The system then creates Pop_{size} trained ART networks, half of them FAM and the other half EAM networks. These FAM and EAM networks are generated by using different values of the baseline vigilance parameter and orders of training pattern presentations, as we did for GFAM and GEAM. For the training of this initial population of Pop_{size} ART networks a list of input patterns/output labels pairs, (i.e. $\{(\mathbf{I}^1, O(\mathbf{I}^1)),...,(\mathbf{I}^r, O(\mathbf{I}^r)),...,(\mathbf{I}^{PT}, O(\mathbf{I}^{PT}))\}$), is repeatedly presented to the FAM/EAM network until it learns the required mapping. Training is over when a user defined maximum list presentation number is reached. After creating a FAM or EAM trained network UART converts it into a chromosome and saves it in the FAM/EAM network container of the UART architecture. A pictorial illustration of the UART architecture in its training phase, consisting of two independently operating FAM and EAM architectures and the associated FAM/EAM chromosome container is shown in Figure 5-4.

In the geometry selection phase, the goal is to find an ART network (UART network) which contains the best types and smallest number of ART categories (rectangles or ellipsoids, or a combination of the two) that achieves good generalization. This UART network is found by starting from an initial population of Pop_{size} UART networks and by applying the GA algorithm to this initial population, in the same way that we applied to produce GFAM and GEAM (see earlier sections). The distinct and important difference between the initial population of FAMs, and EAMs, that we started with then and the initial population of UARTs that we start with here is that the initial population of FAMs and EAMs consisted of chromosomes, each one of which contained categories of the same geometrical structure, such as rectangles or ellipsoids. Now, each chromosome in the initial population of UART starts form an initial population of ART networks, whose chromosome contains a mixture of rectangles and ellipsoids. Each member of this initial population chose the rectangles and categories contained in its chromosome randomly from the population of rectangles and categories included in the UART container from UART's training phase. It is important, to also mention that in the geometry selection phase (or genetic phase) UART is called upon to calculate its output for each input pattern in the validation data, i.e., UART is called upon to operate in the performance mode. The steps that UART is going through to produce an output label for an input pattern presented at its input during UART' performance mode are included below. The UART architecture, when it operates in the performance mode, is different than the UART architecture when it operates in the training mode (see Figure 5-5).

The UART architecture, in its performance phase, consists of three main layers. These are: the *input layer* (U_1^a), the *category representation layer* (U_2^a), and the *output layer* (U_2^b). The input layer of UART has $2M_a$ nodes, nodes numbered 1 through M_a are connected to all the nodes in U_2^a layer that represent a FAM category, while only the nodes 1 to M_a) are connected to those categories in the U_2^a layer that represent an EAM category (remember that EAM does not require complement encoding).

During the performance phase, **a** is fed to U_1^a , so that a occupies the first M_a nodes of U_1^a and its complement coded version, $\mathbf{I}$, occupies the $2M_a$ nodes of U_1^a. Layer U_2^a in UART represents all the categories that the UART network possesses, and hence the name *category representation layer*. This layer could have all the nodes as FAM categories, as EAM categories or as a mixture of both (these nodes represent categories that were randomly chosen from the mixture of FAM/EAM categories stored in the FAM/EAM container of UART's architecture, at the end of its training phase). The nodes in the category representation layer are connected to the nodes in the U_1^a layer as shown in Figure 5-5. Finally, the output layer (layer U_2^b) is the layer that produces the outputs of the network. Every node in the output layer of UART represents one of the labels of the pattern recognition task. The index k $(1 \le k \le N_b)$ designates a generic node in U_2^b; N_b represents the highest index needed to represent all the labels of the pattern classification task at hand.

The UART's performance steps are delineated (below) and then the genetic phase of UART is emphasized in more detail, as it was done for GFAM and GEAM.

5.1.1 Performance Phase of UART

This phase is similar to those of a FAM or an EAM. The process can be summarized in the following steps:

1. Present an input pattern (from the validation or test set) to the UART network
2. Calculate the CCF function, corresponding to this input pattern, for all the nodes in the U_2^a layer according to the following equations:

   a. For a FAM category: $T(j \mid \mathbf{I}) = \dfrac{M_a - s(\mathbf{w}_j^a) - dis(\mathbf{I}, \mathbf{w}_j^a)}{\beta_a + M_a - s(\mathbf{w}_j^a)}$ 5-1

   b. For an EAM category : $T(j \mid \mathbf{I}) = \dfrac{D - s(\mathbf{w}_j^a) - dis(\mathbf{I}, \mathbf{w}_j^a)}{\beta_a + D - s(\mathbf{w}_j^a)}$ 5-2

   Find the node J that has the maximum CCF

3. Check the label of this node J. This will be the predicted label of the UART network for this input pattern.
4. If more patterns are still in the list (validation or test set) present the next input pattern to the UART network. Otherwise, the performance phase is completed.

Through this sequence of four steps UART is able to produce predictions for all the input patterns of the validation set (during UART's genetic phase) or for all the input patterns of the test set (during UART's performance phase for the UART chosen as having the best fitness value at the last generation of the genetic phase).

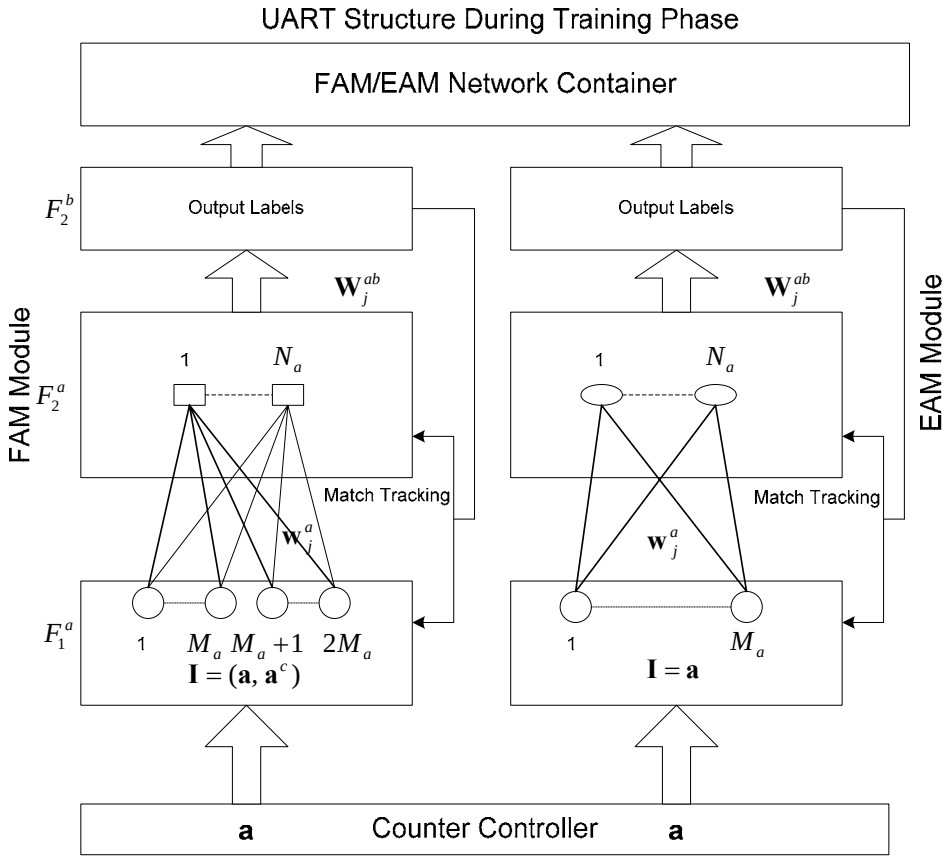

Figure 5-4: A simple UART structural diagram during the training phase

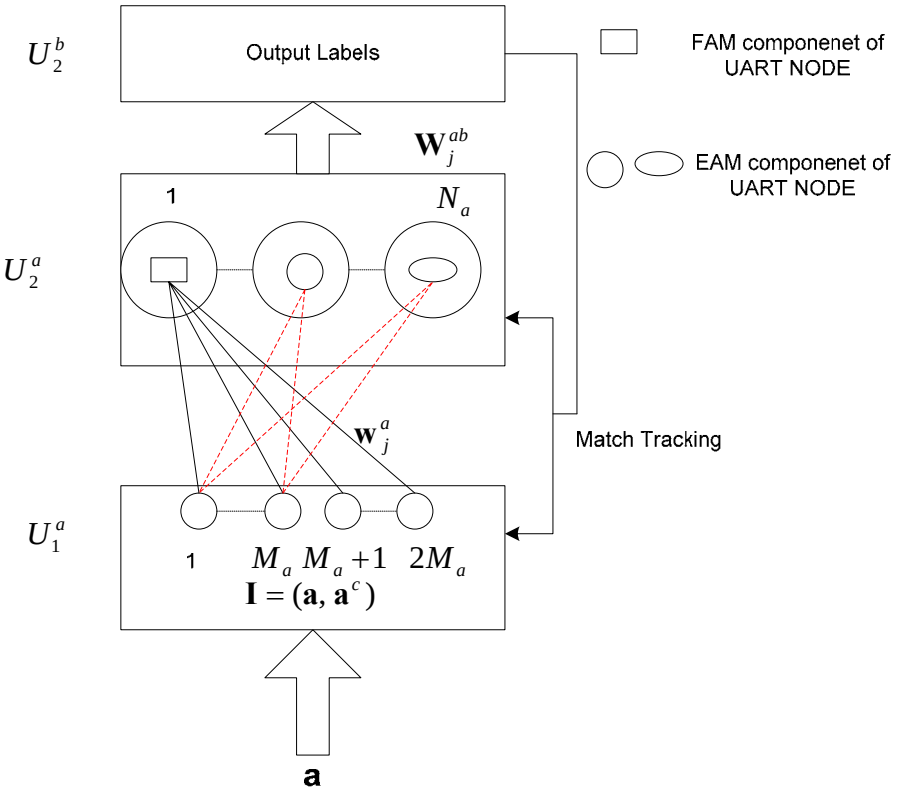

Figure 5-5: A simple UART structural diagram during the performance phase

5.1.2 Geometry Selection Phase (Genetic Phase) of UART

As it was the case for GFAM and GEAM we start with an initial population of Pop_{size} UARTs that we evolve. The GA parameters used for the evolution of FAM and EAM networks are also used for UART networks. This process follows the following steps:

Step 1: A chromosome in the initial population of UARTs contains FAM and EAM categories randomly chosen from the list of FAM and EAM categories, created during the training phase of UART and contained in the FAM/EAM container mixture module. At this stage all one-point categories are cropped.

Step 2: Evolve the chromosomes of the current generation by executing the following sub-steps:

Sub-Step 2a: Calculate fitness for all chromosomes of the current generation.

Sub-Step 2b: Initialize an empty generation (referred to as *temporary generation*).

Sub-Step 2c: Move the best (NC_{best}) chromosomes from the current generation to the temporary generation.

Sub-Step 2d: Select chromosomes for crossover from the current generation and thus further populate the temporary generation.

Sub-Step 2e: With a probability $P(Cat_{add})$ apply the Cat_{add} operator on every individual generated in sub-step 2d.

Sub-Step 4f: With a probability $P(Cat_{del})$ apply the Cat_{del} operator on every individual generated in sub-step 2e.

Sub-Step 4g: With a probability $P(Mut)$ apply the mutation operator on every individual generated in sub-step 2f.

Sub-Step 2h: Replace the current generation with the members of the temporary generation

Step 3: If evolution has reached the maximum number $Gen_{\max}$ of iterations, then calculate the performance of the best-Fitness UART network on the test set and report classification accuracy and number of categories that this Best-Fitness FAM network possesses. If the maximum number of iterations has not been reached yet, go to step 2 to evolve one more population of chromosomes

Each one of the aforementioned steps of the algorithm is now described in more detail, as needed.

Step 1 (More Details): We use a real number representation to encode the UART networks. Each UART chromosome consists of two levels, level 1 containing all the categories of the UART network, and level 2 containing the following components: two generic vectors $\mathbf{u}$ and $\mathbf{v}$, an integer *l* for the label, a double μ for the axis ratio, and a double r for the radius and an integer *t* for the type of the category. In the case of a FAM category, the generic vectors are equal to vectors $\mathbf{u}$ and $\mathbf{v}$ are used to represent the end points of the hyper-rectangle, *l*

represent the label, r and μ are not used and *t* is equal to 1, otherwise (i.e. an EAM category) the generic vectors are equal to center $\mathbf{m}$ and the direction $\mathbf{d}$, *l* represent the label, r encodes the radius, μ encodes the axis ratio and *t* is equal to 0. (see Figure 5-6). We denote the category of a trained UART network with index p $(1 \le p \le Pop_{size})$ by $\mathbf{w}_j^a(p)$, where $\mathbf{w}_j^a(p) = \mathbf{w}_j^{a,FAM}(p) = (\mathbf{u}_j(p), (\mathbf{v}_j(p)^c)$ or $\mathbf{w}_j^a(p) = \mathbf{w}_j^{a,EAM}(p) = (\mathbf{m}_j, \mathbf{d}_j, r_j)$ and the label of this category by $l_j(p)$ for $1 \le j \le N_a(p)$. In this step we are also eliminating single-point categories in the trained FAM networks, referred to as cropping the chromosomes, this is done by deleting FAM categories with size zero and EAM categories with radius equals to zero. Since our ultimate objective is to design a FAM network that reduces the network size and improves generalization we are discouraging at this stage the creation of single-point categories. We also randomly redistribute the categories amongst the chromosomes, this step is necessary to eliminate any advantages of a FAM network over an EAM network or vice versa. This is done by putting all the categories of all chromosomes in one group and then reassigning each chromosome a randomly chosen set of categories.

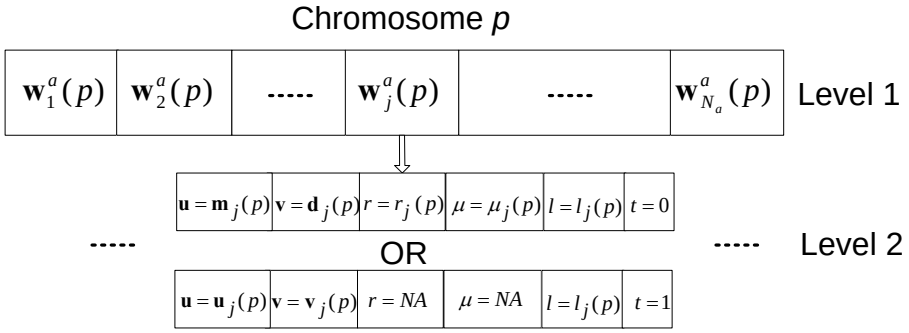

Figure 5-6: GFAM chromosome structure

Step 2 (More Details): In this step the UART applies the GA algorithm on the population of UARTs.

Sub-step 4a (More Details): Use the steps described in the performance phase below, calculate the fitness of each chromosome. This is accomplished by converting each chromosome into a UART network and then feeding into it the validation set and by

calculating the percentage of correct classification exhibited by each one of these UART networks. In particular, if $PCC(p)$ designates the percentage of correct classification, exhibited by the *p*-th UART, and this UART network possesses $N_a(p)$ nodes in its category representation layer, then its fitness function value is defined by:

$$Fit(p) = \frac{(Cat_{\max} - N_a(p)) \cdot PCC^2(p)}{\frac{100}{Cat_{\min}} - \frac{PCC(p)}{N_a(p)} + \varepsilon} \qquad 5\text{-}3$$

where, $Cat_{\min}$ and $Cat_{\max}$ are the minimum and maximum number of categories that a FAM network is allowed to have during the evolutionary process ($Cat_{\min}$ is chosen equal to 1, or equal to the number of classes in the classification problem under consideration, while $Cat_{\max}$ is chosen to be a relatively large number for the classification problem at hand). The constant ε in the denominator of the above equation is a small positive constant and it is needed to make sure that the denominator would not be zero in the case when $N_a(p) = Cat_{\min}$ and $PCC(p) = 100$.

Sub-step 4b (More Details): Obvious, no further explanations are needed.

Sub-step 4c (More Details): The algorithm searches for the best NC_{best} chromosomes from the current generation and copies them to the temporary generation.

Sub-step 4d (More Details): The remaining $Pop_{size} - NC_{best}$ chromosomes in the temporary generation are created by crossing over two parents from the current generation. The parents are chosen using the deterministic tournament selection method, as follows: Randomly select two groups of four chromosomes each from the current generation, and use as a parent from each group the chromosome with the best fitness value in the group. If it happens that from both groups the same chromosome is chosen then we choose from one of the groups the chromosome with the second best fitness value. If two parents with indices p, p' are crossed over two random numbers n, n' are generated from the index sets $\{1, 2, ..., N_a(p)\}$ and

$\{1, 2, ..., N_a(p')\}$, respectively. Then, all the categories with index greater than index n' in chromosome with index p' and all the categories with index less than index n in the category with index p are moved into an empty chromosome within the temporary generation. Notice that crossover is done on level 1 of the chromosome. This operation is pictorially illustrated in the Figure 5-7.

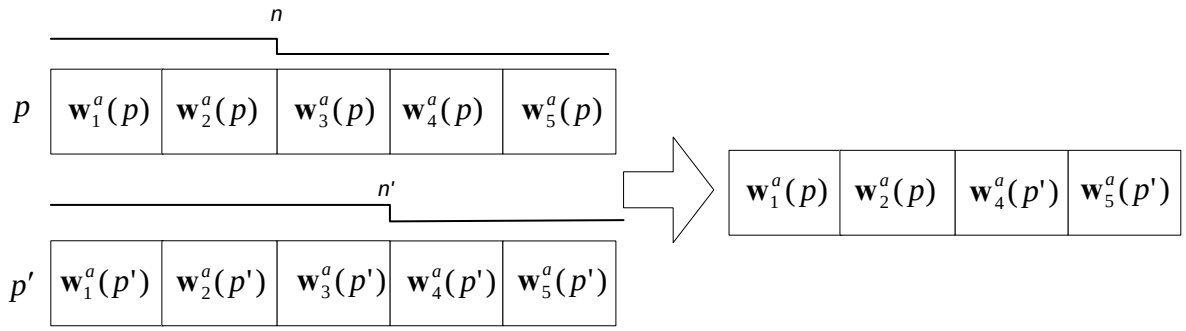

Figure 5-7: Crossover implementation

Sub-step 4e (More Details): The operator Cat_{add} adds a new category to every chromosome created in step 4d with probability $P(Cat_{add})$. With a 0.5 probability the new category is chosen to be a FAM or an EAM category. If the chosen category is a FAM category, the category gets lower and upper endpoints $\mathbf{u}$, $\mathbf{v}$ that are randomly generated as follows: For every dimension of the input feature space (M_a dimensions total) we generate two random numbers uniformly distributed in the interval [0, 1]; the largest of these numbers is associated with the $\mathbf{v}$ coordinate along this dimension, while the smallest of the two random numbers is associated with the $\mathbf{u}$ coordinate along this dimension. If the chosen category is an EAM category then the category has a center $\mathbf{m}$, a direction vector $\mathbf{d}$, a radius r, and an axis ration μ, that are randomly generated as follows: For every dimension of the input feature space (M_a dimensions total) we generate a random number uniformly distributed in the interval [0, 1]; we assign these values to the individual components of $\mathbf{m}$, the direction vector $\mathbf{d}$ is assigned zeros, the axis ratio μ and the radius r are given a positive real random

numbers in the interval [0,1]. The label of this newly created category is chosen randomly amongst the N_b categories of the pattern classification task under consideration. A chromosome does not add a category if the addition of this category results in number of categories for this chromosome that exceeds the designated maximum number of categories $Cat_{\max}$.

Sub-step 4f (More Details): The operator Cat_{del} deletes one of the categories of every chromosome created in step 4e with probability $P(Cat_{del})$. A chromosome does not delete a category if the deletion of this category results in the number of categories for this chromosome to fall below the designated minimum number of categories $Cat_{\min}$.

Sub-Step 4g (More Details): In UART, every chromosome created by step 4f gets mutated as follows: with probability $P(mut)$ every category is mutated. If a FAM category is chosen, its **u** or **v** endpoints is selected randomly (50% probability), and then every component of this selected vector gets mutated by adding to it a small number. This number is drawn from a Gaussian distribution with mean 0 and standard deviation 0.01. If the component of the chosen vector becomes smaller than 0 or greater than 1 (after mutation), it is set back to 0 or 1 respectively. If an EAM category is chosen its center **m** or direction **d** is selected randomly (50% probability), then every component of this selected vector gets mutated by adding to it a small number. This number is drawn from a Gaussian distribution with mean 0 and standard deviation 0.01. If the component of the chosen vector becomes smaller than 0 or greater than 1 (after mutation), it is set back to 0 or 1, respectively. Furthermore, its axis ratio μ or radius r is selected (50% probability), we also add a small number drawn from a Gaussian distribution to the selected item with the same rules as above. Notice that mutation is applied on level 2 of the chromosome structure, but the label of the chromosome is not mutated (the reason being that our initial GA population consists of trained FAMs and

EAMs, and consequently we have a lot of confidence in the labels of the categories that these trained EAMs have discovered through the EAM training process).

Step 5 (More Details): Obvious, no more details are needed.

5.2 Results of UART

We used the same default set of parameters used for GFAM to run all the experiments of UART with one modification, and the results were excellent. Hence, in UART's case we avoided the experimentation (applied to GFAM) to choose good default values for the GA parameters. Hence, UART is produced by first initializing a population of 20 trained FAM and EAM networks (they were trained with different values of the baseline vigilance parameter and different orders of training pattern presentations), and by evolving them for 500 generations. In particular, the GA parameters used for the creation of UART were: $\rho_a^{\min}$ = 0.1, $\rho_a^{\max}$ = 0.75, β_a =0.1, Pop_{size} = 20, $Gen_{\max}$ = 500, NC_{best} = 3, $Cat_{\min}$ = 1, $Cat_{\max}$ = 300, $P(Cat_{add})$ =0.1, $P(Cat_{del})$ =0.1, $P(mut)$ = 1/*Na*. UART is the network that attains the highest value of the fitness function at generation 500 of the evolutionary process.

5.2.1 UART Performance

In this section we are reporting the performance of UART on each one of the database problems, described in Section 3.1.2.1. Similar to that of GFAM, the performance of UART is assessed by reporting the size and the accuracy attained by UART on the test set. The results are reported in Table 5-1. In Table 5-1, the first column is the index of the database that we are experimenting with. The second column is the actual database name, as reported in section 3.1.2. and summarized in Table 3-3. Columns 3 and 4 contain the size and accuracy of the UART network for the designated database. The performance of UART, as it is evidenced by the results in Table 5-1, is verified by some obvious observations. For instance, UART's performance on databases 1-12 (Gaussian datasets of known amount of overlap) is

nearly optimal; for example the best performance on the G6c-40 problem (6 class Gaussian dataset of 40% overlap) is a classifier with 6 categories and 60% correct classification, and UART is a classifier with 6 categories and 59.83% of correct classification. Similarly, in the CINS problem the optimal classifier would require 2 categories and attain a 100% correct classification; UART is a 2 category classifier exhibiting a 99.37% of correct classification. Finally, all of the real problems reported here, MOD-IRIS, ABALONE and PAGE, also gave very good results 94.88%, 64.1% and 95.73% of correct classification respectively, by creating 2,3 and 3 categories respectively only.

5.2.2 Performance Comparisons of UART and other ART Networks

As it was the case with GFAM, we compared UART's performance with the performance of the following networks: ssEAM, ssEAM, ssGAM, and safe micro-ARTMAP.

The comparisons of UART and the aforementioned ART networks are depicted in Table 5-1, where the first column is the index of the database that we are experimenting with. The second column is the actual database name, as reported in earlier sections. Columns 3-10 of Table 5-1 contain the performance of the designated ART networks. The performance reported includes the accuracy of the "best" ART network on the test set. The performance also includes the number of categories created of the designated ART network. The reported numbers of accuracy and size of the "best" network correspond to the ART network that attained the highest value of the fitness function (this value was computed based on the accuracy of the ART network on the cross-validation set, and on the size of the ART network). Note, that for networks, other than UART, the "best" ART network was determined after extensive experimentation with the ART network's parameter values (e.g., in ssFAM the best network was determined after training ssFAM networks with different values of the choice parameter, vigilance parameter, order of pattern presentation, and

amount of mixture of labels allowed within a category; a total of 20,000 ssFAM networks were trained and their performance was examined). On the other hand, the performance of the UART is the one calculated after the evolution of 20 trained FAM and EAM networks for 500 generations with GA parameters as indicated in Section 5.4.

According to the results in Table 5-1, in all instances (except minor exceptions) the accuracy of GGAM (generalization performance) is higher than the accuracy of the other ART network (where ART is ssEAM, ssEAM, ssGAM or safe micro-ARTMAP). According to the results in Table 5-1, in all instances (with no exceptions) the size of GGAM is smaller than the size of the other ART networks (where ART is ssEAM, ssEAM, ssGAM or safe micro-ARTMAP), sometimes even by a factor of 15. For example, the generalization performance of UART can be as 12% better than the generalization performance of ssFAM, while its size can be by a factor of 4 times smaller than the size of ssFAM. Also, the generalization performance of UART can be as 14% better than the generalization performance of ssEAM, while its size can be by a factor of 4 times smaller than the size of ssEAM. Furthermore, the generalization performance of UART can be as 9% better than the generalization performance of ssGAM, while its size can be by a factor of 17 times smaller than the size of ssGAM. Finally, the generalization performance of UART can be as 9% better than the generalization performance of safe micro-ARTMAP, while its size can be by a factor of 4 times smaller than the size of safe micro-ARTMAP.

The comparison results between UART and the other ART networks are also pictorially depicted in figures 5-8a to 5-8d. In each one of these figures we are showing the accuracy of UART, and that of one other network (e.g., ssEAM). In the same figure we are also showing the size of the UART and that of one other ART network. This way the one-to-one comparison of the UART and the other ART networks can be quickly assessed.

What is most worth pointing out is that the better performance of UART is attained with reduced computations compared with the computations needed by the alternate methods (ssEAM, ssEAM, ssGAM, safe micro-ARTMAP). Specifically, the performance attained by ssEAM, ssEAM, ssGAM and the safe micro-ARTMAP required training these networks for a large number of network parameter settings (at least 20,000 experiments) and then choosing the network that achieved the higher value for the fitness function that we introduced earlier in Section 3.2.2. Of course, one can argue that such an extensive experimentation with these networks might not be needed, especially if one is familiar with the functionality of these networks and chooses to experiment only with a limited set of network parameter values. However, the practitioner in the field might lack the expertise to carefully choose the network parameters to experiment with, and consequently might need to experiment extensively to come up with a good network. The computational complexity of UART is given by similar equations as the GFAM computational complexity, calculated in Appendix B. The comparison between the computational complexity GFAM (and UART) and the rest of the ART networks is based under the assumption that extensive parameter experimentation with the network parameters of ssEAM, ssEAM, ssGAM or safe micro-ARTMAP is needed to obtain a good performing ssEAM, ssEAM, ssGAM or safe micro-ARTMAP network, respectively.

Table 5-1: UART performance and size compared to other ART architectures

	Database Name	UART		Safe μAM		ssFAM		ssEAM		ssGAM	
1	G2c-05	95.2	2	95.22	2	94.90	2	94.94	2	94.48	4
2	G2c-15	85.22	2	85.00	2	84.80	3	85.20	2	85.04	2
3	G2c-25	75.16	2	74.98	2	74.60	2	74.50	2	75.10	2
4	G2c-40	61.24	2	61.40	3	61.34	3	60.98	2	61.30	3
5	G4c-05	94.9	4	95.04	4	94.10	7	94.14	4	94.80	4
6	G4c-15	84.6	4	83.28	4	81.40	11	83.20	4	84.24	9
7	G4c-25	75.18	4	74.50	4	70.80	9	72.72	4	72.32	21
8	G4c-40	59.96	4	59.76	5	58.48	14	55.62	13	59.10	14
9	G6c-05	94.7	6	93.57	9	91.42	11	93.80	7	94.40	8
10	G6c-15	84.85	6	80.92	6	81.11	7	81.80	6	84.35	13
11	G6c-25	73.90	6	70.74	13	69.62	15	71.10	7	72.86	20
12	G6c-40	59.83	6	58.03	11	56.35	17	54.21	17	55.65	13
13	4Ci/Sq	98.16	7	95.42	8	87.23	18	94.68	5	93.4	12
14	4Sq/Sq	97.9	8	99.12	9	97.24	13	88.89	5	91.78	16
15	7Sq	98.5	7	97.22	16	97.26	16	88.5	19	95.83	93
16	1Ci/Sq	99.37	2	94.76	8	92.97	8	97.02	8	91.02	8
17	1Ci/Sq/0.3:0.7	99.3	2	96.82	8	93.21	8	97.13	8	92.33	8
18	5Ci/Sq	88.87	30	83.83	52	81.95	52	78.68	87	90.02	111
19	2Ci/Sq/20:30:50	99.2	3	97.22	6	90.24	12	97.01	3	95.6	9
20	7SqWN	89.3	7	86.67	20	80.15	24	75.23	32	83.11	123
21	5Ci/SqWN	80.67	30	71.72	52	68.39	57	69.2	136	81.3	145
22	MOD-IRIS	94.88	2	94.92	2	93.41	8	94.54	2	94.54	2
23	ABALONE	64.1	3	57.18	4	59.52	6	56.80	7	55.10	3
24	PAGE	95.73	3	88.82	6	90.63	3	89.54	3	89.34	5

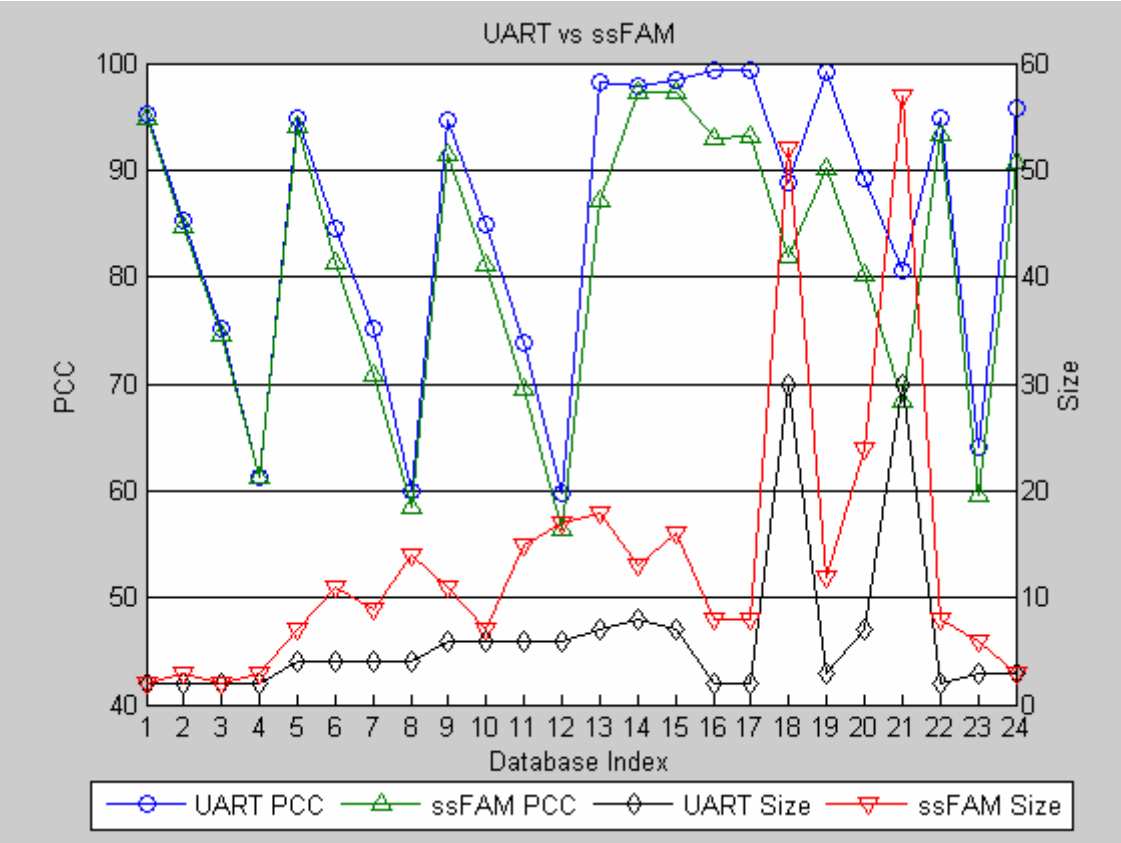

Figure 5-8a: Performance and Size comparison of UART vs ssFAM

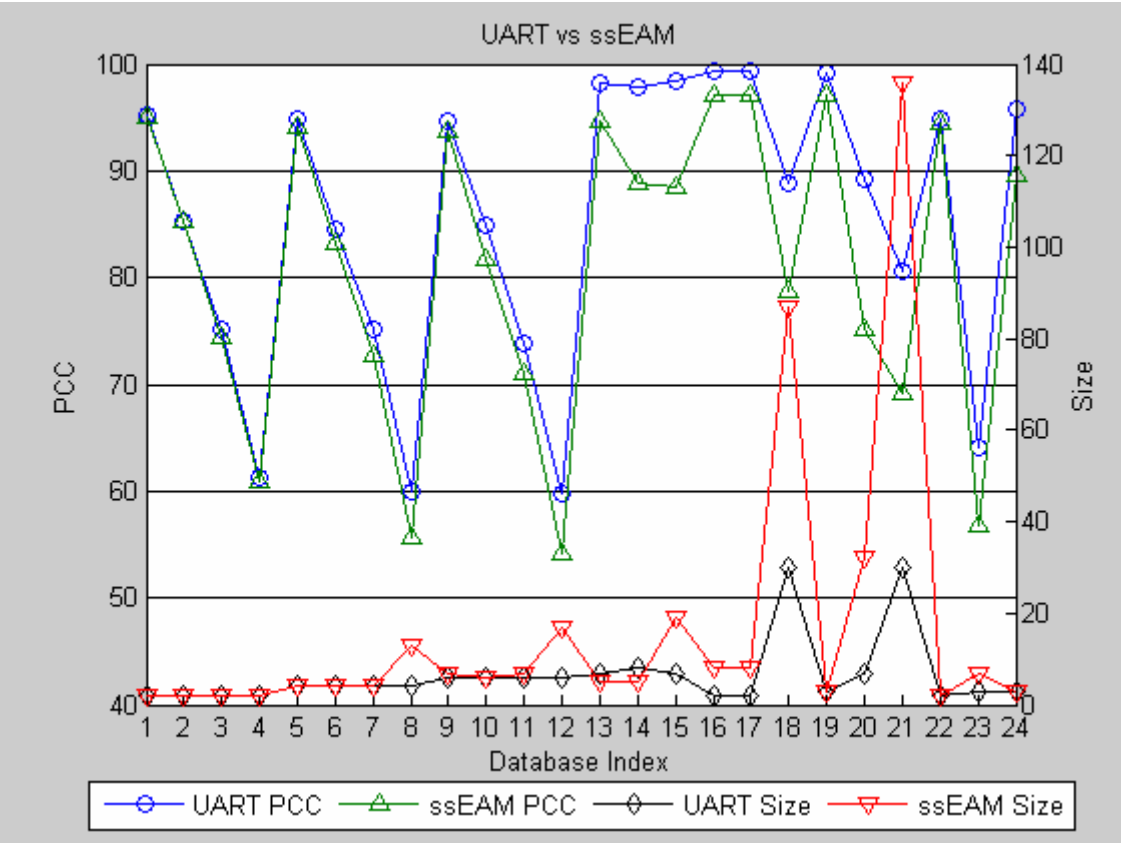

Figure 5-8b: Performance and Size comparison of UART vs ssEAM

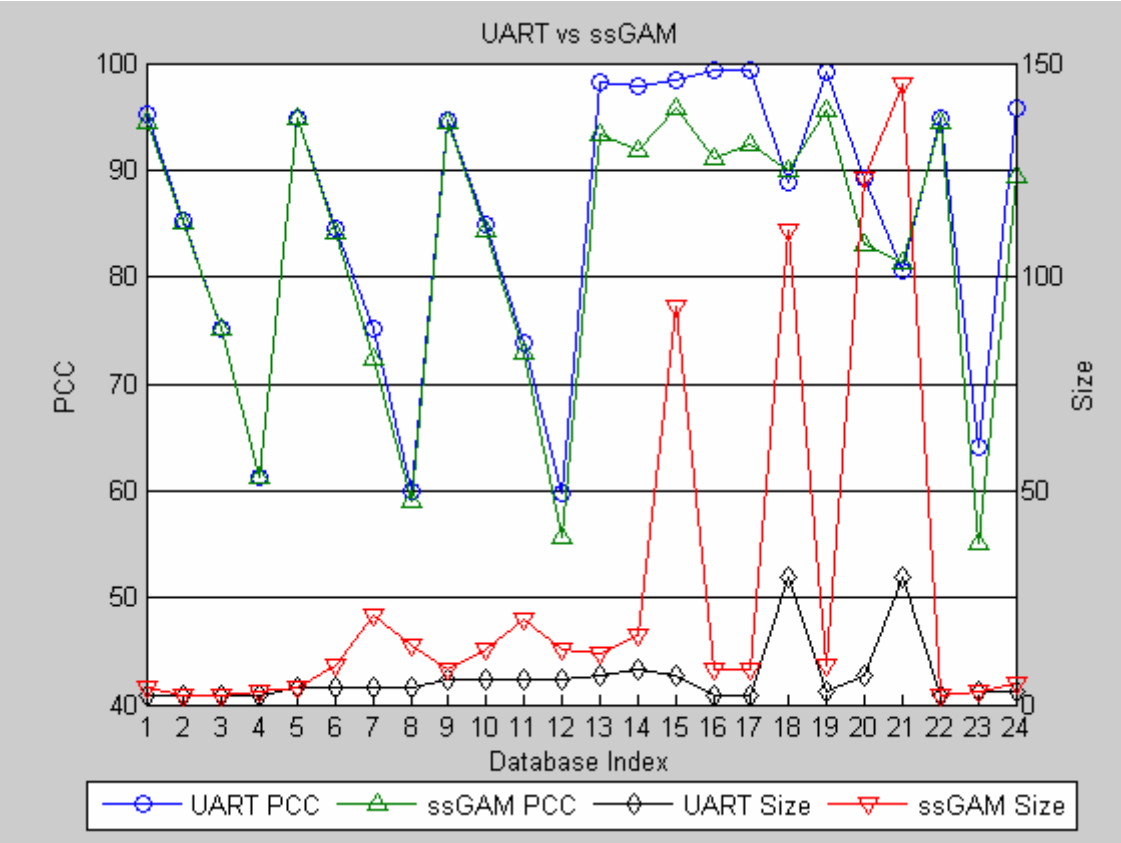

Figure 5-8c: Performance and Size comparison of UART vs ssGAM

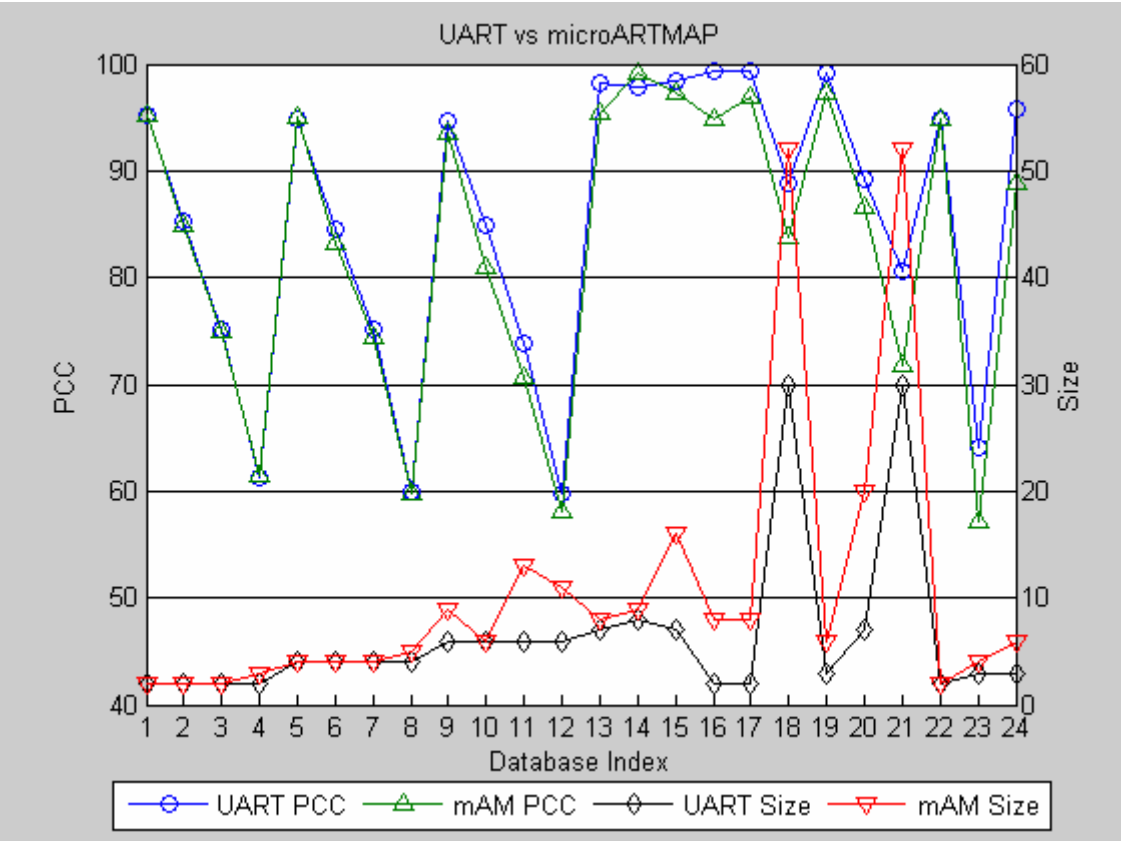

Figure 5-8d: Performance and Size comparison of UART vs microARTMAP

5.2.3 Performance Comparisons of UART and other Genetic ART Networks

In this section we are comparing the size and accuracy of UART with the other three genetic ART modules namely: GFAM, GEAM and GGAM. Table 5-2 shows the results of all of the genetic modules on all the databases presented in section 3.1.2.1, (a one to one comparison figures are also dippected in figures 5-9a to 5-9c). It is clear that we used the same method to choose the best genetic ART in all of the genetic modules. From a quick look at table 5-2, it is clear that in the Gaussian databases all of the genetic ART modules performed very well with minor differences. On the structure within structure databases however, the performance was different. Table 5-2 shows that GFAM gave better accuracy on databases 14, 15, 18 and 21 while GEAM and GGAM gave better accuracy on databases 13, 16, 17, 19 and 20. If we investigate further, we find that GFAM gave better accuracy when the problem didn't have a circle in it, on the other hand, GEAM and GGAM gave better results on those problems that have circles in them, *this was the reason behind the development of UART*.

UART performance compares very well to the best of the three on all problems, for example, UART gave 99.37% and 99.3% for databases 16 and 17 respectively with only 2 categories in each case, this performance is very close to that of GEAM and GGAM on the same databases of 99.99% with 2 categories, on the other hand, UART gave 98.5% accuracy on database 15 with only 7 categories, while GFAM gave 97.2% with 7 categories. the bottom line is UART performed very well all the databases described in 3.1.2.1.

Table 5-2: UART performance and size compared to other genetic ART architectures

	Database Name	UART		GFAM		GEAM		GGAM	
1	G2c-05	95.2	2	95.36	2	95	2	95.2	2
2	G2c-15	85.22	2	85.30	2	85.12	2	85.2	2
3	G2c-25	75.16	2	75.08	2	74.74	2	75.06	2
4	G2c-40	61.24	2	61.38	2	61.6	2	61.64	2
5	G4c-05	94.9	4	95.02	4	94.8	4	94.82	4
6	G4c-15	84.6	4	84.46	4	84.22	4	84.28	4
7	G4c-25	75.18	4	75.20	4	73.7	4	74.74	4
8	G4c-40	59.96	4	60.60	4	59.5	4	59.82	4
9	G6c-05	94.7	6	94.68	6	94.12	6	94.46	6
10	G6c-15	84.85	6	84.71	6	84.03	6	84.81	6
11	G6c-25	73.90	6	73.90	6	73.04	6	74.00	6
12	G6c-40	59.83	6	59.19	6	59.35	6	59.43	6
13	4Ci/Sq	98.16	7	96.32	8	97.4	7	97.2	5
14	4Sq/Sq	97.9	8	97.12	9	92.76	6	93.06	5
15	7Sq	98.5	7	97.2	7	93.46	13	95.13	10
16	1Ci/Sq	99.37	2	97.2	8	99.9	2	99.76	2
17	1Ci/Sq/0.3:0.7	99.3	2	97.8	8	99.9	2	99.9	2
18	5Ci/Sq	88.87	30	92	50	83.03	13	87.76	39
19	2Ci/Sq/20:30:50	99.2	3	97.87	3	96.86	3	98.93	3
20	7SqWN	89.3	7	87.3	7	90.82	10	89.83	10
21	5Ci/SqWN	80.67	30	81.97	50	81	50	81.34	42
22	MOD-IRIS	94.88	2	95.31	2	94.81	2	94.83	2
23	ABALONE	64.1	3	58.73	2	61.00	3	61.53	3
24	PAGE	95.73	3	95.59	3	94.12	3	95.01	2

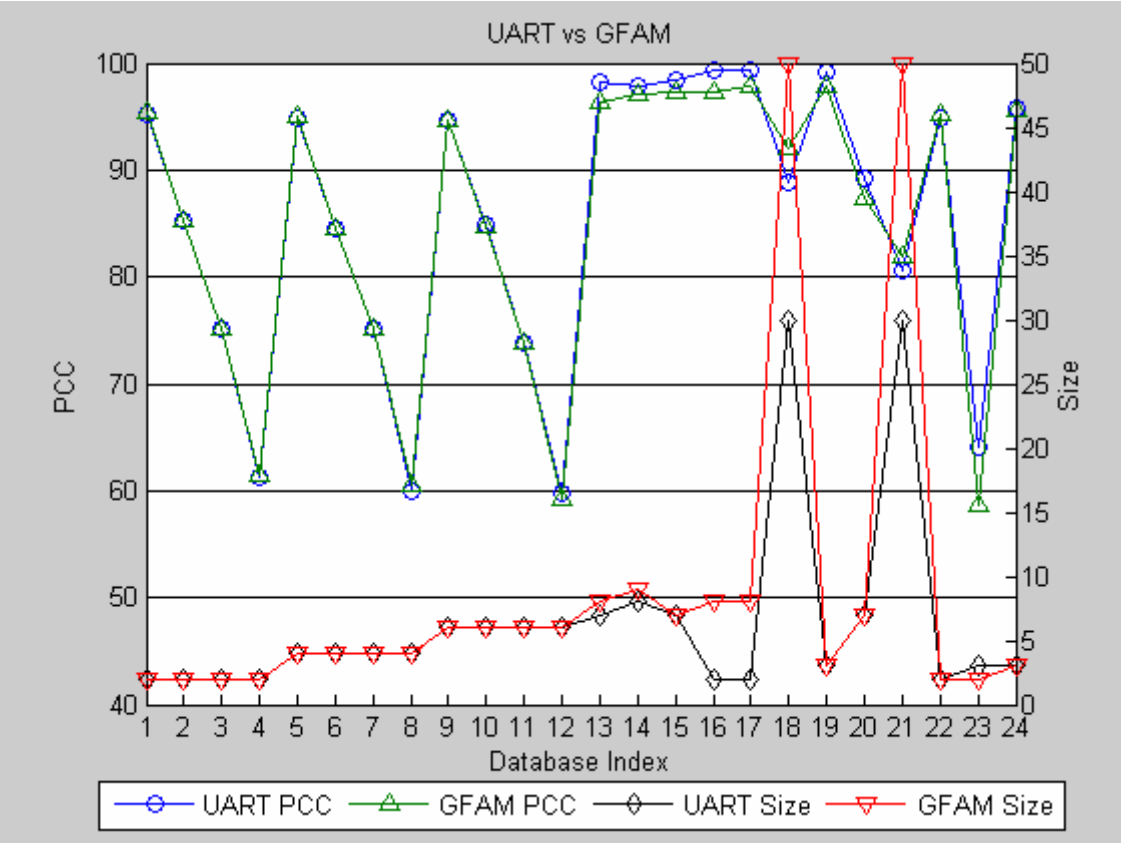

Figure 5-9a: Performance and Size comparison of UART vs GFAM

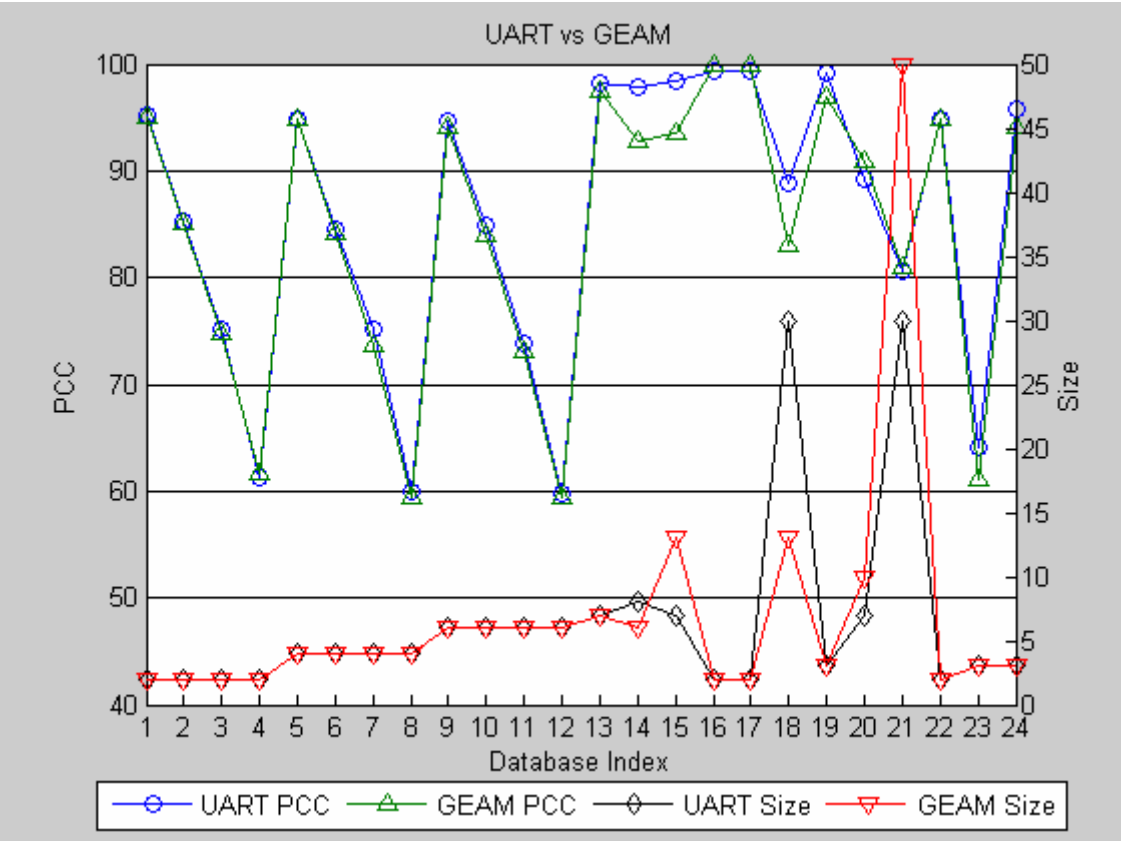

Figure 5-9b: Performance and Size comparison of UART vs GEAM

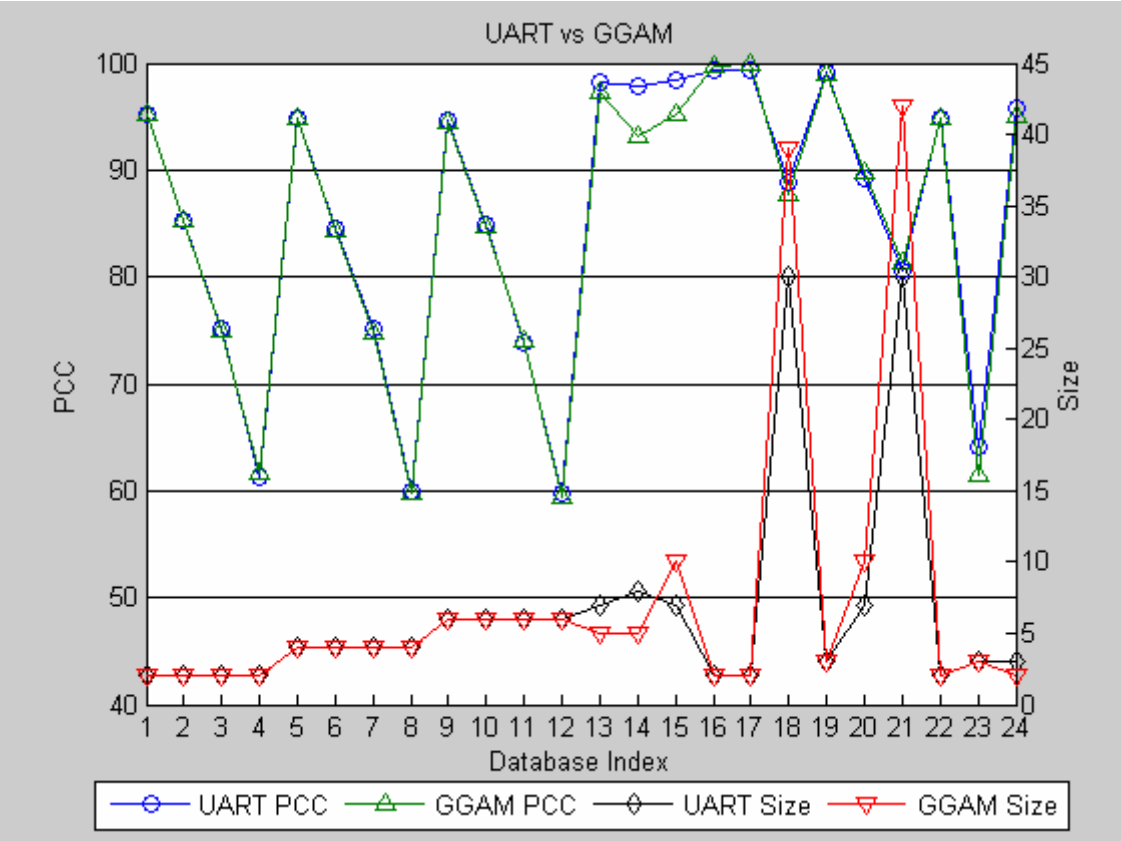

Figure 5-9c: Performance and Size comparison of UART vs GGAM

5.3 UART Summary

UART is a novel approach of mixing different types of ART categories to obtain better coverage of the input space. UART used two types of categories, namely: FAM categories and EAM categories to enhance its performance.

UART used genetic algorithms to solve the category proliferation problem in ART. This method relies on evolving a population of trained ART networks, and more specifically Ellipsoidal ARTMAP (EAM) and Fuzzy ARTMAP (FAM) neural networks. The evolution of trained FAM's and EAM's creates an ART network, referred to as UART.

In chapter 3 we defined a methodology of evolving trained FAM networks, resulting in GFAM. This methodology was also applied successfully for the evolution of GAM networks, resulting in UART. In chapter 3 we experimented with a number of databases that

helped us identify good default parameter settings for the evolution of FAM The same parameters and settings used in chapter 3 for the evolution of FAM networks (GFAM) were also used for the evolution of FAM and EAM networks (UART).

Our experiments with UART indicate that UART is superior to a number of other ART techniques (ssEAM, ssEAM, ssGAM, safe micro-ARTMAP) that have been introduced into the literature to address the category proliferation problem in ART. More specifically, UART gave a better generalization performance (in almost all problems tested) and a smaller size network (in all problems tested), compared to these other ART techniques. What is also worth mentioning is that UART outperformed these other ART techniques by requiring only a fraction of the computations needed by these other networks.

Based on a simple comparison, UART also outperformed the other genetic ART modules we introduced in chapter 3 and 4 on many databases and gave almost as good results on the rest of the databases.

6. ANALYSIS

In UART we have two types of categories that are competing for the attention of an input pattern. In Georgiopoulos, et al., a very comprehensive analysis was presented that explained the order according to which categories are chosen in Fuzzy ARTMAP. We repeat the analysis here for the case when the category in the category representation layer of UART is either an ellipsoid (EAM category) or a rectangle (FAM category). Actually, to simplify the analysis we assume that the ellipsoid is a circle (special case of an ellipsoid).

We also compare the computational complexity associated with GFAM and the computational complexity associated with the other ART algorithms that we have experimented with in the previous sections. This comparisons allows one to see analytically that the superior performance of GFAM is attained by reduced computations compared with the computations needed for the other ART architectures.

6.1 UART Order of Search Analysis

During the training phase of UART, the order according to which categories are accessed in UART depends on the type of categories considered, the closeness of an input pattern to the categories, the size of the categories, and the value of the choice parameter β . Before we present a number of theorems that explain the order according to which categories are accessed we present a few assumptions and definitions, included below.

Assumptions:

- Only circles are considered here which are a special case of an ellipsoid.
- The D parameter of EAM is equal to M (dimensionality of the input pattens).

Definitions

- The size of a FAM category R^a is size(R^a) = $|R^a| = |\mathbf{v}^a - \mathbf{u}^a|$. 6-1
- The size of an EAM category E^a is size (E^a) = $2r$. 6-2

  where r is the radius of E^a

Theorem 1: if an input pattern $\mathbf{I}$ is presented to a FAM category R^a and an EAM category E^a and $\mathbf{I}$ lies inside both categories, then $\mathbf{I}$ will be represented by E^a iff size(E^a) < size (R^a) and will be represented by R^a iff size(E^a) > size (R^a) (i.e. the category with the smaller size).

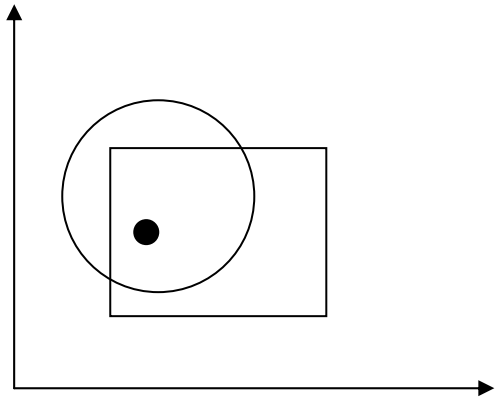

Figure 6-1: Case # 1, a pattern inside both a FAM category and an EAM category

Theorem 1 Proof: Let us consider the first case where $\mathbf{I}$ will be represented by E^a iff size(E^a) < size (R^a), this implies that $2r < |R^a|$

Let's start by calculating the CCF function of both categories where

$$T_E^a = \frac{D - r - \max\{r, \|\mathbf{I} - \mathbf{m}\|_c\}}{D - 2r + \beta}, \qquad 6\text{-}3$$

where $\|\mathbf{I} - \mathbf{m}\|_c$ here is the distance between two points (since we are dealing with circles) (i.e. $\|\mathbf{I} - \mathbf{m}\|_c = \frac{1}{\mu}\sqrt{\|\mathbf{I} - \mathbf{m}\|^2}$), $\mu = 1$ and β is a small positive number.

And

$$T_F^a = \frac{|\mathbf{I} \wedge \mathbf{w}^{a,old}|}{|\mathbf{w}^{a,old}| + \beta} \qquad 6\text{-}4$$

Since the pattern is inside both categories then $\max\{r, \|\mathbf{I} - \mathbf{m}\|_c\} = r$

and $|\mathbf{I} \wedge \mathbf{w}^{a,old}| = |\mathbf{w}^{a,old}|$

And so

$$T_E^a = \frac{D - r - \max\{r, \| \mathbf{I} - \mathbf{m} \|_c\}}{D - 2r + \beta} = \frac{M - 2r}{M - 2r + \beta} \qquad 6\text{-}5$$

And

$$T_F^a = \frac{| \mathbf{I} \wedge \mathbf{w}^{a,old} |}{| \mathbf{w}^{a,old} | + \beta} = \frac{| \mathbf{w}^{a,old} |}{| \mathbf{w}^{a,old} | + \beta} = \frac{M - | R^a |}{M - | R^a | + \beta} \qquad 6\text{-}7$$

If E^a is to be chosen before R^a, then T_E^a should be greater than T_F^a. But,

$\frac{M - 2r}{M - 2r + \beta} > \frac{M - | R^a |}{M - | R^a | + \beta}$ because $M - 2r > M - | R^a |$ and the function

$$\frac{x}{x + \beta} \qquad 6\text{-}8$$

is an increasing function of its argument *x;* hence the result is proven.

The proof of the reverse scenario is straight forward, and it is omitted.

Theorem 2: if an input pattern $\mathbf{I}$ is presented to a FAM category R^a and an EAM category E^a, and $\mathbf{I}$ lies inside R^a but outside E^a, then $\mathbf{I}$ will be represented by E^a iff $d_E < g(\beta)$, where $d_E = dis(\mathbf{I}, E^a)$, $g(\beta) = b - \frac{a(b + \beta)}{a + \beta}$, $a = M - | R^a |$, and $b = M - 2r$.

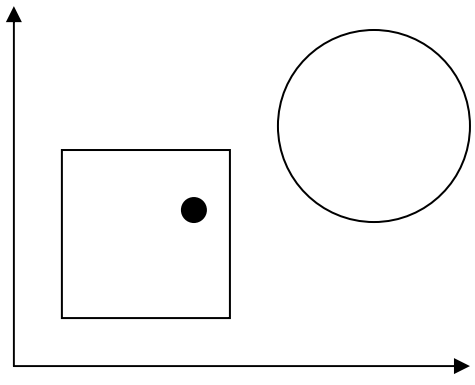

Figure 6-2: Case # 2, a pattern inside a FAM category but outside an EAM category

Theorem 2 Proof: Let's start by calculating the CCF function of both categories as in 6-3 and 6-4 respectively

$$T_E^a = \frac{D - r - \max\{r, \| \mathbf{I} - \mathbf{m} \|_c\}}{D - 2r + \beta}$$

And

$$T_F^a = \frac{| \mathbf{I} \wedge \mathbf{w}^{a,old} |}{| \mathbf{w}^{a,old} | + \beta}.$$

Since the pattern is inside R^a and outside E^a, then $\max\{r, \| \mathbf{I} - \mathbf{m} \|_c\} = \| \mathbf{I} - \mathbf{m} \|_c$, and

$| \mathbf{I} \wedge \mathbf{w}^{a,old} | = | \mathbf{w}^{a,old} |$

Consequently,

$$T_E^a = \frac{D - r - \| \mathbf{I} - \mathbf{m} \|_c}{D - 2r + \beta} = \frac{D - r - \sqrt{\| \mathbf{I} - \mathbf{m} \|^2}}{D - 2r + \beta} = \frac{M - 2r - d_E}{M - 2r + \beta} \qquad 6\text{-}9$$

And

$$T_F^a = \frac{| \mathbf{I} \wedge \mathbf{w}^{a,old} |}{| \mathbf{w}^{a,old} | + \beta} = \frac{| \mathbf{w}^{a,old} |}{| \mathbf{w}^{a,old} | + \beta} = \frac{M - | R^a |}{M - | R^a | + \beta} \qquad 6\text{-}10$$

By substituting a and b with their corresponding equals, in the above equations we get

$$T_E^a = \frac{b - d_E}{b + \beta} \qquad 6\text{-}11$$

and

$$T_F^a = \frac{a}{a + \beta} \qquad 6\text{-}12$$

If E^a is to represent the input pattern then T_E^a should be greater than T_F^a, hence,

$$\frac{b - d_E}{b + \beta} > \frac{a}{a + \beta} \qquad 6\text{-}13$$

which implies that

$$d_E < b - \frac{a(b + \beta)}{a + \beta} \qquad 6\text{-}14$$

Theorem 3: if an input pattern $\mathbf{I}$ is presented to a FAM category R^a and an EAM category E^a, and $\mathbf{I}$ lies inside E^a but outside R^a, then $\mathbf{I}$ will be represented by R^a iff $d_F < k(\beta)$, where $d_F = dis(\mathbf{I}, R^a)$, $k(\beta) = a - \frac{b(a+\beta)}{b+\beta}$, $a = M - |R^a|$, and $b = M - 2r$.

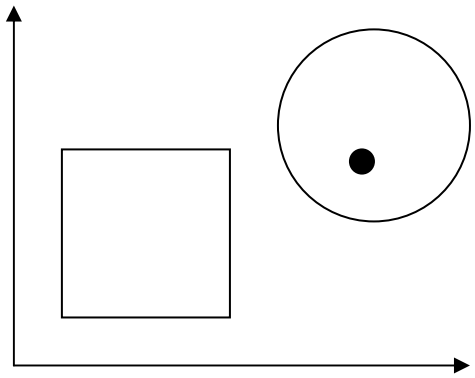

Figure 6-3: Case # 3, a pattern inside an EAM category but outside a FAM category

Theorem 3 Proof:

Let's start by calculating the CCF function of both categories in 6-3 and 6-4 respectively

$$T_E^a = \frac{D - r - \max\{r, \| \mathbf{I} - \mathbf{m} \|_c\}}{D - 2r + \beta}$$

And

$$T_F^a = \frac{| \mathbf{I} \wedge \mathbf{w}^{a,old} |}{| \mathbf{w}^{a,old} | + \beta}.$$

Since the pattern is inside E^a then $\max\{r, \| \mathbf{I} - \mathbf{m} \|_c\} = r$. Furthermore, since the pattern is outside R^a,

$$| \mathbf{I} \wedge \mathbf{w}^{a,old} | = M - | R^{a,new} | = M - | R^a | - d_F \qquad 6\text{-}15$$

Consequently,

$$T_E^a = \frac{D - r - \| \mathbf{I} - \mathbf{m} \|_c}{D - 2r + \beta} = \frac{M - 2r}{M - 2r + \beta} \qquad 6\text{-}16$$

and

$$T_F^a = \frac{| \mathbf{I} \wedge \mathbf{w}^{a,old} |}{| \mathbf{w}^{a,old} | + \beta} = \frac{| \mathbf{w}^{a,old} |}{| \mathbf{w}^{a,old} | + \beta} = \frac{M - | R^a | - d_F}{M - | R^a | + \beta} \qquad 6\text{-}17$$

By substituting a and b for their values in the 6-16 and 6-17 we get

$$T_E^a = \frac{b}{b + \beta} \qquad 6\text{-}18$$

and

$$T_F^a = \frac{a - d_F}{a + \beta} \qquad 6\text{-}19$$

If R^a is to represent the pattern then T_F^a should be greater than T_E^a, hence,

$\frac{a - d_F}{a + \beta} > \frac{b}{b + \beta}$ which implies that

$$d_F < a - \frac{b(a + \beta)}{b + \beta} \qquad 6\text{-}20$$

Theorem 4: if an input pattern $\mathbf{I}$ is presented to a FAM category R^a and an EAM category E^a, and $\mathbf{I}$ lies outside both R^a and E^a, then $\mathbf{I}$ will be represented by E^a iff $d_E < h(\beta)$, and will be represented by R^a iff $d_F < l(\beta)$, where $d_E = dis(\mathbf{I}, E^a)$, $d_F = dis(\mathbf{I}, R^a)$, $h(\beta) = b - \frac{(a - d_F)(b + \beta)}{a + \beta}$, $l(\beta) = a - \frac{(b - d_E)(a + \beta)}{b + \beta}$, $a = M - | R^a |$, and $b = M - 2r$.

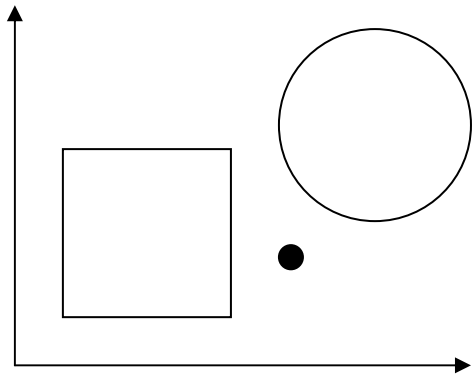

Figure 6-4: Case # 4, a pattern outside both FAM and EAM categories

Theorem 4 Proof:

Let's start by calculating the CCF function of both categories in 6-3 and 6-4 respectively

$$T_E^a = \frac{D - r - \max\{r, \| \mathbf{I} - \mathbf{m} \|_c\}}{D - 2r + \beta}$$

and

$$T_F^a = \frac{| \mathbf{I} \wedge \mathbf{w}^{a,old} |}{| \mathbf{w}^{a,old} | + \beta}.$$

Since the pattern is outside both categories, then $\max\{r, \| \mathbf{I} - \mathbf{m} \|_c\} = \| \mathbf{I} - \mathbf{m} \|_c$, and

$$| \mathbf{I} \wedge \mathbf{w}^{a,old} | = M - | R^{a,new} | = M - | R^a | - d_F$$

Consequently,

$$T_E^a = \frac{D - r - \| \mathbf{I} - \mathbf{m} \|_c}{D - 2r + \beta} = \frac{D - r - \sqrt{\| \mathbf{I} - \mathbf{m} \|^2}}{D - 2r + \beta} = \frac{M - 2r - d_E}{M - 2r + \beta} \qquad 6\text{-}21$$

And

$$T_F^a = \frac{| \mathbf{I} \wedge \mathbf{w}^{a,old} |}{| \mathbf{w}^{a,old} | + \beta} = \frac{| \mathbf{w}^{a,old} |}{| \mathbf{w}^{a,old} | + \beta} = \frac{M - | R^a | - d_F}{M - | R^a | + \beta} \qquad 6\text{-}22$$

By substituting a and b with their values in 6-21 and 6-22 we get

$$T_E^a = \frac{b - d_E}{b + \beta} \qquad 6\text{-}23$$

and

$$T_F^a = \frac{a - d_F}{a + \beta} \qquad 6\text{-}24$$

If E^a is to represent the input pattern then T_E^a should be greater than T_F^a, hence,

$\frac{b - d_E}{b + \beta} > \frac{a - d_F}{a + \beta}$ which implies that

$$d_E < b - \frac{(a - d_F)(b + \beta)}{a + \beta} \qquad 6\text{-}25$$

The second case is obvious.

Now we present the three results for three general values of β namely: very small value of β (close to zero), very large value of β (close to infinite) and intermediate values of β (any value between 0 and infinite). To produce these results we rely on the three theorems we have proven above.

Result 1: If an input pattern is presented to a UART architecture, with a very small β (close to zero), then between a FAM category R^a and an EAM category E^a, the input pattern will make the following choices:

a. If the pattern is inside both categories the pattern will be represented by the category with the smaller size.
b. If the pattern is inside the FAM category R^a and outside the EAM category E^a, the pattern will be represented by R^a.
c. If the pattern is inside the EAM category E^a and outside the FAM category R^a, the pattern will be represented by E^a.

d. If the pattern is outside both categories then, it will be represented by E^a iff $d_E < \frac{d_F b}{a}$, as a special case, if the size of R^a is equal to the size of E^a then the pattern will be represented by the category that is closer to the pattern (i.e. the category whose distance to the pattern is smallest)

Proof:

- **Result 1a**: Theorem1 states that if the pattern is inside both categories it will be represented by the category with the smallest size regardless of the value of β.
- **Result 1b**: Theorem 2 states that the pattern will be represented by E^a iff $d_E < g(\beta)$, but $g(\beta)_{\lim \beta \to 0} = 0$, and since d_E can never be less than zero then it is obvious that R^a will represent the pattern
- **Result 1c**: Theorem 3 states that the pattern will be represented by R^a iff $d_F < k(\beta)$, but $k(\beta)_{\lim \beta \to 0} = 0$, and since d_F can never be less than zero then it is obvious that E^a will represent the pattern
- **Result 1d**: Theorem 4 states that the pattern will be represented by E^a iff $d_E < h(\beta)$, but $h(\beta)_{\lim \beta \to 0} = \frac{d_F b}{a} = \frac{(M - 2r)d_F}{M - |R^a|}$. Furthermore, if $2r = |R^a|$ then $h(\beta)_{\lim \beta \to 0}$ becomes d_F.

Result 2: If an input pattern is presented to a UART architecture, with a very large β (close to infinity), then between a FAM category R^a and an EAM category E^a, the input pattern will make the following choices:

a. If the pattern is inside both categories the pattern will be represented by the category with the smaller size.

b. If the pattern is inside the FAM category R^a and outside the EAM category E^a, the pattern will be represented by E^a iff $d_E < b - a = |R^a| - 2r$ or $d_E + 2r < |R^a|$.

c. If the pattern is inside the EAM category E^a and outside the FAM category R^a, the pattern will be represented by R^a iff $d_F < a - b = 2r - |R^a|$ or $d_F + |R^a| < 2r$.

d. If the pattern is outside both categories then, it will be represented by E^a iff $d_E < b - a + d_F$, or equivalently if $d_E + 2r < d_F + |R^a|$. As a special case, if the size of R^a is equal to the size of E^a then the pattern will be represented by the category that is closer to the pattern (i.e. the category whose distance to the pattern is smallest)

Proof:

- **Result 2a**: Theorem1 states that the pattern if inside both categories will be represented by the one with smaller size regardless of the value of β.
- **Result 2b**: Theorem 2 states that the pattern will be represented by E^a iff $d_E < g(\beta)$, but $g(\beta)_{\lim \beta \to \infty} = b - a$. Equivalently this rule says that the pattern will be represented by E^a iff $d_E + 2r < |R^a|$. This rule reinforces the well-known fact with ART architectures that structures of smaller size are preferred by ART.
- **Result 2c**: Theorem 3 states that the pattern will be represented by R^a iff $d_F < k(\beta)$, but $k(\beta)_{\lim \beta \to \infty} = a - b$. Equivalently this rule says that the pattern will be represented by R^a iff $d_F + |R^a| < 2r$. This rule reinforces the well-known fact with ART architectures that structures of smaller size are preferred by ART.
- **Result 2d**: Theorem 4 states that the pattern will be represented by E^a iff $d_E < h(\beta)$, but $h(\beta)_{\lim \beta \to \infty} = b - a + d_F$. Equivalently, this rule says that the pattern will be represented by E^a iff $d_E + 2r < d_F + |R^a|$. This rule reinforces the well-known

fact with ART architectures that structures of smaller size are preferred by ART. Note that if the size of R^a is equal to the size of R^a , then $2r = |R^a|$, and consequently, E^a will be chosen over R^a , if and only if the distance of the pattern from E^a is smaller than the distance of the pattern from R^a. This statement also reinforces the common knowledge that in ART distances of input patterns from the ART structures matter (and the smaller the distance of a pattern from a structure the more likely it is for this structure to represent the input pattern).

Result 3: If an input pattern is presented to a UART architecture, with intermediate β values (i.e. $0 < \beta < \infty$), then between a FAM category R^a and an EAM category E^a, the input pattern will make the following choices:

a. If the pattern is inside both categories the pattern will be represented by the category with the smaller size.

b. If the pattern is inside the FAM category R^a and outside the EAM category E^a, the pattern will be represented by E^a iff $d_E < g(\beta) = b - \frac{a(b+\beta)}{a+\beta}$. Where $g(\beta)$ is a non-decreasing function of β and $0 < g(\beta) < b - a$ (assuming that size(E^a) < size(R^a)).

c. If the pattern is inside the EAM category E^a and outside the FAM category R^a, the pattern will be represented by R^a iff $k(\beta) = a - \frac{b(a+\beta)}{b+\beta}$. Where $k(\beta)$ is a non-decreasing function of β and $0 < k(\beta) < a - b$ (assuming that size(E^a) > size(R^a)).

d. If the pattern is outside both categories then, it will be represented by E^a iff

$d_E < h(\beta) = b - \frac{(a - d_F)(b + \beta)}{a + \beta}$, where $h(\beta)$ is a non-decreasing function of β

and $\frac{d_F b}{a} < h(\beta) < b - a + d_F$ (assuming that size(E^a) < size(R^a).

Proof:

- **Result 3a**:Theorem1 states that the pattern if inside both categories will be represented by the one with smaller size regardless of the value of β.
- **Result 3b**: Theorem 2 states that the pattern will be represented by E^a iff $d_E < g(\beta)$. To prove that $g(\beta)$ is non-decreasing function we take the derivative of $g(\beta)$ as follows: $\frac{dg(\beta)}{d\beta} = \frac{a(b-a)}{(a+\beta)^2}$, which is clearly always positive based on the assumption size(E^a) < size(R^a) => $b > a$, and knowing the fact that $a \geq 0$ and $b \geq 0$. Since $g(\beta)_{\lim \beta \to 0} = 0$ and $g(\beta)_{\lim \beta \to \infty} = b - a$, it follows that $0 < g(\beta) < b - a$.
- **Result 3c**: Theorem 3 states that the pattern will be represented by R^a iff $d_F < k(\beta)$. To prove that $k(\beta)$ is non-decreasing function we take the derivative of $k(\beta)$ as follows: $\frac{dk(\beta)}{d\beta} = \frac{b(a-b)}{(b+\beta)^2}$, which is clearly always positive based on the assumption size(E^a) > size(R^a) => $b < a$. Since $k(\beta)_{\lim \beta \to 0} = 0$ and $k(\beta)_{\lim \beta \to \infty} = a - b$, it follows that $0 < k(\beta) < a - b$.
- **Result 3d**: Theorem 4 states that the pattern will be represented by E^a iff $d_E < h(\beta)$. To prove that $h(\beta)$ is non-decreasing function we take the derivative of $h(\beta)$ as

follows: $\frac{dh(\beta)}{d\beta} = \frac{(b-a)(a-d_F)}{(a+\beta)^2}$, which is clearly always positive based on the assumption size(E^a) < size(R^a) => $b > a$, and knowing the fact that $a \geq d_F$. Since $h(\beta)_{\lim \beta \to 0} = \frac{d_F b}{a}$ and $h(\beta)_{\lim \beta \to \infty} = b - a + d_F$, it follows that

$\frac{d_F b}{a} < h(\beta) < b - a + d_F$.

6.2 Time Complexity Analysis

The process of finding an optimal or even suboptimal solution could be very lengthy and complicated, in this research we used GA to get this solution in previous research we used simpler method, we trained the network a large number of times, starting by an initial value for all the parameters of the network and then incrementing these parameters by specific increments, in an effort to exhaust all the combination based on the specific increments. Both approaches gave good results although GFAM approach has better solutions in most cases.

Let's assume that T_{train} of a FAM network is $O(N^3)$, and T_{test} is $O(N^2)$, where *N* is the number of nodes. GFAM simply works as follows

For 1 to Pop_{size} // # of chromosomes in a population

    Adjust parameters

Train a FAM network

For 1 to $Gen_{\max}$ // # generations in a run

    For 1 to Pop_{size}

        Apply GA Operators // TOperator and is O(N)

        Decode Chromosome to FAM //TDecode and is O(N)

Test FAM //Ttest

Encode FAM to Chromosome //TEncode and is O(N)

And hence, the time complexity of GFAM is roughly

$$Pop_{size} * Ttrain + Gen_{\max} * Pop_{size} * (Toperator + Tdecode + Ttest + Tencode)$$

Since Pop_{size} and $Gen_{\max}$ are variables we also assume that they are factors of N as follows

$Pop_{szie} = c8 * N$ and $Gen_{\max} = c9 * N$ by substituting values of individual complexities we get GFAM time as

$$c8 * N * Ttrain + c9 * N * c8 * N * Ttrain + c9 * N * c8 * N * (N + N + Ttest + N) \quad 6\text{-}26$$

If we substitute the values from T_{train} and T_{test}

$c8 * N * c4 * N^3 + c9 * N * c8 * N * (N + N + c7 * N^2 + N)$ which could be simplified as

$$c10 * N^4 + c11 * N^3 + c12 * N^4 \quad 6\text{-}27$$

which is clearly $O(N^4)$

For the Exhaustive Search (Linear increment of parameters) approach the algorithm is roughly like this

For 1 to P1 // P1 # number of increments of parmaeter1

Increment parameter1

For 1 to P2 // P2 # number of increments of parmaeter2

Increment parameter2

:

For 1 to *Pn* // *Pn* # number of increments of parametern (n is //the number of parameters for a specific classifier)

Increment parametern

Train FAM

Test FAM

In case of ssFAM we have four parameters (epsilon, order, rho and alpha) so we have four FOR statements and hence the exhaustive search time is

$$P1 * P2 * P3 * P4 * (Ttrain + Ttest) \quad \text{6-28}$$

Since *P1* .. *P4* are all variables dependent on the number of increments we will assume them as factors of N, and so 3 becomes

$$cp1 * N * cp2 * N * cp3 * N * cp4 * N * (Ttrain + Ttest) \Rightarrow cp * N^4 * (Ttrain + Ttest) \quad \text{6-29}$$

where *cp, cp1 … cp4* are all constants

If we substitute values from T_{train} and T_{test} in 4 we get

$$cp * N^4 * (c4 * N^3 + c7 * N^2) \Rightarrow c13 * N^7 + c14 * N^6 \quad \text{6-30}$$

which is $O(N^7)$

6-27 and 6-30 show us that there is a big difference between the time complexity of GFAM and the linear increment of properties approach.

Notice that the above analysis is a rough estimate and is only meant to show the difference between the two approaches. Also notice that here we only considered ssFAM which happened to have only four parameter to change, other networks could have five or more parameters and so their time complexity will be even worse than that of ssFAM while GFAM' frame work does not get affected by the number of parameters.

We also assumed that the number of nodes N is constant which absolutely not the case, however it is important to mention that in the training process N grows from 1 to a large number, while in the testing process N is constant. It is also important to know that in the GFAM approach the N tends to get smaller and smaller as the generations is incremented

(since we optimize on it) , until it converges to a very small number, this leads to even shorter time when testing the chromosome (FAM networks).

It is arguable that the number of increments for each parameter in the linear approach could be chosen to be a small number, this is true, but this defeats the purpose of this approach, because it was meant to try as many parameter combinations as possible to get the best performing network. And so the larger the number of increments of every parameter the better is the chance to find an optimal or suboptimal solution.

7. USER INTERFACE

It would have been difficult to achieve any of the results in this dissertation without the use of a custom built, easy to use, flexible and scalable user interface. In this chapter we present four similar in concept, but different in functionality user interface programs that were used to achieve the goals of this work.

GFAM UI, GEAM UI, GGAM UI and UART UI were developed and tested to find the best GFAM, GEAM, GGAM, and UART networks respectively. All four programs were developed using the C++ language and using the Borland CBuilder 6 software development kit. The user manual is presented in appendix E.

7.1 GFAM User Interface

The GFAM UI is a windows program that allows the user to build a generation of trained FAM networks of any number of individuals, select the training, validation and testing files, use different parameters to train the networks, define the genetic parameters, run the genetic operators, display the categories of any of the chromosomes along with the testing or validation patterns, display the classification borders of any chromosome, manually delete one or all the categories of a specific chromosome, and manually add categories to any chromosome. The program also allows the user to run the GA for a specific number of generations without stopping, or the user could define a number of generations at which the program stops and the then could continue per user command, or to step one generation at a time. The program shows the results of the best network whenever it stops, and also shows the results of individuals when they are displayed. The program logs a variable amount of information to an automatically named file, as follows. In the directory where the program resides it looks for a file called GFAM0.csv, if it finds it, it then looks for GFAM1.csv and so on, until it looks for a file called GFAMn.csv (n is a positive number) and it does not find it,

it creates this file and logs the data to it. The logged data can be customized by the user.

Figure 7-1 shows the GFAM interface program.

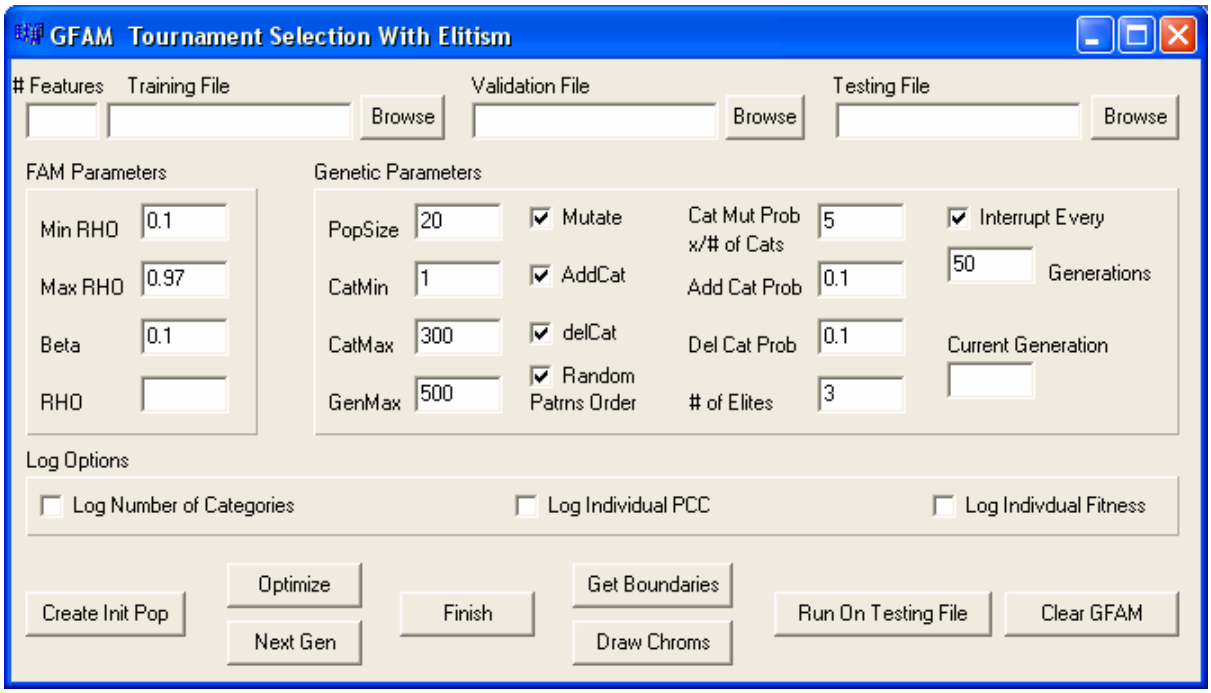

Figure 7-1: GFAM User interface

Now we define the controls on the window of Figure 7-1,

7.1.1 GFAM Controls

- # Features: The user inserts here the number of features in the problem at hand.
- Training File: The user inserts here the path of the file that contains the training data.
- Validation File: The user inserts here the path of the file that contains the validation data.
- Testing File: The user inserts here the path of the file that contains the testing data.
- Browse: These three buttons are used to open an Open File dialogue that allows the user to select the file from its location.

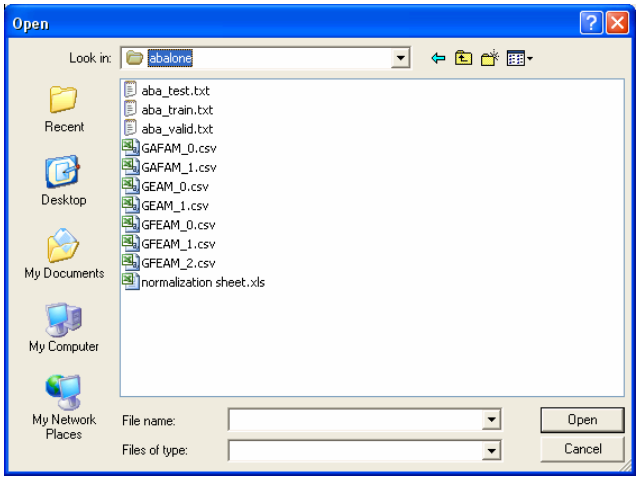

Figure 7-2: An open dialogue window, allows the user to select the training, validating and testing files

- Min RHO: Is used to train the first FAM network in a generation, and also used to determine the RHO' increment for the following networks. The default is 0.1.
- Max RHO: Is used to train the last FAM network in a generation, and also used to determine the RHO' increment for the following networks. The default is 0.97.
- Beta : Is used to set the FAM network β parameter. The default is 0.1.
- RHO: Displays the ρ parameter of the current network or chromosome.
- PopSize: Determine the number of chromosomes in a single generation. The default is set to 20.
- CatMin: The minimum number of categories a chromosome is allowed to have, used in the fitness function calculations. The default is set to 1.
- CatMax: The maximum number of categories a chromosome is allowed to have, used in the fitness function calculations. The default is set to 300.
- GenMax: The maximum number of generation the program has to process before it stops. The default is set to 500.
- Mutate (check box): If checked tells the program to use mutation in its calculations. Checked by default.

- AddCat (check box): If checked tells the program to use CatAdd operator in its calculations. Checked by default.
- DelCat (check box): If checked tells the program to use CatDell in its calculations. Checked by default.
- Random Ptrns Order (check box): If checked tells the program to randomly present the patterns when training the networks of the initial generation. Checked by default.
- Cat Mut Prob (x/ # of Cat's): This number (a positive integer) is used to calculate the probability of category mutation. This number divided by the number of categories in the chromosome is the probability of mutation. The default number is set to 5. The probability of mutation is never allowed to increase beyond the value of 1.0.
- Add Cat Prob: The probability of adding a category to a specific chromosome. Defaults to 0.1.
- Del Cat Prob: The probability of deleting a category from a specific chromosome. The default value is set to 0.1.
- # of Elites: Is the number of chromosomes that should transfer from one generation to the next one, without any modification. Usually the best chromosomes in a generation. The default value is set to 3.
- Interrupt Every (check box): If checked tells the program to stop after running a specific number of generations. The edit box that goes with it used to allow the user to insert any number of generations. Checked by default, the number of generations to interrupt after is set to 50.
- Current Generation: Displays the number of the current generation.
- Log Number of Categories (check box): If checked tells the program to log the number of categories of every chromosome in every generation. Not Checked by default.

- Log Individual PCC (check box): If checked tells the program to log the accuracy (on the validation set) of every chromosome in every generation. Not Checked by default.
- Log Individual Fitness (check box): If checked tells the program to log the value of the fitness function of every chromosome in every generation. Not Checked by default.
- Create Init Pop (Button): When clicked the program trains Pop_{size} FAM networks.
- Optimize (Button): When clicked the program starts the genetic optimization process, it goes on until the number $Gen_{\max}$ is reached, if the interrupt every check box is not checked. If the interrupt every check box is checked the program will stop every 50 generations and would display the results of the best chromosome (see e.g, Figure 7-3) .

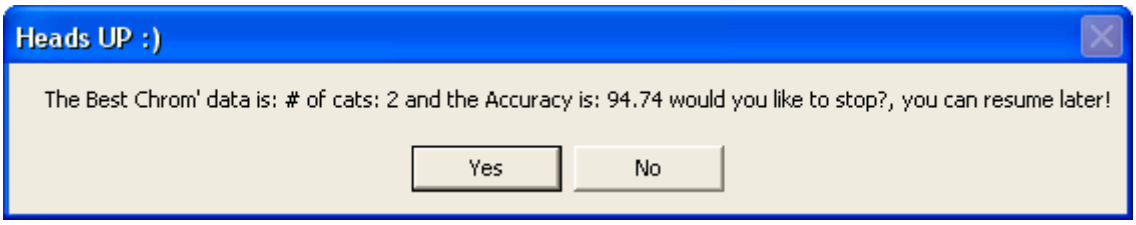

Figure 7-3: A dialogue box that displays the results after an interruption of the process

If the user clicks the No button the program resumes its execution, otherwise the program stops at the current generation. If the optimize button is clicked again the program resumes its execution until it reaches $Gen_{\max}$ or until it is interrupted again.

- Next Gen (Button): Allows the user to run the optimization process one generation at a time.
- Finish (Button): Does some cleaning and finishes logging some information in the file.

- Draw Chroms (Button): This button allows the user to see the categories that every chromosome has so far, and hence, it could be used to monitor the progress of the process and to see how well every chromosome is doing. Figure 7-4 shows the outcome when this button is clicked. This button displays categories only in 2D. If a problem has more than 2 features only the first 2 components of each vector will be displayed. Also, the graph window that appears when the user clicks this button has additional functionality that will be explained later.

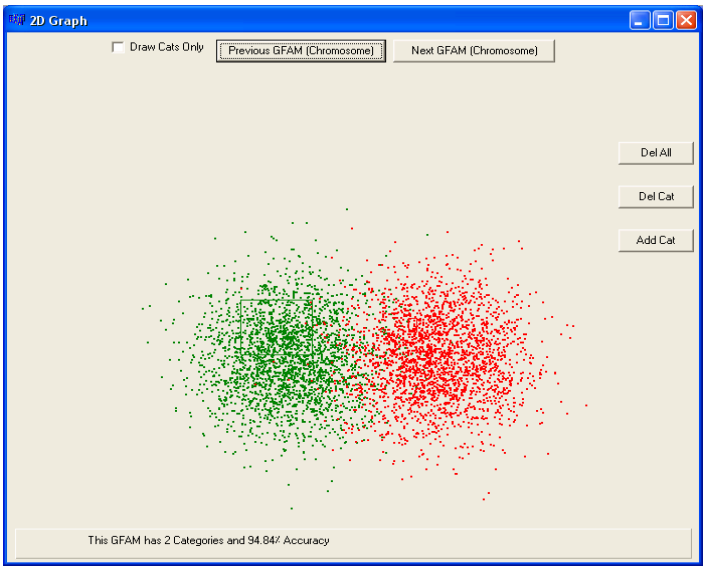

Figure 7-4: A 2D Graph that displays the data points as well as the categories

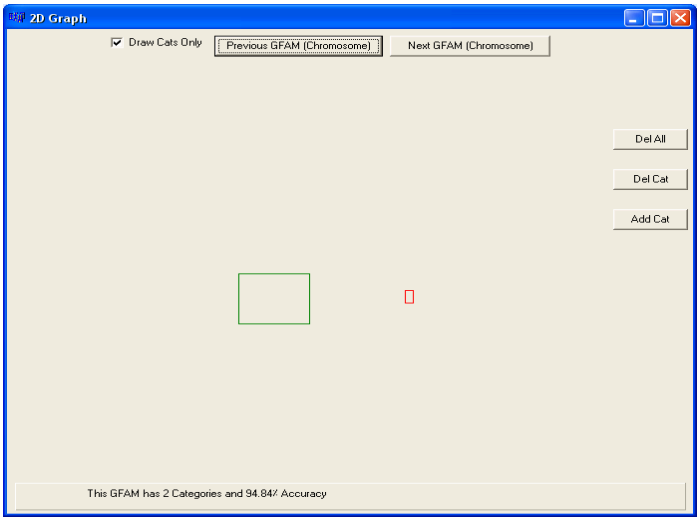

Figure 7-5: Same graph as in figure 7-4 but displaying the categories only

- Get Boundaries (Button): This feature allows the user to visually see how the network is classifying the input space. This feature works only for 2D problems. Figure 7-6 shows the outcome when this button is clicked. To achieve this goal, the program generates a list of 10000 patterns as a matrix with (0.01) increment on both x and y axis, and then feeds this list to the network and displays the way the network has classified these patterns by coloring them differently (in the figure two different colors were needed since this was a two class problem).

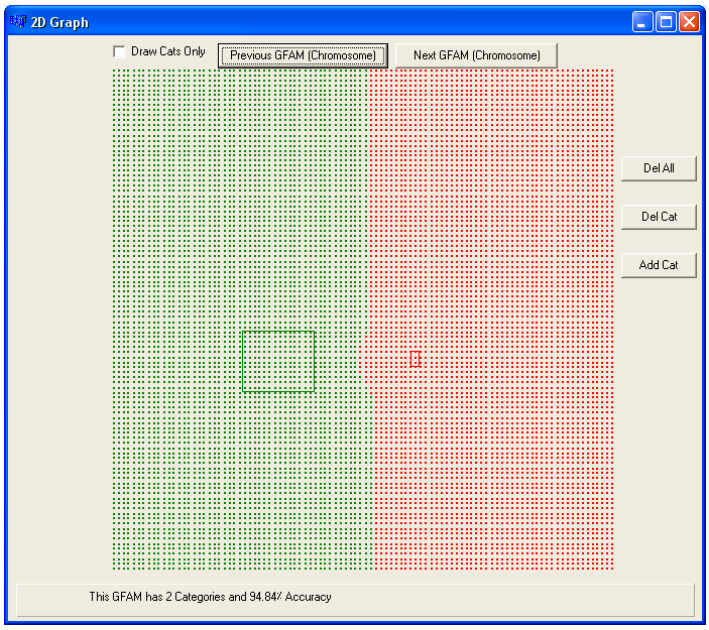

Figure 7-6: A 2D graph displaying the classification borders of this GFAM as well as the categories

- Run on Testing File (Button): Allows the user to run the performance phase on the selected testing file.
- Clear GFAM (Button): Clears the memory so that another problem can be checked.

All of the above were controls and their functionality on the main window. The graphing window (Figure 7-4) has some interesting functionality, as shown below.

- Draw Cats Only (Check Box): If checked the window will only graph the categories but not the patterns, otherwise, the categories as well as patterns (of the validation set) are displayed.

- Previous GFAM (Chromosome) (Button): Displays the content of the previous network.
- Next GFAM (Chromosome) (Button): Displays the content of the next network
- Status Bar: at the bottom of the window, the chromosome' accuracy and number of categories are shown.
- Del All (Button): This feature allows the user to delete all the categories of the current chromosome. (this feature is used to manually add categories and see how that affect the classification accuracy and boundaries).
- Del Cat: Deletes only one category from the current chromosome.
- Add Cat: Brings up a dialogue box that has few controls to allow the user to manually add a category to the current chromosome. Figure 7-9, explains this feature. After clicking the Del All button.

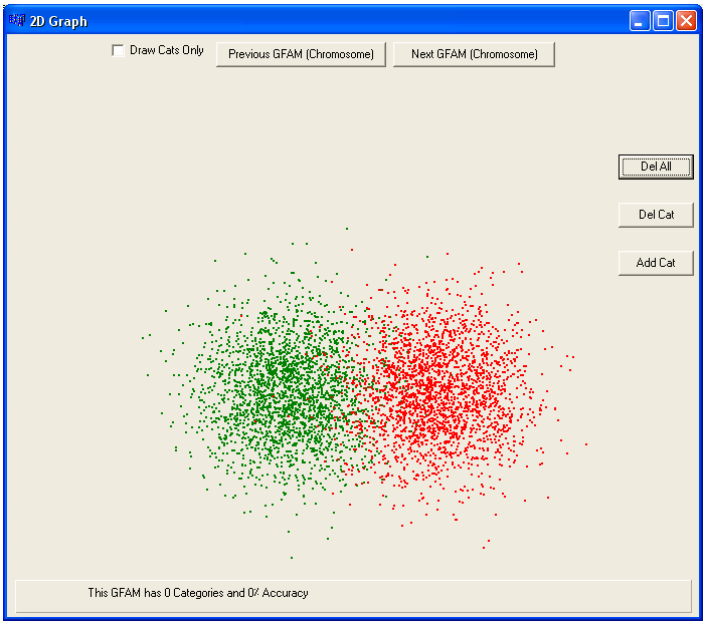

Figure 7-7: After pushing the "Del All" button, the GFAM does not have any more categories

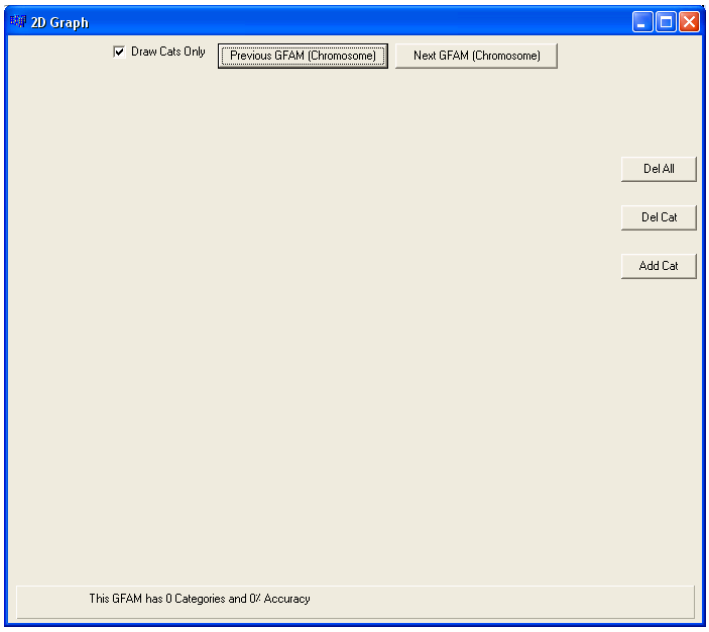

Figure 7-8: This figure shows figure 7-7, but only displaying the categories (none in this case)

after clicking the Add Cat button

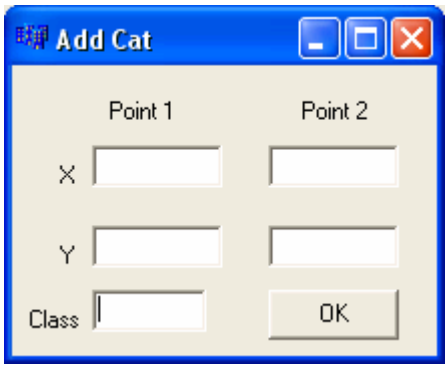

Figure 7-9: An add category dialogue box

Where: point 1 is the **u** vector, point 2 is the **v** vector. The Ok button adds the category to the chromosome:

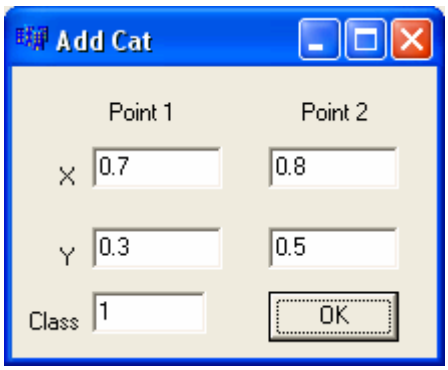

Figure 7-10: This figure is the same as figure 7-9 but after filling in some values

After clicking the ok button:

Figure 7-11: This figure shows the manually added category

Here is how the boundaries look now

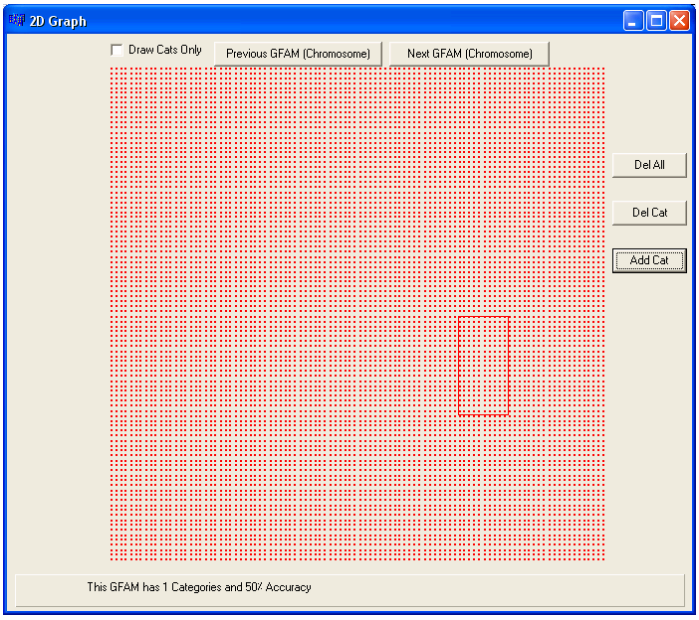

Figure 7-12: This figure shows the classification borders of the manually added category

The accuracy here was 50% because we only had two classes, and the number of patterns is divided 50-50 between them. After adding another category

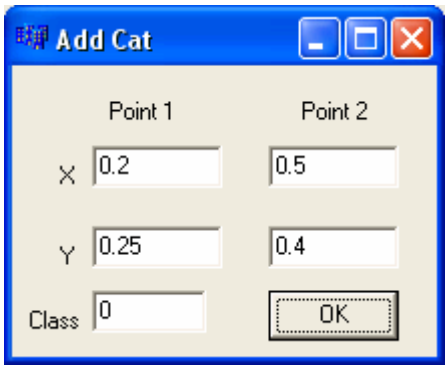

Figure 7-13: This figure shows the values of the endpoints of the second manually added category

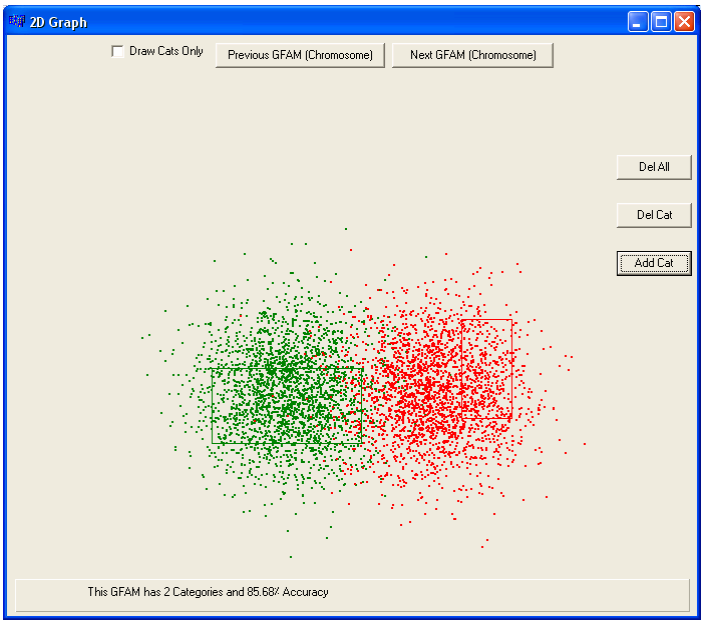

Figure 7-14: This figure shows the two manually added categories

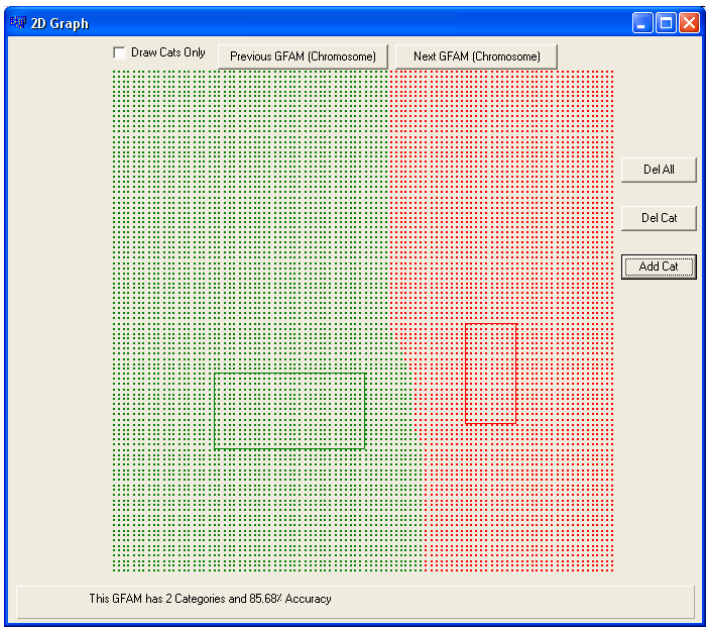

Figure 7-15: This figure shows the classification borders of the manually added categories

7.1.2 GFAM UI Abstract Design

GFAM UI consists of 7 main objects. These are:

- GFAMForm object: The main window object that shows most of the controls and allows for most of the user interaction.
- GraphForm object: The graph window, displays categories and classification boundaries, and allows manual deletion and addition of categories to any chromosome.
- FNode object: Represent a category in FAM network.
- PtrnNode object: Represent an input pattern.
- Chrom object: represent a chromosome, a collection of categories encoded in a special format.
- Cat object: Represent a FAM category after encoding.
- AddCatForm object: The Form that allows the user to insert the categories properties.

Now we present these seven objects in more details:

7.1.2.1 GFAMForm Object

This is the main window object and has most of the controls listed in 7.1.1. Its functionally is briefly described below.

- Validates the inputs.
- Creates the trained FAMs and stores them.
- Optimizes the generation of chromosomes to get the best GFAM.
- Displays the results and prepare the data for the graphing object.

The following is a list of the most important methods this object has:

- openFile: This method opens the files and reads the data creates PtrnNode objects and stores them into vectors.
- initPopClick: Creates FAM parameters and start the loop of creating FAMs.
- buildFam: Trains a FAM network.

- crop: Deletes all the categories that encoded one pattern only.
- decode: Takes a chromosome as an input and converts it to a FAM network.
- optimize: Optimizes the initial generation of trained FAMs.
- crossover: This function does the crossover process as described above.
- addCat: Adds a category to a given chromosome.
- delCat: Deletes a category from a given chromosome.
- mutation: This function mutates a given chromosome as described earlier.
- PrepBoundaries: Creates a list of 10000 patterns as a grid in 2D, their values are as follows (0,0),(0,0.01)…(0,1),(0.01,0)…(0.01,1)…(1,1). It also initialize some other variables for the graphForm object to use.
- drawCats: Initializes some variables and calls the graphForm object.
- showBoundaries: Initializes some variables and calls the graphForm object.
- ~GFAMForm: Destructor function.

This object also has many properties (Variables) and below is a list of the most important ones:

- A vector of FNode objects: Stores FNode objects as a FAM network.
- A vector of strings: Stores the names of the classes in a problem.
- A vector of PtrnNode objects: Stores the training patterns.
- A vector of PtrnNode objects: Stores the testing patterns.
- A vector of Chrom object pointers: Stores pointers to all the created chromosomes in specific generation.
- A vector of Chrom object pointers: Stores pointers to all the chromosomes in temp generation.
- A vector of integers: Stores the predicted labels after running the performance phase.
- An integer: Stores the number of created categories for a specific network.

These variables and many others are used in this object. However, since a detailed design of this UI is not the purpose of this dissertation, the details are not shown here.

7.1.2.2 GraphForm Object

This object is much simpler than the GFAMForm object, it is responsible for showing the categories, the validation patterns, and the classification boundaries of a specific chromosome. It is also responsible of allowing the user to manually add and delete categories from the chromosome. The following is a list of the main methods and variables of this object:

- pbPaint: This function displays the categories along with the validation patterns or the grid of automatically generated patterns along with the categories (classification boundaries). This function is one of the most difficult functions in this UI, since one has to manually convert the origin of the graphing object such that the (0,0) point is the bottom left corner.
- nextClick: This functions displays the graph of the next chromosome.
- previousClick: This function displays the graphs of the previous chromosome.
- addcClick: This function calls the AddCatForm object so that the user can insert the information of the new category.
- delAllCat: This functions invokes the del function of the Chrom object which deletes all the categories from that chromosome.
- ~GraphForm: Destructor function.

Few variables exist in this object and the following is a list of the main ones:

- An Integer c: Is the index of the current chromosome.
- An array of Point object pointers: Stores the points that are displayed in the graphing area.

- A pointer to GFAMForm object: This object gives the GraphForm object access to many functions and variables from the main window.
- A boolean bounds: Tells the object if the categories are to be displayed or the classification boundaries.

7.1.2.3 FNode Object

This object represents a FAM category, A list of the main methods and variables in this object is listed below.

- FNode: This is the constructor. It creates a new FNode object and initializes it either as a new category (single point), or as a template where the information of the template is passed as an argument to this function.
- calc_scaled_CMF: This function calculates the CMF value of this category.
- calc_CCF: This function calculates the CCF value of this category.
- update: This function updates the category during the training session.
- ~FNode: Destructor function.

The main variables are:

- Integer M: The number of features.
- Double Beta: This is the β parameter.
- Double learning_rate: This is set to 1, fast learning scenario.
- Array of Doubles U: Represents the $\mathbf{u}$ vector in FAM.
- Array of Doubles V: Represents the $\mathbf{v}$ vector in FAM.
- Integer Label: represents the category label.

7.1.2.4 PtrnNode Object

This is a very simple object that represent an input pattern. Here is a list of its functions and variables:

- PtrnNode: This is the constructor. It initializes the object from the input arguments it gets.
- ~PtrnNode: Destructor function.

This object has the following variables:

- An array of doubles w: This array holds the value of the features of an input pattern.
- Integer classIndex: This integer holds the label of the pattern.
- Integer wc: It represents the number of features this pattern has.

7.1.2.5 Chrom Object

This object represents a chromosome, which is a collection of categories as shown in Figure 4-1. A chromosome object has the following functions:

- Chrom: It is the constructor of this object, and it has a very important functionality that is converting the FNode object (FAM categories) into Cat objects that are suitable for genetic manipulation. There is another version of this function that creates an empty Chrom object.
- Del: This function deletes all the categories the Chrom object has. This operation is used when manual addition and deletion of the categories is invoked.
- ~Chrom: Destructor function.

A Chrom object has the following variables:

- A vector of Cat object pointers: This vector holds pointers to all Cat object this chromosome has.
- Double fit: This variable holds the fitness value of this chromosome.
- Double rho: This holds the vigilance parameter of the FAM network represented by this chromosome.
- Integer size: This is the number of categories in this chromosome.
- Double accuracy: This variable holds the accuracy of this network.

7.1.2.6 Cat Object

This is an object that resembles an encoded category. This category is ready for genetic manipulations, and here is a list of its functions:

- Cat: This is the constructor, and it takes as an input an FNode object and extracts the data from it and saves it in its internal variables. Another version of it takes as an input a Cat object pointer, and this one copies all the information of the original object to the newly created one.
- pointCat: Returns a Boolean indicating whether or not this category is a point category.
- ~Cat: Destructor function.

The following variables exist in this object:

- Array of Doubles v: obvious.
- Array of Doubles u: obvious.
- Integer l: class label.
- Integer s: number of features.

7.1.2.7 AddCatForm Object

This object is a small form that has six controls to all the user to insert values for manually added categories. It has one function other than constructor that returns a one if a some values where inserted.

In the following sections, the user interface of GEAM, GGAM and UART will be discussed very briefly, since they are very similar to the UI of GFAM. We will only focus on the differences between the two UI's.

7.2 GEAM User Interface

GEAM UI is a windows program that allows the user to build a generation of trained EAM networks of any number of individuals, select the training, validation and testing files,

use different parameters to train the networks, define the genetic parameters, run the genetic operators, display the categories of any of the chromosomes along with the testing or validation patterns, display the classification borders of any chromosome, manually delete one or all the categories of a specific chromosome, and manually add categories to any chromosome. The program also allows the user to run a specific number of generations without stop, or run a specific number of generations at which the program stops and then continue per user command, or finally to run one generation at a time. The program shows the results of the best network whenever it stops, and also shows the results of individuals when they are displayed. The program logs a variable amount of information to an automatically named file, as follows. In the directory where the program resides it looks for a file called GEAM0.csv, if it finds it looks for a file called GEAM1.csv, and so on, until it looks for a file called GEAMn.csv (n is a positive number) and if it does not find it, it creates it and logs the data to it. The logged data can be customized by the user. Figure 7-16 shows the GEAM interface program.

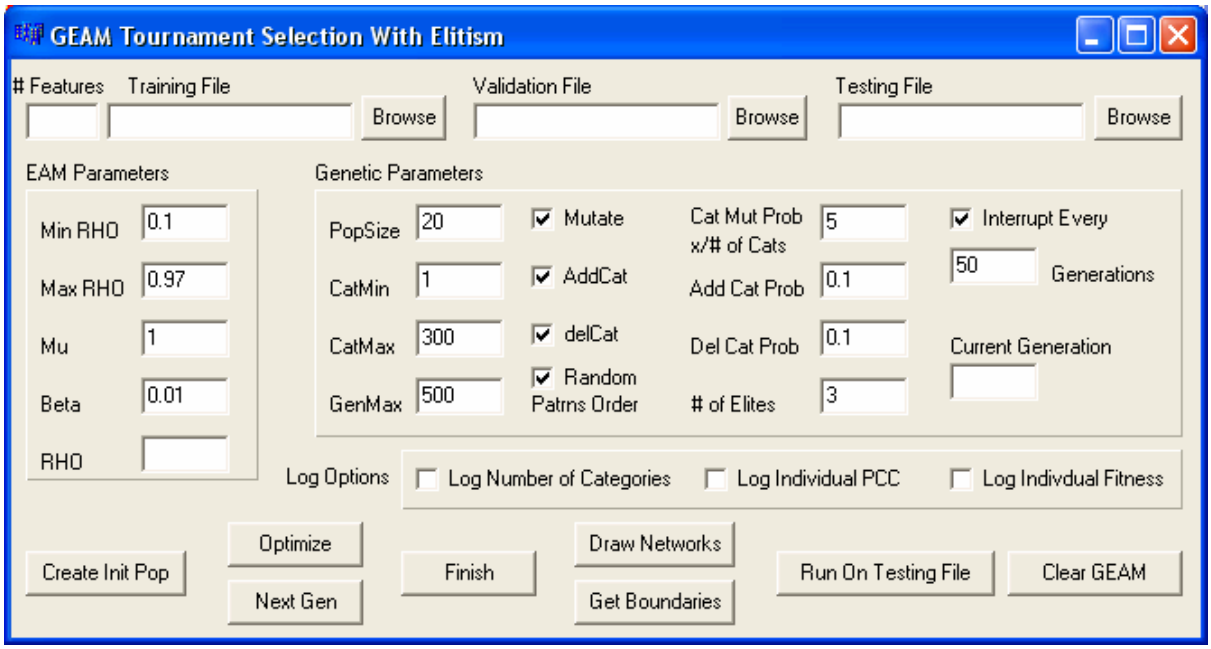

Figure 7-16: GEAM user interface

7.2.1 GEAM Controls

Here we only define the controls on the window of Figure 7-16, that are different from those found in GFAM UI.

- Mu: This parameter is the axis ratio of EAM. The default is set to 1, meaning that the ellipsoids in EAM are actually circles.
- Beta : This parameter is used to set the EAM network β parameter. The default is set to 0.1.
- Create Init Pop (Button): When clicked the program trains Pop_{size} EAM networks.
- Draw Chroms (Button): This button allows the user to see the categories that every chromosome has so far, and hence, it could be used to monitor the progress of the GA process and to see how well every chromosome is doing. Figure 7-17 shows the outcome when this button is clicked (this button displays categories only in 2D; if a problem has more than 2 feature, the first 2 components of each vector will be displayed. Also, the graph window that appear when the user clicks this button has significant functionality that will be explained later).

Figure 7-17: a 2D graph displaying a GEAM network; note here ellipsoids are represented by circles

- Get Boundaries (Button): This feature allows the user to visually see how the network is classifying the input space. This feature works only for 2D problems. Figure 7-18

shows the outcome when this button is clicked. To achieve this goal, the program generates a list of 10000 patterns as a matrix with (0.01) increment on both x and y, and then feeds this list to the network and displays the way the network has classified these patterns by coloring them differently (in the fugure two different colors were needed since this was a two class problem).

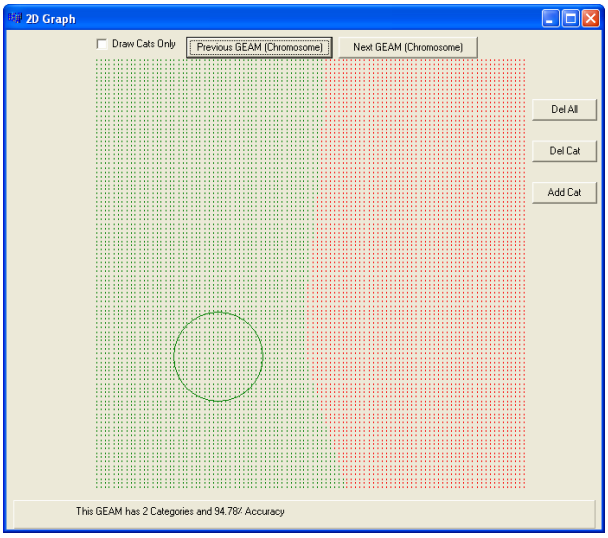

Figure 7 -18: A 2D graph showing the classification borders of the GEAM network

- Clear GEAM (Button): Clears the memory so that another problem can be checked.

The graphing window of GEAM UI (Figure 7-17) has similar functionality to that of a GFAM UI. The differences are discussed below.

- Previous GEAM (Chromosome) (Button): Displays the content of the previous network.
- Next GEAM (Chromosome) (Button): Displays the content of the next network.
- Add Cat: Brings up a dialogue box that has some controls to allow the user to manually add a category to the current chromosome. Figure 7-20 explains this feature. After clicking the Del All button.

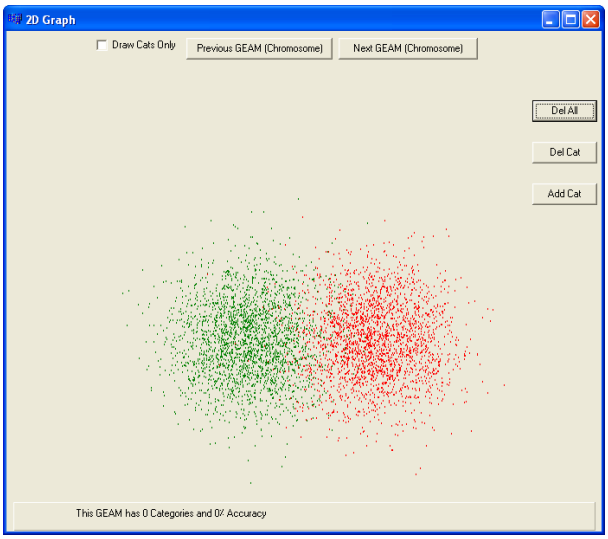

Figure 7 -19: This figure shows the GEAM network after pushing the "Del All" button

After clicking the Add Cat button, we get the following screen:

Figure 7-20: An add GEAM category dialogue box

Where, Center is the **m** vector, Direction is the **d** vector, Class edit box is the label of the category, Mu edit box is the axis ratio and the Rad edit box is the radius. The Ok button adds the category to the chromosome:

Figure 7-21: Manually filling in values for the first category

After clicking the ok button, we get the following screen:

Figure 7-22: GEAM network after manually adding a category

Now, the network classification boundaries look as follows:

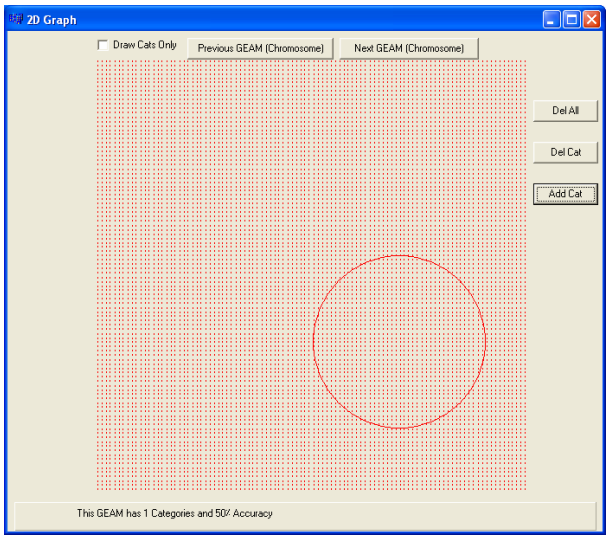

Figure 7-23: Classification borders of the manually added category

The accuracy here was 50% because we only have two classes here, and the number of patterns is divided 50-50 between them.

Figure 7-24: Manually adding a new category

After adding another category we obtain the information depicted in the following screen

Figure 7-25: GEAM network after adding the second category

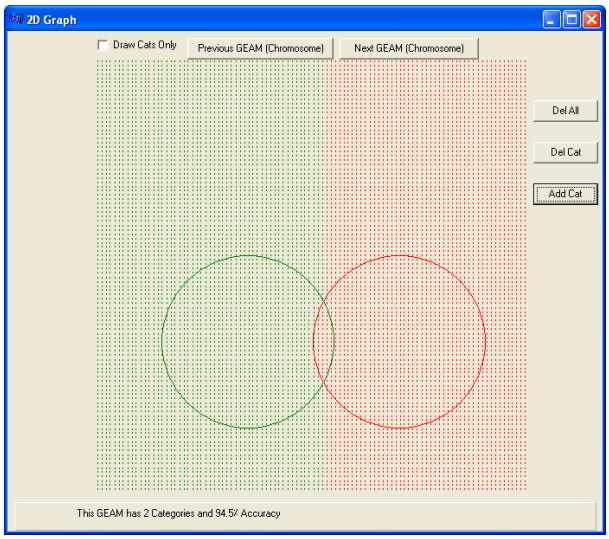

Figure 1-26: Classification borders of the GEAM network

7.2.2 GEAM UI Abstract Design

GEAM UI consists of 7 main objects. These objects are:

- GEAMForm object: The main window object that shows most of the controls and allows for most of the user interaction.
- GraphForm object: This is the graph window that displays categories and classification boundaries, and allows manual deletion and addition of categories to any chromosome.
- ENode object: Represents a category in an EAM network.

- PtrnNode object: Represents an input pattern.
- Chrom object: Represents a chromosome, a collection of categories encoded in a special format.
- Cat object: Represents a EAM category after encoding.
- AddCatForm object: This is the Form that allows the user to insert the categories properties.

Now we present these seven objects in more detail:

7.2.2.1 GEAMForm Object

This is the main window object and it has most of the controls listed in 7.2.1. It employs a number of methods and possesses capabilities that allow for its functionality. This functionality is briefly described in the following points:

- Validates the inputs.
- Creates the trained EAMs and stores them.
- Optimizes the generation of chromosomes to get the best GEAM.
- Displays the results and prepare the data for the graphing object.

The following is a list of the most important methods this objects has that are different from those described in the GFAM UI:

- initPopClick: Creates EAM parameters and start the loop of creating EAMs.
- buildEAM: Trains an EAM network.
- decode: Takes a chromosome as an input and converts it to an EAM network.
- optimize: Optimizes the initial generation of trained EAMs.
- mutation: This function mutates a given chromosome as described earlier in this manuscript.
- ~GEAMForm: Destructor function.

This object has one variable that is not listed in those described in GFAM UI:

- A vector of ENode objects: Stores ENode objects as an EAM network.

7.2.2.2 GraphForm Object

This object is very similar to the GraphForm Object in GFAM UI. However this object displays ellipsoids rather than rectangles for any category.

7.2.2.3 ENode Object

This object represents an EAM category. In the following we describe a list of the main methods and variables this object uses:

- ENode: This is the constructor, it creates a new ENode object and initializes it either as a new category (single point), or as a template where the information of the template is passed as an argument to this function.
- calc_scaled_CMF: This function calculates the CMF value of this category.
- calc_CCF: This function calculates the CCF value of this category.
- update: This function updates the category during the training session.
- ~GNode: Destructor function.

The main variables are:

- Integer M: The number of features.
- Double Beta: This is the β parameter.
- Double learning_rate: This is set to 1, corresponding to fast learning.
- Array of Doubles m: Represents the $\mathbf{m}$ vector in EAM.
- Array of Doubles d: Represents the $\mathbf{d}$ vector in EAM.
- A double r: Represents the radius.
- A double mu: Represents the axis ratio.
- Integer Label: Represents the category label.

7.2.2.4 PtrnNode Object

This object is copy of the PtrnNode object used in GFAM UI.

7.2.2.5 Chrom Object

This object represents a chromosome, which is a collection of categories as shown in figure 4-1. The only function that is different from those in the Chrom Object in GFAM UI is:

- Chrom: This is the constructor of this object. It has a very important functionality that is converting the FNode object (EAM categories) into Cat objects that are suitable for genetic manipulation. There is another version of this function that creates an empty Chrom object.

This object has the same variables as those of Chrom Object in GFAM UI.

7.2.2.6 Cat Object

This is an object that resembles an encoded category. This category is ready for genetic manipulations. Here is a list of its functions:

- Cat: This is the constructor, it takes as an input a ENode object and extracts the data from it and save in its internal variables. Another version of it takes as an input a Cat object pointer; this one copies all the information of the original object to the newly created one.
- pointCat: Returns a Boolean indicating whether or not this category is a point category.
- ~Cat: Destructor function.

The following variables exist in this object:

- Array of Doubles m: Obvious.
- Array of Doubles d: Obvious.
- Double r: This is the radius of the EAM category.
- Double mu: This is the axis ratio of the EAM category.

- Integer l: This is the class label of the EAM category.
- Integer s: Represents the number of features (i.e. the problem dimensionality).

7.2.2.7 AddCatForm Object

This object is a small form that has eight controls to allow the user to insert values for manually added categories. It has one function other than the constructor that returns a one if some values were inserted.

7.3 GGAM User Interface

GGAM UI is a windows program that allows the user to build a generation of trained GAM networks of any number of individuals, select the training, validation and testing files, use different parameters to train the networks, define the genetic parameters, run the genetic operators, display the categories of any of the chromosomes along with the testing or validation patterns, display the classification borders of any chromosome, manually delete one or all the categories of a specific chromosome, and manually add categories to any chromosome. The program also allows the user to run a specific number of generations without stop, or the user could define a number of generation at which the program stops and the then could continue per user command, or to step one generation at a time. The program shows the results of the best network whenever it stops, and also shows the results of individuals when they are displayed. The program logs a variable amount of information to an automatically named file, as follows. In the directory where the program resides, it looks for a file called GGAM0.csv, if it finds it looks for a file called GGAM1.csv, and so on, until the time comes to look for a file called GGAMn.csv (n is a positive number); if it does not find the file, the program creates it, and logs the data to it. The logged data can be customized by the user. Figure 7-27 shows the GGAM interface program.

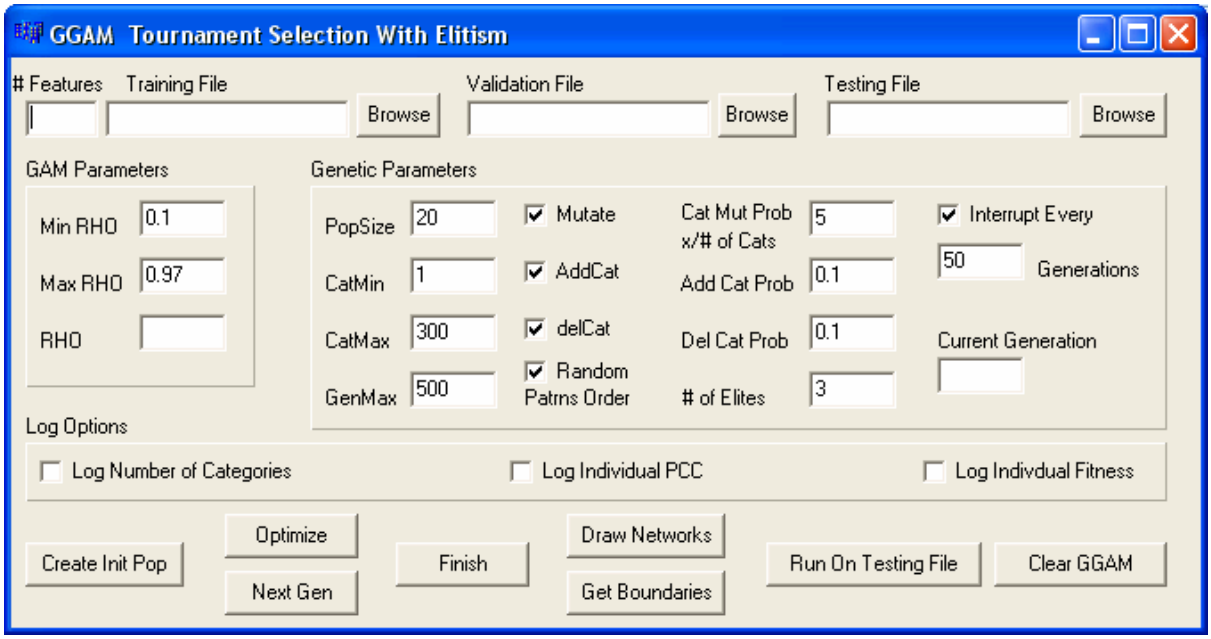

Figure 7-27: GGAM user interface

7.3.1 GGAM Controls

Here we define the controls on the window of Figure 7-27, that are different from those found in GFAM UI.

- Create Init Pop (Button): When clicked the program trains Pop_{size} GAM networks.
- Draw Chroms (Button): This button allows the user to see the categories that every chromosome has so far, and hence, it could be used to monitor the progress of the process and observe how well every chromosome is doing. Figure 7-28 shows the outcome when this button is clicked (this button displays categories only in 2D, if a problem has more than 2 features and the button is clicked the first 2 components of each vector will be displayed. Also, the graph window that appears when the user clicks this button has significant functionality that will be explained later). Since a bell-shaped normal distribution is hard to graph, we show the center of each category as a big dot.

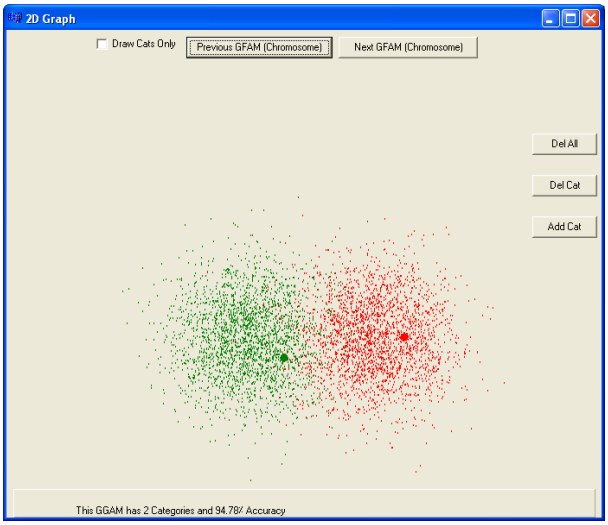

Figure 7-28: A 2D graph showing a GGAM network; note here a GGAM category is represented by a large dot

- Get Boundaries (Button): This feature allows the user to visually see how the network is classifying the input space. This feature works only for 2D problems. Figure 7-29 shows the outcome when this button is clicked. To achieve this goal, the program generates a list of 10000 patterns as a matrix with (0.01) increment on both x and y, and then feeds this list to the network and displays the way the network has classified these patterns by coloring them differently (in the figure two different colors were needed since this was a two class problem).

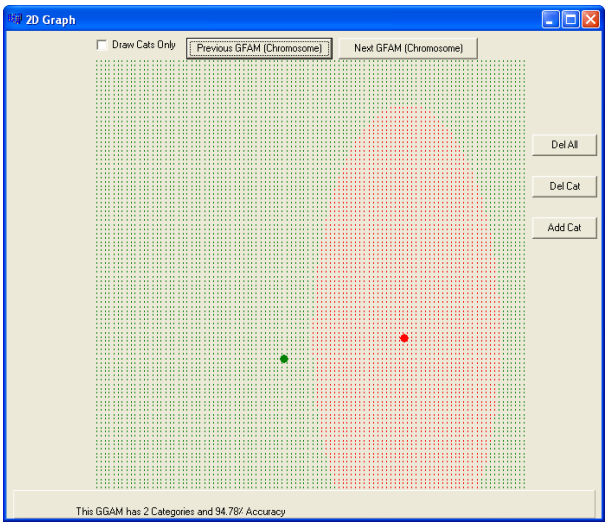

Figure 7-29: A 2D graph showing the classification boundaries of the GGAM network

- Clear GGAM (Button): Clears the memory so that another problem can be checked

The graphing window of GGAM UI (Figure 7-28) has similar functionality to that of a GFAM UI also, here are the differences

- Previous GGAM (Chromosome) (Button): Displays the content of the previous network.
- Next GGAM (Chromosome) (Button): Displays the content of the next network.
- Add Cat: Brings up a dialogue box that has controls to allow the user to manually add a category to the current chromosome. Figure 7-30, explains this feature: after clicking the Del All button.

Figure 7-30: After deleting all the categories

after clicking the Add Cat button

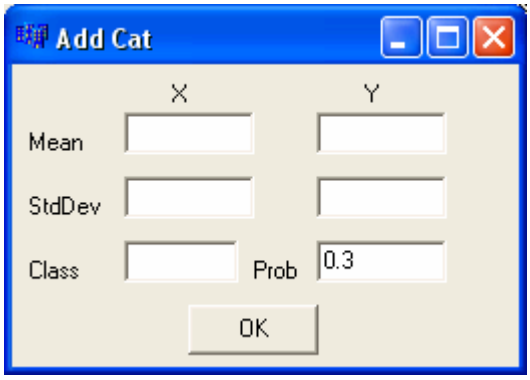

Figure 7-31: An add GGAM category dialogue box

where, Mean is the μ vector, Direction is the σ vector, Class edit box is the label of the category, Prob edit box is the probability of the category. The Ok button adds the category to the chromosome:

Figure 7-32: Add category dialogue box with numbers in the available boxes

after clicking the ok button:

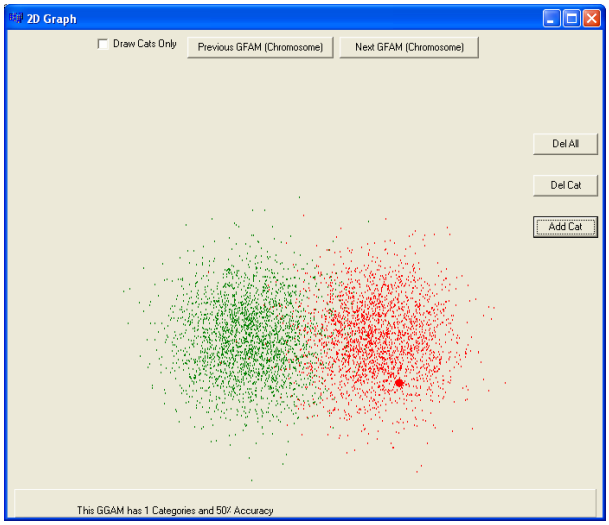

Figure 7-33: A figure showing the manually added category

and here is how the boundaries look now with the added category

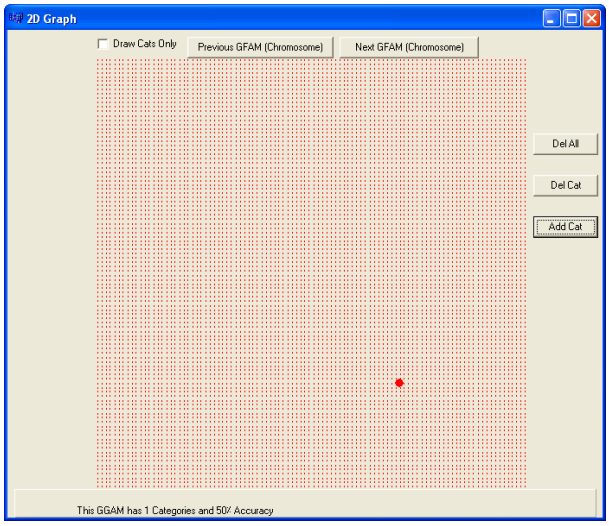

Figure 7-34: The classification boundaries corresponding to the manually added category

The accuracy of the network is 50% because we have two classes, and the number of patterns is divided equally amongst them. After adding another category

Figure 7-35: Filling in numbers for the second category

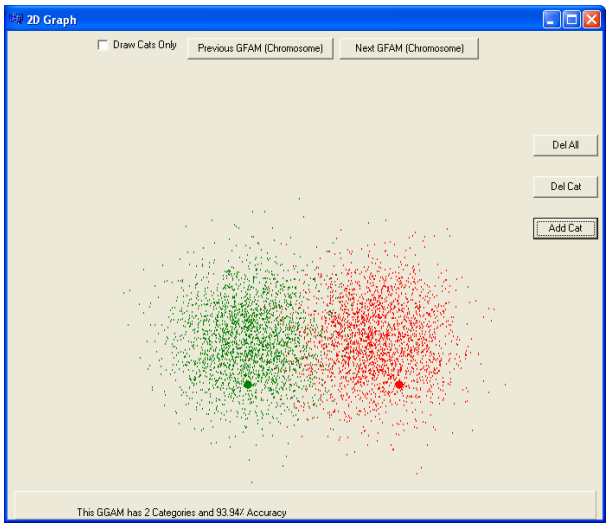

Figure 7-36: The GGAM network after adding the second category

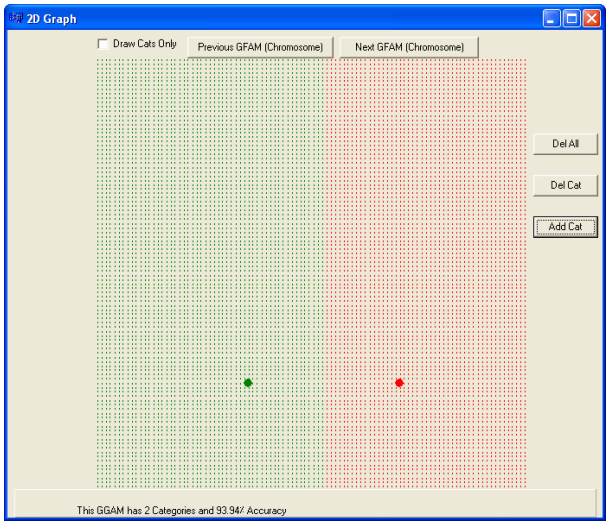

Figure 7-37: The classification boundaries of the GGAM after adding two categories

7.3.2 GGAM UI Abstract Design

GGAM UI consists of 7 main objects that are listed below.

- GGAMForm object: The main window object that shows most of the controls and has most of the user interaction.
- GraphForm object: This is the graph window that displays categories and classification boundaries, and allows manual deletion and addition of categories to any chromosome.
- GNode object: Represents a category in an GAM network.
- PtrnNode object: Represents an input pattern.
- Chrom object: Represents a chromosome, that is a collection of categories encoded in a special format.
- Cat object: Represents a GAM category after encoding.
- AddCatForm object: This is the Form that allows the user to insert the categories' properties.

Now we discuss these seven objects in more detail:

7.3.2.1 GGAMForm Object

This is the main window object that has most of the controls listed in 7.2.1. It has a number of features and employs a number of methods that allows it to accomplish its functionality. This functionality is briefly described in the following:

- Validates the inputs.
- Creates the trained GAMs and stores them.
- Optimizes the generation of chromosomes to get the best GGAM.
- Displays the results and prepares the data for the graphing object.

The following is a list of the most important methods that this object employs that are different from those described in the GFAM UI.

- initPopClick: Creates GAM parameters and start the loop of creating GAMs.
- buildGAM: Trains a GAM network.
- decode: Takes a chromosome as an input and converts it to a GAM network.
- optimize: Optimizes the initial generation of trained GAMs.
- mutation: This function mutates a given chromosome as described earlier.
- ~GGAMForm: Destructor function.

The main different variable from those described in GFAM UI that this object has is:

- A vector of GNode objects: Stores GNode objects as a GAM network.

7.3.2.2 GraphForm Object

The GraphForm Object is very similar to the GraphForm Object in GFAM UI, however this one displays big dots at the center of the category, rather than rectangles.

7.3.2.3 GNode Object

This object represents a GAM category. Here is a list of the main methods that this object employs and variables that it uses:

- GNode: This is the constructor, it creates a new GNode object and initializes it either as a new category (single point), or as a template where the information about this template is passed as an argument to this function.
- calc_scaled_CMF: This function calculates the CMF value of this category.
- calc_CCF: This function calculates the CCF value of this category.
- update: This function updates the category during the training session.
- ~GNode: Destructor function.

here are the main variables:

- Integer M: The number of features.
- Array of Doubles micro: Represents the μ vector in GAM.
- Array of Doubles sigma: Represents the σ vector in GAM.
- A double prob: Represents the probability of a category in GAM.
- Integer Label: Represents the category label.

7.3.2.4 PtrnNode Object

This object is a clone of the PtrnNode object of a GFAM UI, and no further discussion is needed.

7.3.2.5 Chrom Object

This object represents a chromosome, which is a collection of categories as shown in Figure 4-3, the only function that is different from those in the Chrom Object in GFAM UI is:

- Chrom: This is the constructor of this object. It has a very important functionality, which is converting the GNode object (GAM categories) into Cat objects that are suitable for genetic manipulation. There is another version of this function that creates an empty Chrom object.

This object has the same variables as those of the Chrom Object in GFAM UI.

7.3.2.6 Cat Object

This is an object that resembles an encoded category. This category is ready for genetic manipulations, and in the following we present a list of its functions:

- Cat: This is the constructor that takes as an input a GNode object and extracts the data from it, and saves this data in its internal variables. Another version of it takes as an input a Cat object pointer; this version copies all the information of the original object to the newly created one.
- pointCat: Returns a Boolean indicating whether or not this category is a point category.
- ~Cat: Destructor function.

The following variables exist in this object:

- Array of Doubles micro: Obvious.
- Array of Doubles sigma: Obvious.
- Double prob: This is the probability of this category (i.e. number of encoded patterns over the total number of training patterns).
- Integer l: This is the class label.
- Integer s: This integer represents the number of features (i.e. problem's input dimensionality).

7.3.2.7 AddCatForm Object

This object is a small form that has eight controls. It allows the user to insert values for manually added categories. It has one function other than the constructor, that returns a one if some values were inserted.

7.4 UART User Interface

UART UI is a windows program that allows the user to build a generation of trained FAM networks and trained EAM networks of any even number of individuals, select the training, validation and testing files, use different parameters to train the networks, define the

genetic parameters, run the genetic operators, display the categories of any of the chromosomes along with the testing or validation patterns, display the classification boundaries of any chromosome, manually delete one or all the categories of a specific chromosome, and manually add categories to any chromosome. The program also allows the user to run a specific number of generations without stop, or the user could define a number of generations at which the program stops and then could continue per user command, or to step one generation at a time. The program shows the results of the best network whenever it stops, and also shows the results of individuals when they are displayed. The program logs a variable amount of information to an automatically named file, as follows: In the directory where the program resides it looks for a file called UART0.csv, if it finds it looks for a file named UART1.csv, and so on, until the time comes to look for a file called UARTn.csv (n is a positive number); if it does not find it creates it and logs the data to it. The logged data can be customized by the user. Figure 7-38 shows the UART interface program.

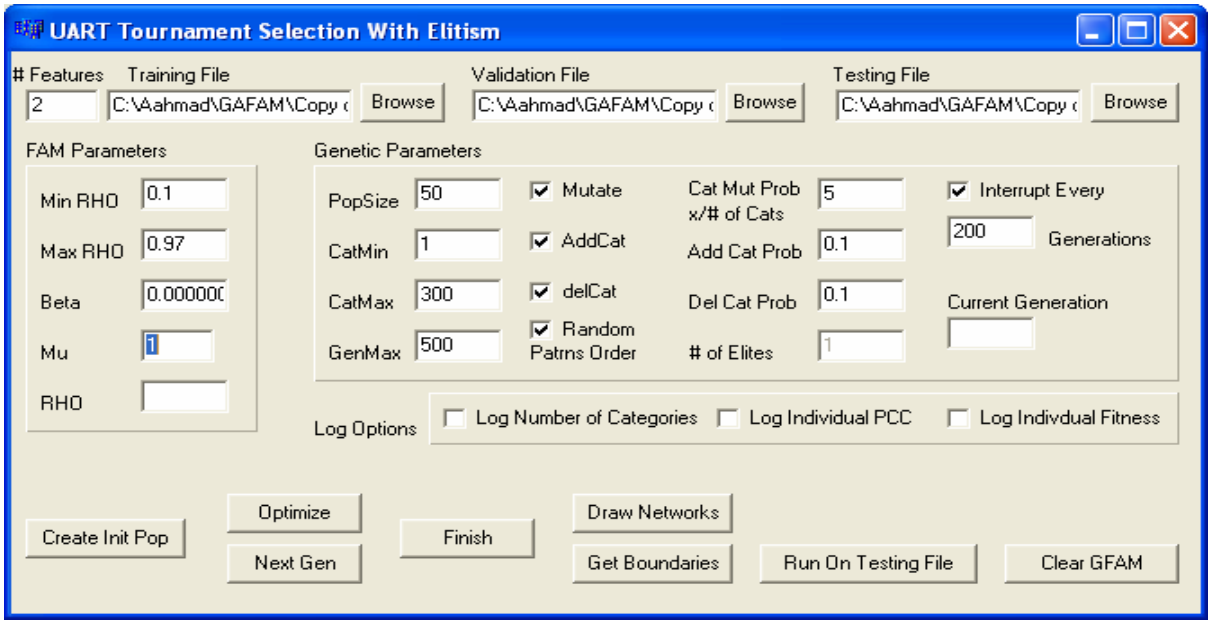

Figure 7-38: UART user interface

Although the look and feel of the UI is similar to that of GFAM and GEAM, UART UI operations and design are very different. Now we define the controls on the window of Figure 7-38 that are different from those in GEAM UI.

7.4.1 UART Controls

- # Features: This is the number of features, used by both EAM and FAM networks.
- Training File: The user inserts here the path of the file that contains the training patterns, used by both EAM and FAM networks.
- Validation File: The user inserts here the path of the file that contains the validation patterns, used by both EAM and FAM networks.
- Beta : This is used to set the FAM and EAM β parameter. The defaults is equal to 0.1.
- RHO: This field displays the ρ parameter of the current network or chromosome.
- PopSize: This control determines the number of chromosomes in a single generation. The defaults is set to 20, 10 FAM and 10 EAM networks.
- Create Init Pop (Button): When this button is clicked the program trains $Pop_{size}/2$ FAM networks and $Pop_{size}/2$ EAM networks.
- Draw Chroms (Button): This button allows the user to see the categories that every chromosome has so far, and hence, it could be used to monitor the progress of the process and to see how well every chromosome is doing. Figure 7-39 shows the outcome when this button is clicked. This button displays categories only in 2D, and if a problem has more than 2 features and this button is clicked, the first 2 components of each vector will be displayed. Also, the graph window that appears when the user clicks this button has a great functionality that will be explained later).

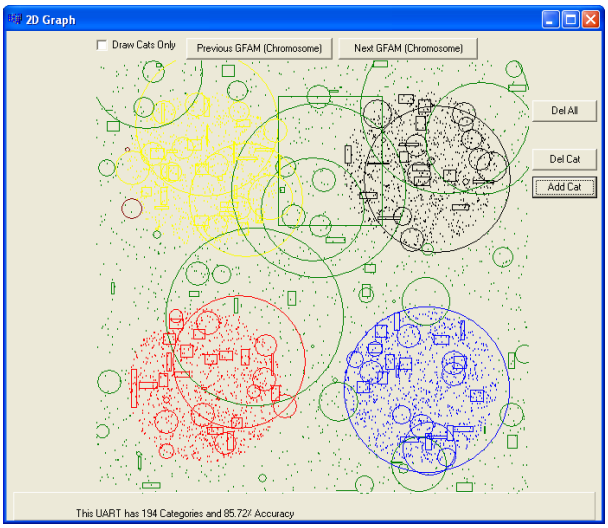

Figure 7-39: A UART network after randomly mixing FAM categories with EAM categories

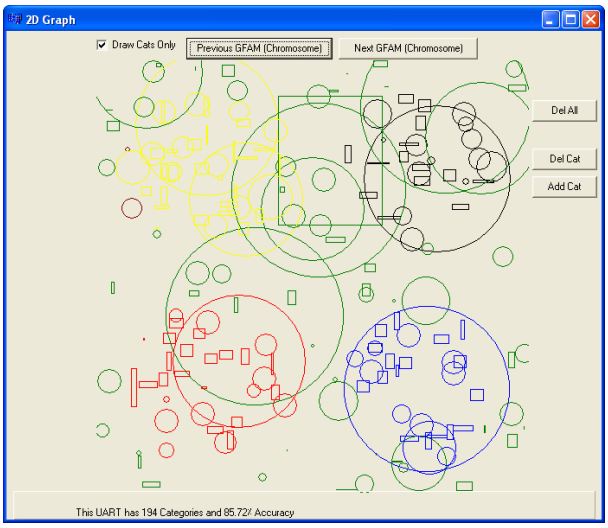

Figure 7-40: A 2D graph showing only the categories

- Get Boundaries (Button): This feature allows the user to visually see how the network is classifying input data. This feature works only for 2D problems (Figure 7-41 shows the outcome when this button is clicked). To achieve this goal, the program generates a list of 10000 patterns as a matrix with (0.01) increment on both x and y, and then feeds this list to the network and displays the way the network has classified these patterns by coloring them differently.

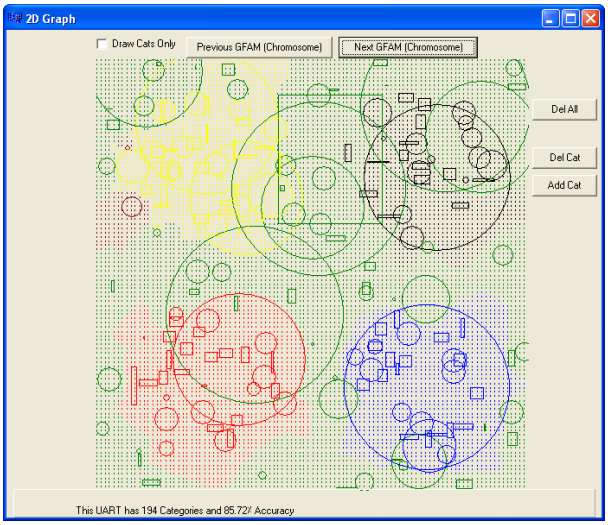

Figure 7-41: The classification borders of the UART network after the random mixing

- Clear UART (Button): Clicking this button clears the memory so that another problem can be checked.

All of the above were controls and their functionality on the main window. The graphing window (Figure 7-39) have some very interesting functionality, discussed below.

- Previous UART (Chromosome) (Button): Displays the contents of the previous network.
- Next UART (Chromosome) (Button): Displays the content of the next network.
- Add Cat: Brings up a dialogue box that has controls to allow the user to manually add a category to the current chromosome. This form uses the same controls to add a FAM category or an EAM category, based on the radio button at the bottom right corner of the form. Figure 7-43, explains this feature: after clicking the Del All button.

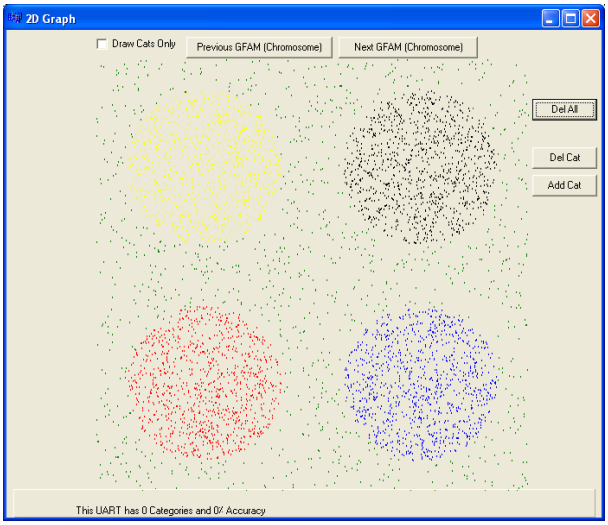

Figure 7-42: After deleting all the categories of UART

after clicking the Add Cat button

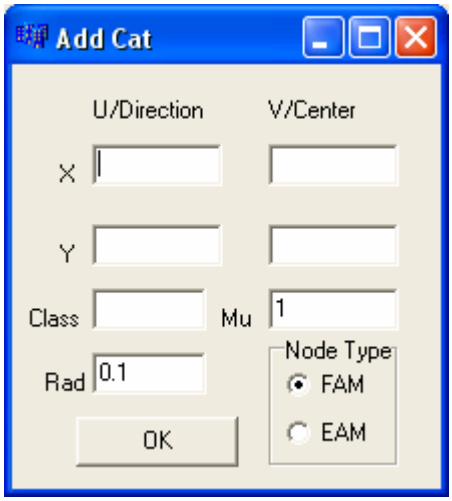

Figure 7-43: An add category dialogue box

where. U/Direction is the **u** vector of FAM and the direction vector of an EAM category, V/Center is the **v** vector of a FAM and the center vector of an EAM category. The Ok button adds the category to the chromosome:

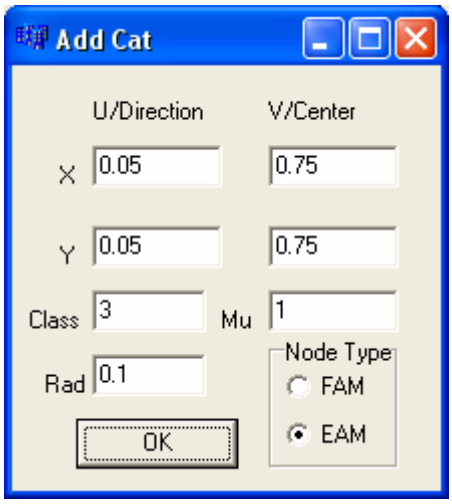

Figure 7-44: Filling in data for an EAM category

after clicking the ok button:

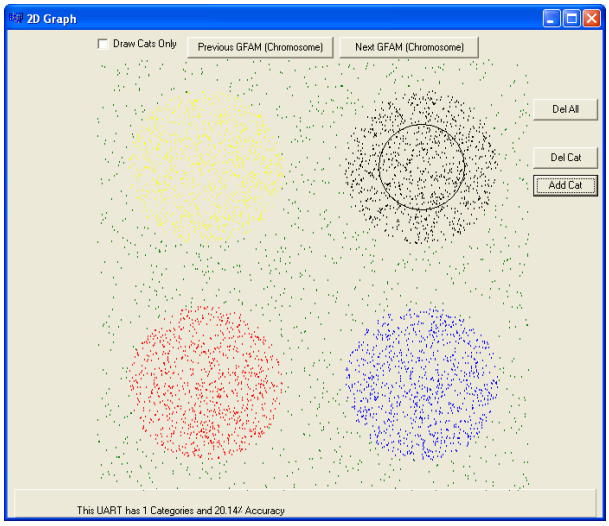

Figure 7-45: UART after manually adding an EAM category

After adding the above category, the classification boundaries look as shown in figure 7-46

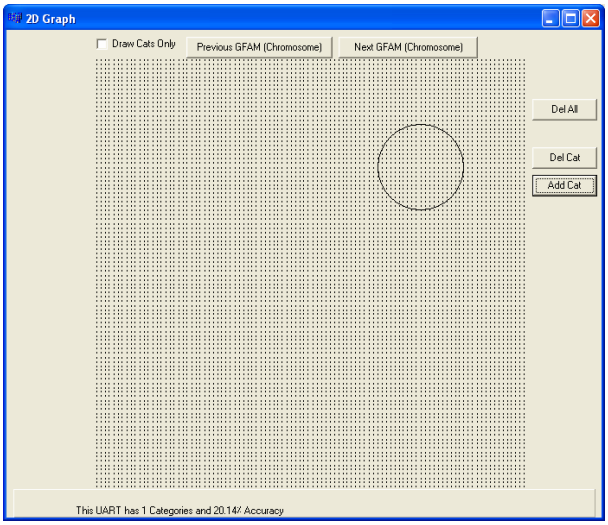

Figure 7-46: The classification boundaries of UART after addition

The accuracy here was 20% because we have five classes here, and the number of patterns is divided equally amongst them. After adding another category

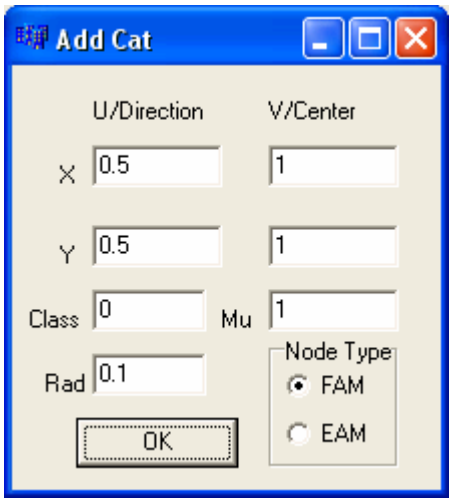

Figure 7-47: Filling in data for a FAM category

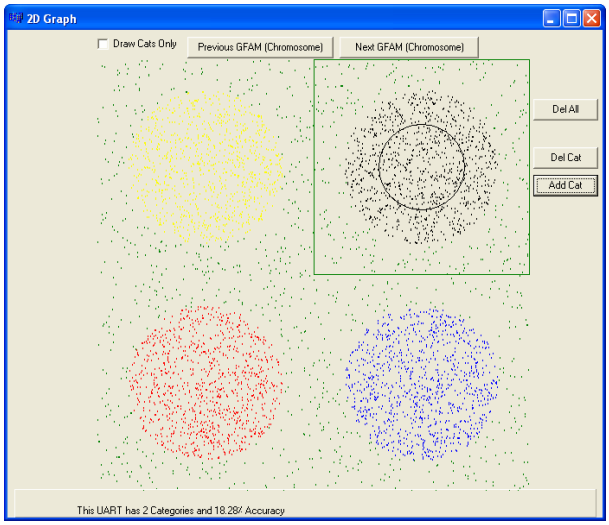

Figure 7-48: UART after manually adding an EAM and a FAM category

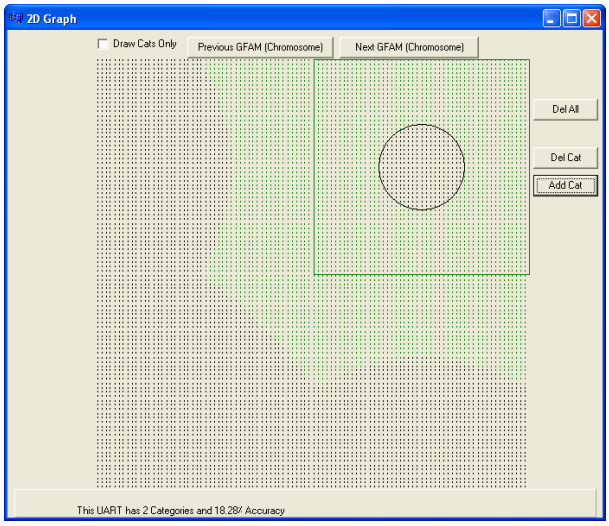

Figure 7-49: UART classification boundaries after the manual addition

7.4.2 UART UI Abstract Design

UART UI consists of 8 main objects, which are:

- UARTForm object: This is the main window object that shows most of the controls and has most of the user interaction.

- GraphForm object: This is the graph window, displays categories and classification boundaries, and allows manual deletion and addition of categories to any chromosome.
- FNode object: This object represents a category in FAM network.
- ENode object: This object represents a category in an EAM network.
- PtrnNode object: This object represents an input pattern.
- Chrom object: This object represents a chromosome, a collection of categories encoded in a special format.
- Cat object: This object represents a FAM category after encoding.
- AddCatForm object: This is the form that allows the user to insert the categories' properties.

Now we present these seven objects in more detail:

7.4.2.1 UARTForm Object

This is the main window object that has most of the controls listed in 7.1.1. It has many features and employs methods that allow it to do its functionality. This functionality is , briefly described in the following:

- Validates the inputs.
- Creates the trained FAMs and EAMs and stores them.
- Optimizes the generation of chromosomes to get the best UART.
- Displays the results and prepares the data for the graphing object.

The following is a list of the most important methods this object has:

- openFile: This method opens the files and reads the data, creates PtrnNode objects and stores them into vectors.

- initPopClick: This method creates FAM and EAM parameters and starts the loop of creating $Pop_{size}/2$ FAMs, then loops one more time to create $Pop_{size}/2$ EAM networks.
- buildFam: This method trains a FAM network.
- buildEam: This method trains an EAM network.
- crop: This method deletes all the categories that encoded one pattern only.
- redistributeCats: This function copies all the categories generated in the initial generation and stores them in a container, then, it deletes all the categories of all the chromosomes, shuffles the categories, and redistributes them to the chromosome as follows: If the total number of all categories is N^{all} then each chromosome gets $\frac{N^{all}}{Pop_{size}}$ categories, the first chromosome gets the first set of $\frac{N^{all}}{Pop_{size}}$ categories, the second chromosome gets the second set of $\frac{N^{all}}{Pop_{size}}$ categories, and so on. The last chromosome however, gets $\frac{N^{all}}{Pop_{size}} + N^{all} \bmod Pop_{size}$ categories.
- decode: This method takes a chromosome as an input and converts it to a UART network.
- optimize: This method optimizes a generation of trained UART networks.
- crossover: This function does the crossover process as described above.
- addCat: This method adds a category to a given chromosome, if a chromosome is chosen the added category have 50% probability of being a FAM category and 50% probability of being an EAM category.
- delCat: This method deletes a category from a given chromosome.
- mutation: This function mutates a given chromosome. If a chromosome is selected, every category in it will be mutated, if the category is a FAM category, the FAM

mutation procedure (described earlier) is applied, otherwise the EAM mutation procedure is applied.

- PrepBoundaries: This method creates a list of 10000 patterns as a grid in 2D. Their values are as follows: (0,0),(0,0.01)…(0,1),(0.01,0)…(0.01,1)…(1,1). It also initializes some other variables for the graphForm object to use.
- drawCats: This method initializes some variables and calls the graphForm object.
- showBoundaries: This method initializes some variables and calls the graphForm object.
- ~UARTForm: This is the destructor function.

this object also has many properties (Variables), here is a list of the most important ones:

- A vector of Node objects: This variable stores FNode and ENode objects as a UART network. Node object is the base class of both FNode and ENode objects.
- A vector of strings: This variable stores the names of the classes in a problem.
- A vector of PtrnNode objects: This variable stores the training patterns.
- A vector of PtrnNode objects: This variable stores the testing patterns.
- A vector of Chrom object pointers: This variable stores pointers to all the created chromosomes in specific generation.
- A vector of Chrom object pointers: This variable stores pointers to all the chromosomes in temp generation .
- A vector of integers: This variable stores the predicted labels after running the performance phase.
- An integer: This variable stores the number of created categories for a specific network.

These variables and many others are used in this object. However, since a detailed design of this UI is not the purpose of this dissertation, the details are omitted.

7.4.2.2 GraphForm Object

This object is much simpler than the UARTForm object. It is responsible for showing the categories, the validation patterns, and the classification boundaries of a specific chromosome. It is also responsible of allowing the user to manually add and delete categories from the chromosome. The following is a list of the main methods that this object employs and the variables that this object uses:

- pbPaint: This function displays the categories along with the validation patterns or the grid of automatically generated patterns along with the categories (classification boundaries). This function is one of the most difficult functions in this UI. Since we have to manually convert the origin of the graphing object such that the (0,0) point is the bottom left corner. This method has to differentiate between a FAM cat (displays it as a rectangle), and an EAM cat (displays it as an ellipsoid).
- nextClick: This functions displays the graph of the next chromosome.
- previousClick: This function displays the graph of the previous chromosome.
- addcClick: This function calls the AddCatForm object so that the user can insert the information of the new category.
- delAllCat: This functions invokes the del function of the Chrom object which deletes all the categories from that chromosome.
- ~GraphForm: This is the destructor function.

A few variables are used in this object. The following is a list of the main ones:

- An Integer c: This is the index of the current chromosome.
- An array of Point object pointers: This variable stores the points that are displayed in the graphing area.
- A pointer to UARTForm object: This object gives the GraphForm object access to many functions and variables from the main window.

- A boolean bounds: This variable tells the object if the categories are to be displayed or the classification boundaries are to be displayed.

7.4.2.3 FNode, ENode and PtrnNode Objects

FNode and PtrnNode objects are identical to those found in GFAM, while, ENode is identical to that of GEAM.

7.4.2.4 Chrom Object

This object represents a chromosome that is a collection of categories as shown in Figure 5-6. A chromosome object has the following functions:

- Chrom: This is the constructor of this object. It has a very important functionality that is converting the FNode object (FAM categories) and FNode objects into Cat objects that are suitable for genetic manipulation. There is another version of this function that creates an empty Chrom object.
- Del: This function deletes all the categories that it has. This operation is used when manual addition and deletion of the categories is invoked.
- ~Chrom: This is the destructor function.

And it has the following variables:

- A vector of Cat object pointers: This vector holds pointers to all Cat object this chromosome has.
- Double fit: This variable holds the fitness value of this chromosome.
- Double rho: This variable holds the vigilance parameter of the FAM network represented by this chromosome.
- Integer size: This is the number of categories in this chromosome.
- Double accuracy: This variable holds the accuracy of this network.

7.4.2.5 Cat Object

This is an object that resembles an encoded category. The category could be a FAM or an EAM category. This category is ready for genetic manipulations, and below is a list of its functions:

- Cat: This is the constructor. It takes as an input a FNode object or an ENode object and extracts the data from it, and saves it in its internal variables. Another version of it takes as an input a Cat object pointer; this version copies all the information of the original object to the newly created one. It also sets the ntype variable to 1, if the input category was an FNode, otherwise it sets it to 2.
- pointCat: This function returns a Boolean indicating whether or not this category is a point category.
- ~Cat: This is the destructor function.

The following variables exist in this object:

- Array of Doubles v: obvious.
- Array of Doubles u: obvious.
- Integer l: This is the class label.
- Integer ntype: This variable represents the type of the category (i.e. FAM or EAM, 1 for a FAM category, 2 for an EAM category).
- Integer s: This integer represents the number of features (problem dimensionality).
- Double mu: This variable is the axis ratio of an EAM category.
- Double r: This is the radius of an EAM category.

7.4.2.6 AddCatForm Object

This object is a small form that has ten controls to allow the user to insert values for manually added categories. It has one function other than constructor, that is, it returns a one if some values were inserted.

8. SUMMARY/CONTRIBUTIONS, AND FUTURE WORK

8.1 Summary/Contributions

In this dissertation we accomplished the following:

- Designed a methodology to create genetically engineered ART networks. We used this methodology to create genetically engineered FAMs, EAMs and GAMs, called GFAM, GEAM, GGAM.
- Experimented extensively with GFAM, GEAM, GGAM, and managed to create ART architectures that occasionally were optimal classifiers (achieved highest possible classification accuracy, while using the minimum possible number of categories). GFAM, GEAM, and GGAM compared favorably with other ART approaches that were introduced into the literature to solve the ART category proliferation problem.
- Extended the methodology to design genetically engineered FAMs, EAMs, and GAMs to the case where we created a genetically engineered combined FAM and EAM architecture, called UART. Once more UART's performance was compared with other ART architectures and it was also compared with GFAM and GEAM. The conclusion from this comparison was that UART has merit and should be investigated further.
- Produced analytical results that explained the order of search of FAM versus EAM categories in UART. This analysis helped us better understand how UART makes category choices.
- Calculated analytically the computational complexity of GFAM, GEAM, GGAM and UART. This analytical calculation demonstrated that the computational complexity of genetically engineered ART architectures compares very favorably compared to the complexity of other ART architectures introduced into the literature.

- Tested the genetic ART modules on a set of database that consists of real and artificial databases.
- In the process of running extensive experiments with the genetically engineered ART architectures we created four user-friendly interfaces UI GFAM, UI GEAM, UI GGAM, and finally UI UART. These interfaces not only allowed us to conduct this extensive experimentation effort in a timely manner, but it also allowed us to visualize the results, a feat that helps one to better understand the network's functionality.

8.2 Future Work

We see a number of directions that this research can be extended to, and they are briefly being discussed below.

- Choose the GA parameters in GEAM, and GGAM, and UART through experimentation. At this dissertation only the GFAM parameters were chosen through experimentation. The GEAM, GGAM and UART use as GA parameters the ones found from the GFAM experimentation with the GA parameters.
- Compare GFAM, and other genetically engineered ART networks with some of the state-of-the ART classifiers, such as support vector machines (SVMs). Preliminary results have shown that GFAM is very competitive, compared to the SVM approach.
- Extend the ideas presented in UART to other combinations of ART networks, such as FAM, GAM, or EAM, GAM, and finally FAM, EAM, GAM.
- Extend the analysis of the order of search of FAM versus EAM categories to orders of search of FAM versus GAM categories, EAM versus GAM categories, and finally FAM, versus EAM, versus GAM categories.
- Conduct a study to uncover the effect of using genetic algorithms on the size of the training and validation sets needed for the accurate training of ART architectures.

APPENDIX A: TERMINOLOGY

- FAM: Fuzzy ARTMAP.
- EAM: Ellipsoidal ARTMAP.
- GAM: Gaussian ARTMAP.
- ssFAM, ssEAM, ssGAM: Semi-suprvised Fuzzy ARTMAP, Semi-supervised Ellipsoidal ARTMAP, Semi-supervised Gaussian ARTMAP.
- M_a : The dimensionality of the input patterns in the training, validation and test sets provided to us by the classification problem under consideration.
- *Training Set*: The collection of input/output pairs used in the training of FAMs that constitute the initial FAM population in GFAM.
- *Validation Set*: The collection of input/output pairs used to validate the performance of the FAM networks during the evolution of FAMs from generation to generation.
- *Test Set:* The collection of input/output pairs used to assess the performance of the chosen FAM network, after the evolution of FAMs is completed.
- PT : Number of points in the training set.
- PV : Number of points in the validation set.
- $\overline{\rho}_a^{\min}$: This is the lower limit of the baseline vigilance parameter used in the training of the FAM networks that comprise the initial population of the FAM networks.
- $\overline{\rho}_a^{\max}$: This is the upper limit of the baseline vigilance parameter used in the training of the FAM networks that comprise the initial population of the FAM networks.
- β_a : The choice parameter used in the training of the FAM networks that comprise the initial population of the FAM networks. This parameter is fixed, and chosen equal to 1.0.
- Pop_{size} : The number of chromosomes (FAM trained networks) in each generation.

- $N_a(p)$: The number of categories in the p^{th} FAM network from the Pop_{size} trained FAM networks in a generation.
- $\mathbf{w}_j^a(p) = (\mathbf{u}_j^a(p), (\mathbf{v}_j^a(p))^c)$: the weight vector corresponding to category *j* of the p^{th} FAM network from the Pop_{size} trained FAM networks in a generation; $\mathbf{u}_j^a(p)$ corresponds to the lower endpoint of the hyperbox that the weight vector $\mathbf{w}_j^a(p)$ defines and $\mathbf{v}_j^a(p)$ corresponds to the upper endpoint of this hyperbox.
- $l_j(p)$: The label of category *j* of the p^{th} FAM network from the Pop_{size} trained FAM networks in a generation.
- $PCC(p)$: The percentage of correct classification on the validation set exhibited by the p^{th} FAM network from the Pop_{size} trained FAM networks in a generation.
- $Gen_{\max}$: The maximum number of generations allowed for the FAM networks to evolve. When this maximum number is reached evolution stops and the FAM with the best fitness value on the validation set is reported.
- NC_{best} : Number of best chromosomes that the GFAM transfers from the old generation to the new generation (elitism).
- $Cat_{\min}$, $Cat_{\max}$: The minimum and the maximum number of categories that a chromosome is allowed to have during the evolutionary process that GFAM undergoes.
- Cat_{add}, Cat_{del} : New genetic operators that add and delete a category from a chromosome.
- $P(Cat_{add})$, $P(Cat_{del})$, $P(Mut)$: The probabilities of adding, deleting and mutating a category.
- PT : Number of points in the training set.

- PV : Number of data-points in the validation set.
- $PTes$: Number of points in the test set.
- PS : Number of network parameter settings to produce the best ART network (ART is ssFAM, ssEAM, ssGAM and safe micro-ARTMAP).
- $\mathbf{I}^r$: Input pattern from your training or test collection.
- $\mathbf{O}^r$: Output pattern of your training or test collection. Output pattern $\mathbf{O}^r$ corresponds to input pattern $\mathbf{I}^r$.
- $S(\mathbf{I}^r)$: The set of committed nodes in the F_2^a layer of GAM during the presentation of the input/output pair $(\mathbf{I}^r, \mathbf{O}^r)$.
- $S_C(\mathbf{I}^r)$: The set of committed nodes in the F_2^a layer of GAM during the presentation of the input/output pair $(\mathbf{I}^r, \mathbf{O}^r)$ that are still competing to represent the input pattern $\mathbf{I}^r$. The set $S_C(\mathbf{I}^r)$ is a subset of $S(\mathbf{I}^r)$ consisting of nodes that pass the vigilance threshold and they have not been deactivated during to match-tracking.
- $\overline{\rho}_a$: Baseline vigilance parameter value. This value determines of how high the level of match between input pattern and F_2^a node template should be for this template to be an acceptable candidate of encoding the input pattern.
- ρ_a : Vigilance parameter. It starts at the value of $\overline{\rho}_a$ and then as training of an input/output pair progresses it might change (maybe increase) its value due to match-tracking.
- *Epsilon*: A small positive constant that designates of how much the vigilance parameter value will increase beyond the weighted vigilance value of the deactivated nodes after match tracking is enforced.

- $\mathbf{w}_j$: Template of node *j* in the F_2^a layer of dFAM. A template $\mathbf{w}_j$ is a vector comprised of two vectors: (a) the vector $\boldsymbol{\mu}_j$ that is the mean of all the input patterns that chose node *j* as their representative node, and were coded by this node, and (b) the vector $\boldsymbol{\sigma}_j$ that is the standard deviation vector corresponding to of all the input patterns that chose node *j* as their representative node, and were coded by this node.
- $\rho(\mathbf{I}^r \mid \mathbf{w}_j)$: Match function value of input pattern $\mathbf{I}^r$ and node *j* with template $\mathbf{w}_j$.
- $T_j(\mathbf{I}^r \mid \mathbf{w}_j)$: Choice function (bottom-up input) value of node *j* when the dFAM input is input pattern $\mathbf{I}^r$.
- $\mathbf{W}_j^{ab}$ **:** The inter-ART weight vector from node *j* in F_2^a to all the nodes in F_2^b . When node *j* is first committed by (say) input pattern $\mathbf{I}^r$, this vector becomes equal to $\mathbf{O}^r$, and its value does not change any more during training. In essence, only one of the components of $\mathbf{W}_j^{ab}$ is 1, and the rest of the its components are 0. The component of $\mathbf{W}_j^{ab}$ that is 1 identifies for us the label (output pattern) that node *j* is mapped to.
- $\mathbf{v}_j$: A vector whose components correspond to the second moment of the components of the input patterns that chose node j as their representative node and were encoded by node *j*.
- γ : The initial value of the standard deviation that $\boldsymbol{\sigma}_j$ is equal to at the time that node *j* has encoded a single input pattern.
- n_j : This parameter is equal to the number of times that node *j* was activated by an input pattern, and node *j* coded this input pattern.

APPENDIX B: FAM STEP-BY-STEP TRAINING & TESTING

Training Phase of FAM

The step-by-step implementation of the off-line training in FAM is presented below:

Step 1: Set the baseline vigilance parameter $\overline{\rho}_a$ to a value from the interval [0, 1]. Also, initialize the parameter α. The weight values corresponding to a node *j* in F_2^a are: $\mathbf{w}_j$ (a weight vector equal to $(\mathbf{u}_j, (\mathbf{v}_j)^c)$, where $\mathbf{u}_j$ corresponds to the vector of the minima of the components of patterns that chose this node as their representative while $\mathbf{v}_j$ corresponds to the vector of the maxima of the components of patterns that chose this node as their representative node, and the inter-ART weights $\mathbf{W}_j^{ab}$ (with components W_{jk}^{ab}; $j = 1,..., N_a$, $k = 1,..., N_b$). As training progresses every vector $\mathbf{W}_j^{ab}$ that has been committed has one of its components equal to 1 and the other components equal to zero. The component of $\mathbf{W}_j^{ab}$ that is equal to 1 designates the label that node *j* is mapped to. The number of nodes in the F_1^a layer is denoted by M_a. The number of committed nodes in the F_2^a layer is denoted by N_a. The number of nodes in the F_2^b layer (denoted by N_b) corresponds to the number of output classes. If there are 4 output classes the number of nodes in the F_2^b output layer is 4. The index *r* of the input/output pairs is set to 1. The set $S(\mathbf{I}^r)$ represents the set of committed nodes in FAM, and is initially set to be equal to the empty set.

Step 2: Present the r^{th} input pattern $(\mathbf{I}^r, \mathbf{O}^r)$ to FAM. That is, the input pattern $\mathbf{I}^r$ is presented at the input layer F_1^a and the output $\mathbf{O}^r$ is presented at the output layer F_2^b. The vigilance parameter ρ_a is set to the baseline vigilance value $\overline{\rho}_a$.

Step 3: $S(\mathbf{I}^r)$ designates the set of all the committed nodes in F_2^a. If the set $S(\mathbf{I}^r)$ is the empty set (i.e., there are no committed nodes) go to Step 7a to do learning of the input/output

pair by an uncommitted node. Otherwise, calculate the match function for all the committed nodes *j* in $S(\mathbf{I}^r)$. The match function $\rho(\mathbf{I}^r \mid \mathbf{w}_j)$ is calculated as follows:

$$\rho(\mathbf{I}^r \mid \mathbf{w}_j) = \frac{|\mathbf{I}^r \wedge \mathbf{w}_j|}{|\mathbf{I}^r|} = \frac{|\mathbf{I}^r \wedge \mathbf{w}_j|}{M_a}$$

Step 4: Find from the set $S(\mathbf{I}^r)$, a subset of nodes, designated as $S_C(\mathbf{I}^r)$, which represents the subset of nodes in the set $S(\mathbf{I}^r)$ whose match function exceeds the vigilance ρ_a. If the set $S_C(\mathbf{I}^r)$ is the empty set you go to Step 7a to do learning of the input/output pair by an uncommitted node. Otherwise, for every node $j \in S_C(\mathbf{I}^r)$ we calculate the choice function (if it has not been calculated before), as follows:

$$T_j(\mathbf{I}^r) = \frac{|\mathbf{I}^r \wedge \mathbf{w}_j|}{\alpha + |\mathbf{w}_j|}$$

Step 5: Find the node J that has the maximum choice function (bottom-up input) value. That is, find

$$J = \max_j \{T_j(\mathbf{I}^r)\}$$

Note: If there are more than one node indices that maximize the choice function choose the lowest index.

Step 6: Find the prediction of the activated node *J*. The prediction *K* of the input pattern $\mathbf{I}^r$, is the node *K* for which $W_{JK}^{ab} = 1$.

We now distinguish two cases.

If the label *K* is the correct label, then we move to Step 7b to do learning.

If label *K* is the incorrect label

$$\rho_a = \rho(\mathbf{I}^r \mid \mathbf{w}_J) + epsilon$$

where *epsilon* is a very small positive value and we also redefine the set $S(\mathbf{I}^r)$, as follows:

$S(\mathbf{I}^r) = S(\mathbf{I}^r) - J$

and then we go back to Step 4.

Step 7: We now distinguish two cases.

An uncommitted node (say node *J*; *J* is always one index higher than the highest index of a committed node) is chosen to learn the input/output pair. Then,

$\mathbf{w}_J = \mathbf{I}^r$

$\mathbf{W}_J^{ab} = \mathbf{O}^r$

$S(\mathbf{I}^{r+1})$ = set of all committed nodes in F_2^a

(**Note:** if *r* is the last index *r*+1 is the first index) Go to Step 8.

A committed node *J* is chosen to learn the input/output pair; then for this node *J* we have

$\mathbf{w}_J = \mathbf{I}^r \wedge \mathbf{w}_J$

$S(\mathbf{I}^{r+1})$ = set of all committed nodes in F_2^a

(**Note:** if *r* is the last index *r*+1 is the first index) Go to Step 8.

Step 8: If we are not at the end of an epoch (i.e., one complete presentation of all the input/output pairs in the training set), *r* is incremented to *r+1* and we go back to Step 2 to present the $r+1^{\text{th}}$ input/output pair. If we are at the end of an epoch (i.e., all input/output pairs in the training set have been presented once) then two cases can be distinguished.

In the previous list presentation at least one component of the top-down weights or the inter-ART weights has been changed and we have not reached the maximum number of list presentations allowed. In this case we go back to Step 2 and present the first input/output pair in the set of input/output pairs, by setting *r* to 1.

In the previous list presentation no weight changes occurred in the top-down weights and the inter-ART weights or we have reached the maximum number of list presentations

allowed. Hence training is considered to be complete and the network is considered to have learnt the training patterns perfectly.

Performance Phase of FAM

The step-by-step implementation of FAM's performance phase is described below:

Step 1: Initialize the weights $\mathbf{w}_j$; $j = 1,...N_a$, W_{jk}^{ab}; $j = 1,..., N_a$, $k = 1,..., N_b$, to the values that they had at the end of the training phase of FAM.

Step 2: Present the r^{th} input pattern $(\mathbf{I}^r, \mathbf{O}^r)$ to FAM. That is, the input pattern $\mathbf{I}^r$ is presented at the input layer F_1^a and the output $\mathbf{O}^r$ is presented at the output layer F_2^b. The vigilance parameter ρ_a is set to the baseline vigilance value $\overline{\rho}_a$.

Step 3: For every node j we calculate the choice function, as follows:

$$T_j(\mathbf{I}^r) = \frac{|\mathbf{I}^r \wedge \mathbf{w}_j|}{\alpha + |\mathbf{w}_j|}$$

Step 4: Find the node J that has the maximum choice function (bottom-up input) value. That is find

$$J = \max_j \{T_j(\mathbf{I}^r)\}$$

Note: If there are more than one node indices that maximize the choice function choose the lowest index.

Step 5: Find the prediction of the activated node *J*. The prediction K of the input pattern $\mathbf{I}^r$, is the node *K* for which $W_{JK}^{ab} = 1$.

Step 6: If all the test patterns in the test set have not been applied to the network then go back to Step 2 and present the next input/output test pair in the sequence. If we have presented all the input/output test pairs then the results can be analyzed to find the misclassification error and other such statistics.

APPENDIX C: EAM STEP-BY-STEP TRAINING & TESTING

Training Phase of EAM

The step-by-step implementation of the off-line training in EAM is presented below:

Step 1: The weight values corresponding to a node *j* in F_2^a are: $\mathbf{m}_j$ (the center of the ellipsoid corresponding to node *j*), $\mathbf{d}_j$ (the direction vector of the major axis of the ellipsoid corresponding to node *j*) , and R_j (half of the length of the major axis of the ellipsoid corresponding to the node *j*); all these weight values are represented by the generic vector $\mathbf{w}_j$. We also have inter-ART weights $\mathbf{W}_j^{ab}$ (with components W_{jk}^{ab}; $j = 1,..., N_a$, $k = 1,..., N_b$). As training progresses every vector $\mathbf{W}_j^{ab}$ that has been committed has one of its components equal to 1 and the other components equal to zero. The component of $\mathbf{W}_j^{ab}$ that is equal to 1 designates the label that node *j* is mapped to. The number of nodes in the F_1^a layer is denoted by M_a . The number of committed nodes in the F_2^a layer is denoted by N_a . The number of nodes in the F_2^b layer (denoted by N_b) corresponds to the number of output classes. If there are 4 output classes the number of nodes in the F_2^b output layer is 4. The index *r* of the input/output pairs is set to 1. The set $S(\mathbf{I}^r)$ represents the set of committed nodes in FAM, and is initially set to be equal to the empty set. Set the baseline vigilance parameter $\overline{\rho}_a$ to a value from the interval [0, 1]. Set the parameter μ to a value from the interval (0, 1]. Actually use a default value for μ equal to 0.5. Set the parameter value α to a value between $(0, \infty)$; typical values for α are small positive constants. Also, set the parameter $D = \dfrac{\sqrt{M_a}}{\mu}$ (in UART $D = M_a$). You need to normalize your data so that they all have component values in the interval [0, 1]. For EAM you do no need to complement encode your inputs.

Step 2: Present the r^{th} input pattern $(\mathbf{I}^r, \mathbf{O}^r)$ to EAM. That is, the input pattern $\mathbf{I}^r$ is presented at the input layer F_1^a and the output $\mathbf{O}^r$ is presented at the output layer F_2^b. The vigilance parameter ρ_a is set to the baseline vigilance value $\overline{\rho}_a$.

Step 3: $S(\mathbf{I}^r)$ designates the set of all the committed nodes in F_2^a. If the set $S(\mathbf{I}^r)$ is the empty set (i.e., there are no committed nodes) go to Step 7a to do learning of the input/output pair by an uncommitted node. Otherwise, calculate the match function for all the committed nodes j in $S(\mathbf{I}^r)$. The match function $\rho(\mathbf{I}^r \mid \mathbf{w}_j)$ is calculated as follows:

$$\rho(\mathbf{I}^r \mid \mathbf{w}_j) = \frac{D - R_j - \max\{R_j, \| \mathbf{I}^r - \mathbf{m}_j \|_{\mathbf{C}_j}\}}{D}$$

where

$$\| \mathbf{I}^r - \mathbf{m}_j \|_{\mathbf{C}j} = \frac{1}{\mu} \sqrt{\| \mathbf{I}^r - \mathbf{m}_j \|_2^2 - (1-\mu^2)\,(\mathbf{d}_j^T\,(\mathbf{I}^r - \mathbf{m}_j))^2}$$

and $\| \mathbf{I}^r - \mathbf{m}_j \|_2^2$ stands for the square of the Euclidean distance of $\mathbf{I}^r$ and $\mathbf{m}_j$.

Step 4: Find from the set $S(\mathbf{I}^r)$, a subset of nodes, designated as $S_C(\mathbf{I}^r)$, which represents the subset of nodes in the set $S(\mathbf{I}^r)$ whose match function exceeds the vigilance ρ_a. If the set $S_C(\mathbf{I}^r)$ is the empty set you go to Step 7a to do learning of the input/output pair by an uncommitted node. Otherwise, for every node $j \in S_C(\mathbf{I}^r)$ we calculate the choice function (if it has not been calculated before), as follows:

$$T_j(\mathbf{I}^r) = \frac{D - R_j - \max\{R_j, \| \mathbf{I}^r - \mathbf{m}_j \|_{\mathbf{C}_j}\}}{D - 2R_j + \alpha}$$

Step 5: Find the node J that has the maximum choice function (bottom-up input) value. That is find

$$J = \max_j \{T_j(\mathbf{I}^r)$$

Note: If there are more than one node indices that maximize the choice function choose the lowest index.

Step 6: Find the prediction of the activated node *J*. The prediction K of the input pattern $\mathbf{I}^r$, is the node *K* for which $W_{JK}^{ab} = 1$.

We now distinguish two cases.

If the label *K* is the correct label, then we move to Step 7b to do learning.

If label *K* is the incorrect label

$$\rho_a = \rho(\mathbf{I}^r \mid \mathbf{w}_J) + epsilon$$

where *epsilon* is a very small positive value and we also redefine the set $S(\mathbf{I}^r)$, as follows:

$$S(\mathbf{I}^r) = S(\mathbf{I}^r) - J$$

and then we go back to Step 4.

Step 7: We now distinguish two cases.

An uncommitted node (say node *J*; *J* is always one index higher than the highest index of a committed node) is chosen to learn the input/output pair. Then,

$$\mathbf{m}_J = \mathbf{I}^r$$

$$\mathbf{d}_J = \mathbf{0}$$

$$r_J = 0$$

$$\mathbf{W}_J^{ab} = \mathbf{O}^r$$

$S(\mathbf{I}^{r+1})$ = set of all committed nodes in F_2^a

(**Note:** if *r* is the last index *r*+1 is the first index) Go to Step 8.

A committed node *J* is chosen to learn the input/output pair; then the network weights are updated as follows :

$$\mathbf{m}_J = \mathbf{m}_J^{old} + \frac{1}{2}\left(\frac{\max\{\| \mathbf{I}^r - \mathbf{m}_J^{old} \|_{\mathbf{C}J}^{old} - R_J^{old}\} - R_J^{old}}{\| \mathbf{I}^r - \mathbf{m}_J^{old} \|_{\mathbf{C}J}^{old}}\right)(\mathbf{I}^r - \mathbf{m}_J^{old})$$

and for the case where $\mathbf{I}^r$ is not the second pattern encoded by node J we have:

$$\mathbf{d}_J = \mathbf{d}_J$$

$$R_J = R_J^{old} + \frac{1}{2}\left(\max\{\| \mathbf{I}^r - \mathbf{m}_J^{old} \|_{C_J^{old}}, R_J^{old}\} - R_J^{old}\right)$$

while for the case where $\mathbf{I}^r$ is the second pattern encoded by node *J* we have:

$$\mathbf{d}_J = \frac{\mathbf{I}^r - \mathbf{m}_J^{old}}{\| \mathbf{I}^r - \mathbf{m}_J^{old} \|}$$

$$R_J = \frac{\| \mathbf{I}^r - \mathbf{m}_J^{old} \|}{2}$$

$S(\mathbf{I}^{r+1})$ = set of all committed nodes in F_2^a

(**Note:** if *r* is the last index *r*+1 is the first index) Go to Step 8.

Step 8: If we are not at the end of an epoch (i.e., one complete presentation of all the input/output pairs in the training set), *r* is incremented to *r+1* and we go back to Step 2 to present the $r+1^{\text{th}}$ input/output pair. If we are at the end of an epoch (i.e., all input/output pairs in the training set have been presented once) then two cases can be distinguished.

In the previous list presentation at least one component of the top-down weights or the inter-ART weights has been changed and we have not reached the maximum number of list presentations allowed. In this case we go back to Step 2 and present the first input/output pair in the set of input/output pairs, by setting *r* to 1.

In the previous list presentation no weight changes occurred in the top-down weights and the inter-ART weights or we have reached the maximum number of list presentations allowed. Hence training is considered to be complete and the network is considered to have learnt the training patterns perfectly.

Performance Phase of EAM

The step-by-step implementation of EAM's performance phase is described below:

Step 1: Initialize the weights $\mathbf{m}_j$, $\mathbf{d}_j$, R_j, W_{jk}^{ab}, to the values that they had at the end of the training phase of EAM. Initialize also all the other EAM parameters (baseline vigilance, choice parameter, μ, etc…)

Step 2: Present the r^{th} input pattern $(\mathbf{I}^r, \mathbf{O}^r)$ to EAM. That is, the input pattern $\mathbf{I}^r$ is presented at the input layer F_1^a and the output $\mathbf{O}^r$ is presented at the output layer F_2^b. The vigilance parameter ρ_a is set to the baseline vigilance value $\overline{\rho}_a$.

Step 3: For every node j we calculate the choice function, as follows:

$$T_j(\mathbf{I}^r) = \frac{D - R_j - \max\{R_j, \| \mathbf{I}^r - \mathbf{m}_j \|_{\mathbf{C}_j}\}}{D - 2R_j + \alpha}$$

Step 4: Find the node J that has the maximum choice function (bottom-up input) value. That is find

$$J = \max_j \{T_j(\mathbf{I}^r)\}$$

Note: If there are more than one node indices that maximize the choice function choose the lowest index.

Step 5: Find the prediction of the activated node *J*. The prediction K of the input pattern $\mathbf{I}^r$, is the node *K* for which $W_{JK}^{ab} = 1$.

Step 6: If all the test patterns in the test set have not been applied to the network then go back to Step 2 and present the next input/output test pair in the sequence. If we have presented all the input/output test pairs then the results can be analyzed to find the misclassification error and other such statistics.

APPENDIX D: GAM STEP-BY-STEP TRAINING & TESTING

Training Phase of GAM

The step-by-step implementation of the off-line training in GAM is presented below:

Step 1: Set the baseline vigilance parameter $\overline{\rho}_a$ to a value from the interval [0, 1]. Also, initialize the parameter γ. The weight values corresponding to a node *j* in F_2^a are: $\boldsymbol{\mu}_j$ (mean of the data that have activated and were encoded by node *j*), $\boldsymbol{\sigma}_j$ (the standard deviation vector of the data that have activated and were encoded by node *j*), n_j (the number of training input patterns that were encoded by node *j* in F_2^a), and the inter-ART weights $\mathbf{W}_j^{ab}$ (with components W_{jk}^{ab}; $j = 1,..., N_a$, $k = 1,..., N_b$). As training progresses every vector $\mathbf{W}_j^{ab}$ that has been committed has one of its components equal to 1 and the other components equal to zero. The component of $\mathbf{W}_j^{ab}$ that is equal to 1 designates the label that node *j* is mapped to. The number of nodes in the F_1^a layer is denoted by M_a. The number of committed nodes in the F_2^a layer is denoted by N_a. The number of nodes in the F_2^b layer (denoted by N_b) corresponds to the number of output classes. If there are 4 output classes the number of nodes in the F_2^b output layer is 4. The index *r* of the input/output pairs is set to 1. The set $S(\mathbf{I}^r)$ represents the set of committed nodes in GAM, and is initially set to be equal to the empty set.

Note: There is another parameter vector $\mathbf{v}_j$ for every committed node *j* in F_2^a that we need to keep track of. The vector $\mathbf{v}_j$ is the vector with components the experimental mean of the squared values of the input patterns that choose node *j* as their representative node. The relationships are:

$\sigma_{ji} = \sqrt{v_{ji} - \mu_{ji}^2}$ and $v_{ji} = \sigma_{ji}^2 + \mu_{ji}^2$ for every component *i* of vectors $\mathbf{v}_j, \boldsymbol{\sigma}_j, \boldsymbol{\mu}_j$

Step 2: Present the r^{th} input pattern $(\mathbf{I}^r, \mathbf{O}^r)$ to GAM or GAM. That is, the input pattern $\mathbf{I}^r$ is presented at the input layer F_1^a and the output $\mathbf{O}^r$ is presented at the output layer F_2^b. The vigilance parameter ρ_a is set to the baseline vigilance value $\overline{\rho}_a$.

Step 3: $S(\mathbf{I}^r)$ designates the set of all the committed nodes in F_2^a. If the set $S(\mathbf{I}^r)$ is the empty set (i.e., there are no committed nodes) go to Step 7a to do learning of the input/output pair by an uncommitted node. Otherwise, calculate the match function for all the committed nodes *j* in $S(\mathbf{I}^r)$. The match function $\rho(\mathbf{I}^r \mid \mathbf{w}_j)$ is calculated as follows:

$$\rho(\mathbf{I}^r \mid \mathbf{w}_j) = G(j \mid \mathbf{I}^r) = \exp\left(-\frac{1}{2} \sum_{i=1}^{M_a} \left(\frac{I_i^r - \mu_{ji}}{\sigma_{ji}} \right)^2 \right)$$

Note: It might be more computationally attractive to calculate the logarithm of the match function, as follows:

$$\log_e(\rho(\mathbf{I}^r \mid \mathbf{w}_j) = \log_e(G(j \mid \mathbf{I}^r)) = -\frac{1}{2} \sum_{i=1}^{M_a} \left(\frac{I_i^r - \mu_{ji}}{\sigma_{ji}} \right)^2$$

When we use the logarithm of the match function, we need to be comparing $\log_e(\rho(\mathbf{I}^r \mid \mathbf{w}_j)$ of every node with $\log_e(\rho_a)$ to determine if a node's match function value exceeds the vigilance threshold (see beginning of Step 4).

Step 4: Find from the set $S(\mathbf{I}^r)$, a subset of nodes, designated as $S_C(\mathbf{I}^r)$, which represents the subset of nodes in the set $S(\mathbf{I}^r)$ whose match function exceeds the vigilance ρ_a. If the set $S_C(\mathbf{I}^r)$ is the empty set you go to Step 7a to do learning of the input/output pair by an uncommitted node. Otherwise, for every node $j \in S_C(\mathbf{I}^r)$ we calculate the choice function (if it has not been calculated before), as follows:

$$T_j(\mathbf{I}^r \mid \mathbf{w}_j) = g(j \mid \mathbf{I}^r) = \frac{n_j / (\sum_{l=1}^{N_a} n_l)}{\prod_{i=1}^{M_a} \sigma_{ji}} G(j \mid \mathbf{I}^r)$$

Note: In the calculation of $T_j(\mathbf{I}^r \mid \mathbf{w}_j)$ we have omitted a term in the denominator. This term is equal to $(2\pi)^{M_a/2}$. So the correct expression for the calculation of $T_j(\mathbf{I}^r \mid \mathbf{w}_j)$ is as follows:

$$T_j(\mathbf{I}^r \mid \mathbf{w}_j) = g(j \mid \mathbf{I}^r) = \frac{n_j / (\sum_{l=1}^{N_a} n_l)}{\prod_{i=1}^{M_a} \sigma_{ji}} \frac{1}{(2\pi)^{\frac{M_a}{2}}} G(j \mid \mathbf{I}^r)$$

In the expression above, the terms

$$\frac{1}{\prod_{i=1}^{M_a} \sigma_{ji}} \frac{1}{(2\pi)^{\frac{M_a}{2}}} G(j \mid \mathbf{I}^r) \text{ and } \quad \frac{n_j}{\sum_{l=1}^{N_a} n_l},$$

represent the class conditional probabilities of the input pattern belonging to class *j* , and the a-priori estimate of the probability that a pattern belongs to class *j*, respectively. But note that since we are using the *T* (or *g*) values in order to calculate normalized activations, the common terms in the evaluation of *T* (or *g*) can be ignored. These terms are: $(2\pi)^{M_a/2}$ and $\sum_{l=1}^{N_a} n_l$. In our calculation of *T* (or *g*) we have ignored the term $(2\pi)^{M_a/2}$, but we have included the term $\sum_{l=1}^{N_a} n_l$.

Step 5: Find the node J that has the maximum choice function (bottom-up input) value. That is find

$$J = \max_j \{T_j(\mathbf{I}^r)$$

Note: If there are more than one node indices that maximize the choice function choose the lowest index.

Step 6: Find the prediction of the activated node *J*. The prediction *K* of the input pattern $\mathbf{I}^r$, is the node *K* for which $W_{JK}^{ab} = 1$.

We now distinguish two cases.

If the label *K* is the correct label, then we move to Step 7b to do learning.

If label *K* is the incorrect label

$\rho_a = \rho(\mathbf{I}^r \mid \mathbf{w}_J) + epsilon$

where *epsilon* is a very small positive value and we also redefine the set $S(\mathbf{I}^r)$, as follows:

$S(\mathbf{I}^r) = S(\mathbf{I}^r) - J$

and then we go back to Step 4.

Step 7: We now distinguish two cases.

An uncommitted node (say node *J*; *J* is always one index higher than the highest index of a committed node) is chosen to learn the input/output pair. Then,

$n_J = 1$

$\boldsymbol{\mu}_J = \mathbf{I}^r$

$\mathbf{v}_J = (\mathbf{I}^r)^2 + \gamma^2$

$\boldsymbol{\sigma}_J = \gamma$

$\mathbf{W}_J^{ab} = \mathbf{O}^r$

$S(\mathbf{I}^{r+1})$ = set of all committed nodes in F_2^a

(**Note:** if *r* is the last index *r*+1 is the first index) Go to Step 8.

A committed node *J* is chosen to learn the input/output pair; then for this node *J* we have

$n_J = n_J + 1$

$$\boldsymbol{\mu}_J = \left(1 - \frac{1}{n_J}\right)\boldsymbol{\mu}_J + \frac{1}{n_J}\mathbf{I}^r$$

$$v_{Ji} = (1-(n_j)^{-1})v_{Ji} + (n_J)^{-1}(I_i^r)^2$$

$$\sigma_{Ji} = \sqrt{v_{Ji} - \mu_{Ji}^2}$$

$S(\mathbf{I}^{r+1})$ = set of all committed nodes in F_2^a

(**Note:** if *r* is the last index *r*+1 is the first index) Go to Step 8.

Step 8: If we are not at the end of an epoch (i.e., one complete presentation of all the input/output pairs in the training set), *r* is incremented to *r+1* and we go back to Step 2 to present the $r+1^{\text{th}}$ input/output pair. If we are at the end of an epoch (i.e., all input/output pairs in the training set have been presented once) then two cases can be distinguished.

In the previous list presentation at least one component of the top-down weights or the inter-ART weights has been changed and we have not reached the maximum number of list presentations allowed. In this case we go back to Step 2 and present the first input/output pair in the set of input/output pairs, by setting *r* to 1.

In the previous list presentation no weight changes occurred in the top-down weights and the inter-ART weights or we have reached the maximum number of list presentations allowed. Hence training is considered to be complete and the network is considered to have learnt the training patterns perfectly.

Performance Phase of GAM

The step-by-step implementation of GAM's performance phase is described below:

Step 1: Initialize the weights $\boldsymbol{\mu}_j$, $\boldsymbol{\sigma}_j$, n_j; $j = 1,...N_a$, W_{jk}^{ab}; $j = 1,..., N_a$, $k = 1,..., N_b$, to the values that they had at the end of the training phase of Gaussian ARTMAP.

Step 2: Present the r^{th} input pattern $(\mathbf{I}^r, \mathbf{O}^r)$ to GAM. That is, the input pattern $\mathbf{I}^r$ is presented at the input layer F_1^a and the output $\mathbf{O}^r$ is presented at the output layer F_2^b. The vigilance parameter ρ_a is set to the baseline vigilance value $\overline{\rho}_a$.

Step 3: For every node j we calculate the choice function, as follows:

$$T_j(\mathbf{I}^r \mid \mathbf{w}_j) = g(j \mid \mathbf{I}^r) = \frac{n_j / (\sum_{l=1}^{N_a} n_l)}{\prod_{i=1}^{M_a} \sigma_{ji}} G(j \mid \mathbf{I}^r)$$

Step 4: Find the node J that has the maximum choice function (bottom-up input) value. That is find

$$J = \max_j \{T_j(\mathbf{I}^r)\}$$

Note: If there are more than one node indices that maximize the choice function choose the lowest index.

Step 5: Find the prediction of the activated node *J*. The prediction K of the input pattern $\mathbf{I}^r$, is the node *K* for which $W_{JK}^{ab} = 1$.

Step 6: If all the test patterns in the test set have not been applied to the network then go back to Step 2 and present the next input/output test pair in the sequence. If we have presented all the input/output test pairs then the results can be analyzed to find the misclassification error and other such statistics.

APPENDIX E: USER MANUAL

It is very important to mention that all the programs developed in this research have similar user interfaces, and hence, one user manual should be sufficient to represent all four of them. The manual is presented as a series of steps as follow:

Locate one of the following executable files (GFAM.exe, GEAM.exe, GGAM.exe or UART.exe) and double click it. When The selected program runs it expect that the user will supply values to at least the number of features, the training file, the validation file , and testing file, if any of the above is not supplied the program complains by popping up a message box to warn the user as in Figure e-1,

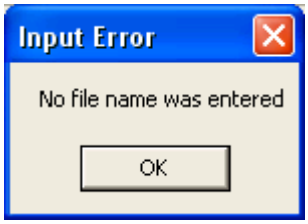

Figure e-1: Error message

All four programs give default values to all of the other parameters; the user though may change some of the default values. The user needs to click the "Create Init Pop" button, which prompts the program to take the following actions:

- Reads the files and extract the pattern information into objects called PtrnNode's.
- Stores the PtrnNode objects into vectors.
- Goes into this loop (as long as chromosomes counter is less than Pop_{size}):
  - Calculate the value of ρ_a. In particular, we first define $\rho_a^{inc} = \frac{\rho_a^{\max} - \rho_a^{\min}}{Pop_{size} - 1}$, and then the vigilance parameter of every network is determined by the equation $\rho_a^{\min} + i * \rho_a^{inc}$, where $i \in \{0, Pop_{size} - 1\}$
  - Invokes the network specific training algorithm (not discussed here), upon finishing, the specific network is created, this network is basically a group of Category objects along with its properties.

- The program encodes the network into a chromosome and stores the chromosome into a vector.

At this point we have a vector that contains all the Pop_{size} chromosomes. The program changes the display of the current ρ_a on the main window, signaling the user, that the training process is over. If the user clicks the optimize or the Next Gen button, the genetic optimization process starts, when the program stops, the user could use the draw networks button to see the categories of every chromosome, or could use the get boundaries button to see the classification boundaries of each chromosome. The user also can run the test file, buy clicking the Run on Test File button. The user can clear the internal memory of the program by clicking the clear button, this way the program is ready for another problem. The use of the manual add and delete category is explained earlier in sections 7.1.1, 7.2.1, 7.31, and 7.4.1

REFERENCES

1. Anagnostopoulos, G.C. & Georgiopoulos, M. (2001). Ellipsoid ART and ARTMAP for incremental clustering and classification, in Proc. of the IEEE-INNS International Joint Conference on Neural Networks, pp. 1221-1226, Washington, DC, July 14-19.
2. Anagnostopoulos, G.C. (2001). Novel Approaches in Adaptive Resonance Theory for Machine Learning, Doctoral Thesis, University of Central Florida, Orlando, Florida.
3. Anagnostopoulos, G.C., Bharadwaj, M., Georgiopoulos, M., Verzi, S.J. & Heileman, G.L. (2003). "Exemplar-based Pattern Recognition via Semi-Supervised Learning", to appear in the Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN ’03), Portland, Oregon: IEEE, INNS, ENNS
4. Anagnostopoulos, G.C., Bharadwaj, M., Georgiopoulos, M., Verzi, S.J., & Heileman, G. L. (2003). Exemplar-based pattern recognition via semi-supervised learning, in Proc. of the International Joint Conference on Neural Networks, Vol. 4, pp. 2782-2787, Portland, Oregon, July 20-24.
5. Anagnostopoulos, G.C., Georgiopoulos, M., Verzi, S.J., & Heileman, G.L. (2002). Reducing generalization error and category proliferation in ellipsoid ARTMAP via tunable misclassification error tolerance: Boosted Ellipsoid ARTMAP, in Proc. of the International Joint Conference on Neural Networks, Honolulu, Hawaii, May 12
6. Andon V. Topalov, Kwang-Choon Kim, Jong-Hwan Kim, Bong-Kuk Lee: Fast Genetic On-Line Learning Algorithm for Neural Network and Its Application to Temperature Control. International Conference on Evolutionary Computation 1996: 649-654
7. Auwatanamongkol, S., Pattern recognition using genetic algorithm. Evolutionary Computation, 2000. Proceedings of the 2000 Congress on Volume 1, 16-19 July 2000 Page(s):822 - 828 vol.1

8. Baker, J., “Adaptive selection methods for genetic algorithms”, Proc. Of 2ndICGA. 1987, pp.100-111
9. Bala, J., De Jong K., Huang, J., Vafaie, H., and Wechsler, H., “Using learning to facilitate the evolution of features for recognizing visual concepts,” Evolutionary Computation, Vol. 4, No. 3, pp. 297-311, 1996.
10. Belew, R. K., J. McInerney, and N. N. Schraudolph, “Evolving networks: Using genetic algorithm with connectionist learning,” Comput. Sci. Eng. Dep. (C-014), , Univ. of California, San Diego, Tech. Rep. CS90-174 (revised), Feb. 1991.
11. Brotherton, T. W., and Simpson, P. K., “Dynamic feature set training of neural nets for classification,” in Evolutionary Programming IV, J. R. McDonnel, R. G. Reynolds, and D. B. Fogel, Eds., Cambridge, MA, pp. 83-94, MIT Press, 1995.
12. Burton, A.R, Vladimirova, T., “Utilisation of an adaptive resonance theory neural network as a genetic algorithm fitness evaluator”, Proceedings of the 1997 IEEE International Symposium on Information Theory, June 29 - July 4, 1997, pages 209.
13. Burton, A.R.; Vladimirova, T.; Utilisation of an adaptive resonance theory neural network as a genetic algorithm fitness evaluator, Information Theory. 1997. Proceedings., 1997 IEEE International Symposium on 29 June-4 July 1997 Page(s):209
14. Cantu-Paz, E., “Feature sub-set selection by estimation of distribution algorithms,” in GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference.
15. Carpenter, G.A. & Milenova, B. (1998). Distributed ARTMAP: A neural network for fast distributed supervised learning, Neural Networks, 11 (2), 323-336.
16. Carpenter, G.A., Grossberg, S., Markuzon, N., & Reynolds, J.H. (1992). Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multi-dimensional maps, IEEE Trans. Neural Networks, 3 (5), 698-713.

17. Charalampidis, D., Kasparis, T., & Georgiopoulos, M. (2001). Classification of noisy signals using Fuzzy ARTMAP neural networks, IEEE Trans. Neural Networks, 12 (5), 1023-1036.

18. Davidor, Y.; CompEuro '90. Robot programming with a genetic algorithm, Proceedings of the 1990 IEEE International Conference on Computer Systems and Software Engineering 8-10 May 1990 Page(s):186 – 191

19. Fieldsend, J.E.; Singh, S.; Pareto evolutionary neural networks, Neural Networks, IEEE Transactions on Volume 16, Issue 2, March 2005 Page(s):338 - 354

20. Ghosh, R.; Verma, B.; Finding optimal architecture and weights using evolutionary least square based learning, in Neural Information Processing, 2002. ICONIP '02. Proceedings of the 9th International Conference on Volume 1, 18-22 Nov. 2002 Page(s):528 - 532 vol.1

21. Gomez-Sanchez, E., Dimitriadis, Y.A., Cano-Izquierdo, J.M., & Lopez-Coronado, J. (2001). Safe-μARTMAP: a new solution for reducing category proliferation in Fuzzy ARTMAP, in Proc. of the IEEE International Joint Conference on Neural Networks, Vol. 2, pp. 1197-1202, July 15-19.

22. Gomez-Sanchez, E., Dimitriadis, Y.A., Cano-Izquierdo, J.M., & Lopez-Coronado, J. (2002). μARTMAP: use of mutual information for category reduction in Fuzzy ARTMAP, IEEE Trans. Neural Networks, 13 (1), 58-69.

23. Grossberg, S. (1976). Adaptive pattern recognition and universal recoding II: Feedback, expectation, olfaction, and illusions, Biological Cybernetics, 23, 187-202.

24. Han, Seung-Soo, Gary S. May: Optimization of Neural Network Structure and Learning Parameters Using Genetic Algorithms. ICTAI 1996: 200-206

25. Hancock, P. J. B., “Pruning neural networks by genetic algorithm,” In Proceedings of the 1992 International Conference on Neural Networks, I. Aleksander and J. Taylor, Eds., Amsterdam, Netherlands, Vol. 2, pp. 991-994, Elsevier Science, 1992.

26. Harp, S. A., T. Samad and A. Guha. Towards the genetic synthesis of neural networks. Proceedings of the third international conference on genetic algorithms. Morgan Kaufmann, 1989

27. Hettich, S., Blake, C.L., & Merz, C.J. (1998). UCI Repository of machine learning databases [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science.

28. Holland, J. H., Adaptation in Natural and Artificial Systems. Ann Arbor, MI: Univ. of Michigan Press, 1975

29. Hruschka, E.R.; Ebecken, N.F.F.; Applying a clustering genetic algorithm for extracting rules from a supervised neural network, in Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on Volume 3, 24-27 July 2000 Page(s):407 - 412 vol.3

30. Hui Liu; Yue Liu; Jian Liu; Bofeng Zhang; Gengfeng Wu; Impulse force based ART network with GA optimization, Neural Networks and Signal Processing, 2003. Proceedings of the 2003 International Conference on Volume 1, 14-17 Dec. 2003 Page(s):499 - 502 Vol.1

31. Inza, I., Larranaga, P., Etxeberria, R., and Sierra, B., “Feature sub-set selection by Bayesian networks based optimization,” Artificial Intelligence, Vol. 123, No. 1-2, pp. 157-184, 1999.

32. Ishibuchi, H.; Nakashima, T.; Evolution of fuzzy nearest neighbor neural networks, in Evolutionary Computation, 1997., IEEE International Conference on 13-16 April 1997 Page(s):673 – 678

33. James D. Kelly Jr., Lawrence Davis: Hybridizing the Genetic Algorithm and the K Nearest Neighbors Classification Algorithm. ICGA 1991: 377-383

34. Juang Chia-Feng; Yuan-Chang Liou; On the hybrid of genetic algorithm and particle swarm optimization for evolving recurrent neural network, Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on Volume 3, 25-29 July 2004 Page(s):2285 - 2289 vol.3

35. Kasuba, T. (1993). Simplified Fuzzy ARTMAP, AI Expert, 18-25.

36. Konstantinos P. Ferentinos: Biological engineering applications of feedforward neural networks designed and parameterized by genetic algorithms. Neural Networks 18 (7): 934-950 (2005)

37. Koufakou, A., Georgiopoulos, M., Anagnostopoulos, G.C., & Kasparis, T. (2001). Cross-validation in Fuzzy ARTMAP for large databases, Neural Networks, (14), 1279-1291.

38. Krasniewicz, J.; Winfield, M.J.; Green, P.; Design of a distributed neuro-genetic learning architecture, in Knowledge-Based Intelligent Engineering Systems and Allied Technologies, 2000. Proceedings. Fourth International Conference on Volume 2, 30 Aug.-1 Sept. 2000 Page(s):612 - 615 vol.2

39. Langdon, W. B., E. Cantu-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishna, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, Eds., San Francisco, CA, 2002, pp. 302-310, Morgan Kaufmann Publishers, 2002.

40. Lee, S.-W., “Off-line recognition of totally unconstrained handwritten numerals using multilayer cluster neural network,” IEEE Trans. Pattern Anal. Machine Intell., vol. 18, pp. 648–652, June 1996.

41. Leung, F.H.F.; Lam, H.K.; Ling, S.H.; Tam, P.K.S.; Tuning of the structure and parameters of a neural network using an improved genetic algorithm, Neural Networks, IEEE Transactions on Volume 14, Issue 1, Jan. 2003 Page(s):79 - 88

42. Liang-Hsuan Chen; Cheng-Hsiung Chiang; An intelligent control system based on multiobjective genetic algorithms and fuzzy neural network, in Systems, Man and Cybernetics, 2002 IEEE International Conference on Volume 3, 6-9 Oct. 2002 Page(s):6 pp. vol.3

43. Liu, H., Liu, Y., Liu, J., Zhang, B., Wu, G., "Impulse force based ART network with GA optimization, Neural Networks and Signal Processing", Proceedings of the 2003 International Conference on Neural Networks and Signal Processing, Volume 1, Dec. 14-17, 2003, pages 499 – 502.

44. Manic, M.; Wilamowski, B.; Robust algorithm for neural network training, Neural Networks, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on Volume 2, 12-17 May 2002 Page(s):1528 - 1533

45. Marriott, S., & Harrison, R.F. (1995). A modified Fuzzy ARTMAP architecture for the approximation of noisy mappings, Neural Networks, 8 (4), 619-641.

46. Miller, G. F., Todd, P. M., Hedge, S. U., "Designing neural networks using genetic algorithms," In Proceedings of the Third International Conference on Genetic Algorithms, San Mateo, CA, J.D. Scaffer, Ed., Morgan Kaufmann, 1989.

47. Mitchell, Tom, machine learning. McGraw-Hill, 1997, ISBN:0-07-042807-7

48. Palaniappan, R.; Raveendran, P.; Genetic algorithm to select features for fuzzy ARTMAP classification of evoked EEG, in Circuits and Systems, 2002. APCCAS '02. 2002 Asia-Pacific Conference on Volume 2, 28-31 Oct. 2002 Page(s):53 - 56 vol.2

49. Palaniappan, R.; Raveendran, P.; Omatu, S.; VEP optimal channel selection using genetic algorithm for neural network classification of alcoholics, in Neural Networks, IEEE Transactions on Volume 13, Issue 2, March 2002 Page(s):486 – 491

50. Palmes, P., T. Hayasaka, and S. Usui. Mutation-based genetic neural network. IEEE Transactions on Neural Network, 2004.

51. Parrado-Hernandez E., Gomez-Sanchez, E., & Dimitriadis, Y.A. (2003). Study of distributed learning as a solution to category proliferation in Fuzzy ARTMAP-based neural systems, Neural Networks, (16), 1039-1057.

52. Punch, W. F., E. D. Goodman, M. Pei, L. ChiaShun, P. Hovland, and R. Enbody, Further Research on Feature Selection and Classification Using Genetic Algorithms, International Conference on Genetic Algorithms 93, pp. 557-564, 1993

53. Rovithakis, G.A.; Chalkiadakis, I.; Zervakis, M.E.; High-order neural network structure selection for function approximation applications using genetic algorithms, in Systems, Man and Cybernetics, Part B, IEEE Transactions on Volume 34, Issue 1, Feb. 2004 Page(s):150 - 158

54. Santos, R.T.; Nievola, J.C.; Freitas, A.A.; Extracting comprehensible rules from neural networks via genetic algorithms, in Combinations of Evolutionary Computation and Neural Networks, 2000 IEEE Symposium on 11-13 May 2000 Page(s):130 – 139

55. Siedlecki, W., and Sklansky, J., “A note on genetic algorithms for large-scale feature selection,” Pattern Recognition Letters, Vol. 10, pp. 335-347, 1989.

56. T.Ba¨ck, U.Hammel, and H.-P.Schwefel, “Evolutionary computation: Comments on the history and current state,” IEEE Trans. Evolutionary Computation, vol. 1, pp. 3-17, Apr. 1997.

57. Taghi, M., Bagmisheh, V., & Pavesic, N. (2003). A Fast Simplified Fuzzy ARTMAP network, Neural Processing Letters, 17(3), 273-316.

58. Tang, K. S., C. Y. Chan, K. F. Man, and S. Kwong, “Genetic structure for NN topology and weights optimization,” in Proceedings of the 1st IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALE-SIA’95), (Stevenage, England), pp. 250-255, IEE Conference Publication 414, 1995.

59. Verzi, S.J., Georgiopoulos, M., Heileman, G.L., & Healy, M. (2001). Rademacher penalization applied to Fuzzy ARTMAP and Boosted ARTMAP, in Proc. of the IEEE-INNS International Joint Conference on Neural Network, pp. 1191-1196, Washington, DC, July 14-19.

60. Whitley, D., “The GENITOR algorithm and selective pressure: Why rank-based allocation of reproductive trials is best,” in Proc. 3rd Int. Conf. Genetic Algorithms and Their Applications, J. D. Schaffer, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 116–121.

61. Whitley, D., Starkweather, T., and Bogart, C., “Genetic algorithms and neural networks: Optimizing connections and connectivity,” Parallel Computing, Vol. 14, pp. 347-361, 1990.

62. Whitley, D., T. Starkweather, and C. Bogart, “Genetic algorithms and neural networks: Optimizing connections and connectivity,” Parallel Comput., vol. 14, no. 3, pp. 347–361,1990.

63. Williamson, J.R. (1996). Gaussian ARTMAP: A Neural Network for Fast Incremental Learning of Noisy Multi-Dimensional Maps, Neural Networks, 9(5), 881-897.

64. Williamson, J.R. (1997). A constructive, incremental-learning network for mixture modeling and classification, Neural Computation, (9), 1517-1543.

65. Wu, Annie S., Han Yu, Shiyuan Jin, Kuo-Chi Lin, and Guy Schiavone (2004). An incremental genetic algorithm approach to multiprocessor scheduling. IEEE Transactions on Parallel and Distributed Systems , 15:9.

66. Wu, Annie S., Ivan I. Garibay (2004). Genetic algorithm optimization of life support system control. IEEE Transactions on Systems, Man, and Cybernetics, Part B , 34:3, 1423-1434

67. Xin Yao; Evolving artificial neural networks, in Proceedings of the IEEE, Volume 87, Issue 9, Sept. 1999 Page(s):1423 - 1447

68. Xiuju Fu; Lipo Wang; A GA-based RBF classifier with class-dependent features, in Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on Volume 2, 12-17 May 2002 Page(s):1890 – 1894

69. Yang, J., and Honavar, V., “Feature subset selection using genetic algorithms,” IEEE Intelligent Systems, Vol. 13, pp. 44-49, 1998.

70. Yen, G.G.; Haiming Lu; Hierarchical genetic algorithm based neural network design, in Combinations of Evolutionary Computation and Neural Networks, 2000 IEEE Symposium on 11-13 May 2000 Page(s):168 - 175