

Amit Konar

**Computational Intelligence**

Principles, Techniques and Applications

Amit Konar

# **Computational Intelligence**

Principles, Techniques and Applications

With 215 figures and 13 tables



Springer

**Prof. Dr. Amit Konar**  
Artificial Intelligence Laboratory  
Electronics and Tele-Communication Engineering Department  
Jadavpur University  
Calcutta 700 032  
India

Library of Congress Control Number:2004117176

ISBN 3-540-20898-4 **Springer Berlin Heidelberg New York**

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilm or in other ways, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable to prosecution under German Copyright Law.

**Springer is a part of Springer Science+Business Media**  
[springeronline.com](http://springeronline.com)

© Springer-Verlag Berlin Heidelberg 2005  
Printed in The Netherlands

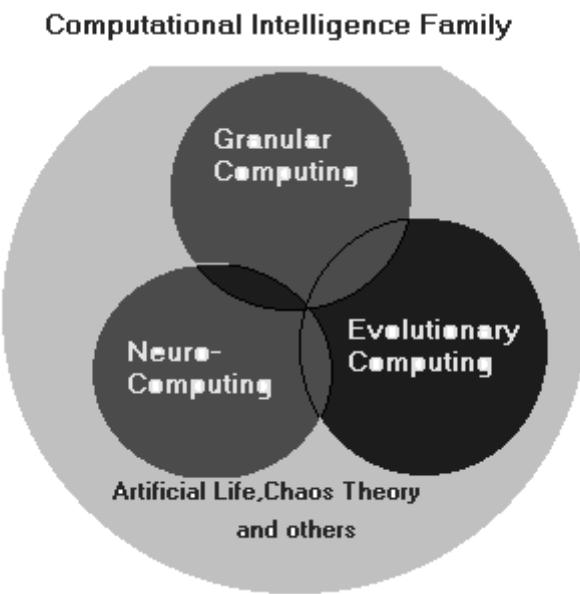
The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Typesetting: Data conversion by the author.  
Final processing by PTP-Berlin Protago-T<sub>E</sub>X-Production GmbH, Germany  
Cover-Design: Erich Kirchner, Heidelberg  
Printed on acid-free paper 89/3141/Yu – 5 4 3 2 1 0

# Preface

## 1. About the Subject

Computational Intelligence (CI) is a brainchild of Professor Lotfi A. Zadeh. Since its inception in the early 1990's, the subject has undergone a dramatic change in both its content and organization. The early definition of CI is centered around the logic of fuzzy sets, neural networks, genetic algorithm and probabilistic reasoning along with the study of their synergism. The modern definition of CI is greatly influenced by the biologically inspired models of machine intelligence. It deals with the models of granular computing, neural computing and evolutionary computing and their interactions with artificial life, chaos theory and others. Fig. 1 provides a brief outline of the definition of modern CI. The list of the members of the computational intelligence family is given in Fig. 2.

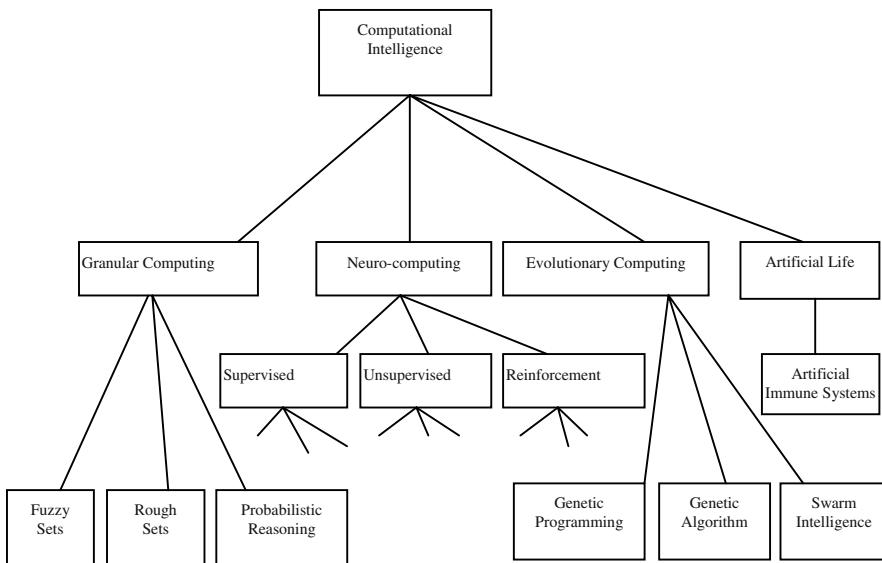


**Fig. 1:** The modern definition of CI.

## 2. About the Book

The book takes a modest attempt to cover the entire framework of computational intelligence and its applications in a single volume. It includes 23 chapters, covering all aspects of the subject in a clear, precise and highly comprehensive style. The book also includes two appendices. Appendix-A contains sample runs of programs, the source codes of which are supplied in a companion CD. Appendix-B includes evolutionary algorithms of recent interest, which are

increasingly being used to handle complex engineering problems of diverse domains.



A parent node denotes a broad discipline, and a child node denotes a subset of its parent. Detailed list of children of a parent node is not given to keep the diagram readable.

**Fig. 2:** A fragment of the computational intelligence family tree.

## Features

- This is the first text on the subject that covers all aspects of computational intelligence in a clear, precise and highly comprehensive style.
- The book includes plenty of numerical examples and worked out problems.
- The exercise is carefully organized to guide students to handle complex real world problems. Hints are given to most problems so as to enable the students to complete the solutions on their own.
- Case studies on control, databases, telecommunication, image understanding and robotics will help the students to grow interest in the subject and enhance their potential to handle problems of similar domains.
- Research problems included in chapter 23 will enable graduate researchers to identify problems of their own choice.

- Source code of some interesting problems supplied in the companion CD will enhance the level of confidence of the students as they can examine the direct impact of the theories in practical problems.

### **3. The Audience**

The book is primarily meant for graduate students of Electrical Engineering and Computer Science. Students of any other engineering discipline, science and humanities will also find the book a useful resource for possible applications in their respective domains. A little background in linear algebra and high school calculus, however, is a pre-requisite for the latter group of students. The book is equally useful to practicing engineers as the classical methods of system design, identification, optimization, diagnosis and control are currently being replaced by the models of computational intelligence. Research students looking for a suitable problem for their MS or Ph D thesis will also find this book helpful as it introduces a number of open-ended problems with sufficient hints to each of them.

### **4. Contents**

The book consists of two main parts. Part I of the book is concerned with fundamental issues of computational intelligence, whereas part II deals with specialized topics, emphasizing synergism of the computational models introduced in part I. A section of part II is dedicated to engineering applications of computational intelligence principles.

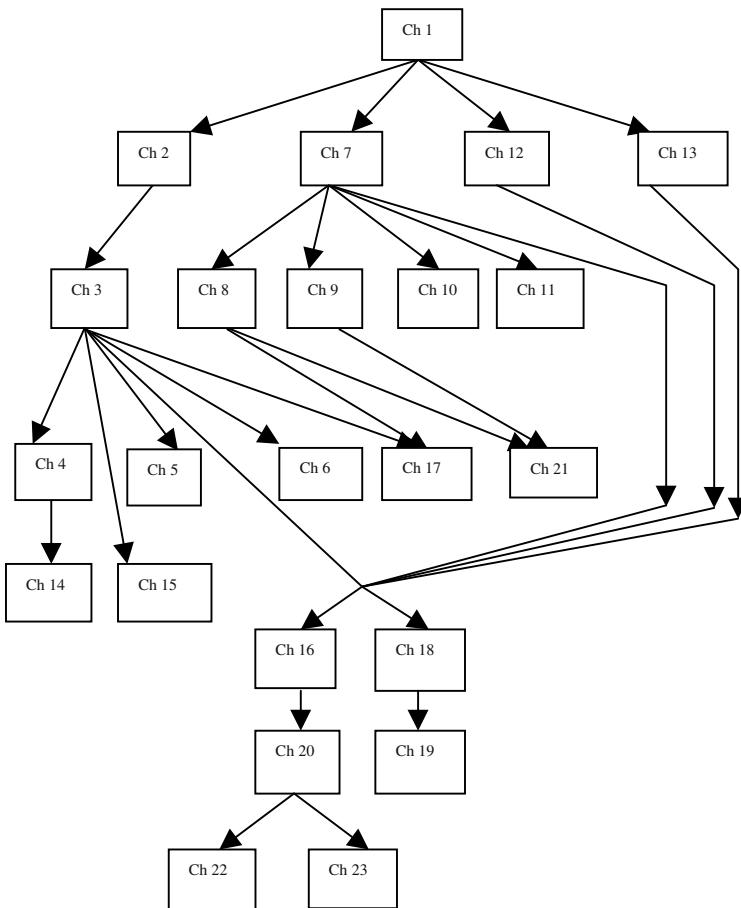
Part I of the book contains 13 chapters. Chapter 1 provides a precise introduction to the subject. Principles of fuzzy sets, fuzzy logic and approximate reasoning are covered in chapters 2 and 3 respectively. Fuzzy control is introduced in chapter 4. Fuzzy databases and fuzzy pattern recognition are covered in chapter 5 and 6 respectively. Chapter 7 through 11 is devoted to neural nets. Fundamental principles of neural nets are covered in chapter 7. Principles of supervised, unsupervised, reinforcement and competitive learning are discussed in chapters 8, 9 10 and 11 respectively. Evolutionary computing algorithms are introduced in chapter 12. Methodologies of belief computing are presented in chapter 13.

Part II of the book begins with reasoning in fuzzy Petri nets outlined in chapter 14. Algorithms for human face recognition and mood detection are covered in chapter 15. Chapter 16 presents the principles of synergism of soft computing tools. Chapter 17 covers neuro-fuzzy synergism using fuzzy ADALINE neurons. Chapter 18 introduces machine learning models on a fuzzy cognitive map. Chapter 19 presents machine learning by fuzzy Petri nets. Chapter 20 discusses the scope of Computational Intelligence tools in Computer Networking. Chapter 21 presents neuro-Kalman synergism in target-tracking application of mobile robots. Chapter 22 deals with issues of modern relevance, such as granular computing, rough sets, particle swarms, artificial immune

systems and others. Chapter 23 presents open-ended research-problems for graduate students and researchers in computational intelligence.

## 5. How to plan your tour?

Since the book has a large volume, a guide to plan your tour is necessary to save your valuable time. Fig. 3 provides a directed acyclic graph depicting the order of traversal of the chapters. The arrow between a predecessor and a successor in this graph indicates that the successor chapter should be read after the predecessor has been completed.



**Fig. 3:** A dependence graph representing the order of studying the chapters.

## **Acknowledgement**

The author would like to thank many of his friends, colleagues and co-workers for help, co-operation and support, without which the book could not be completed in the present form.

First and foremost the author wishes to thank Professor A. N. Basu, Vice-Chancellor, Jadavpur University for providing him the necessary support to write the book. He is equally indebted to Professor M. Mitra, Dean, faculty of Engineering and Technology, Jadavpur University for encouraging him to write the book. The author also wishes to thank Prof. S. K. Sanyal, the Deputy Vice-Chancellor, Jadavpur University, for his constant mental support in writing the book. The author gladly acknowledges the support extended by the current and the past two HODS of his department: Professor A. K. Bandyopadhyay, Professor H. Saha and Professor R. Nandi for the successful completion of the book.

The author is indebted to all his B. E., M.E. and Ph.D. students for sharing their views over the past 15 years, which ultimately took shape in the form of the book. The author gratefully acknowledges the contribution of his students: Swagatam Das, Biswarup Kolay, Adris Misra, Priyanko Mitra, Soumyajit Dey, Sonai Ray, Debraj De, Saibal Mukhopadhyay, Abhijit Das and Abhik Mukherjee in developing the software supplied with the book on a CD. Among his Ph.D. students, the author acknowledges the contribution of Ms. Aruna Chakraborty and Ms. Mahasweta Bagchi for their constant support in reviewing and editing several versions of the manuscript to improve its readability.

Professor Jaya Sil, the senior-most ex-doctoral student of the author used this book for teaching a one-semester course in Computer Science department, Calcutta University and provided the author necessary feed-backs to update the manuscript. The author is indebted to Dr Bijita Biswas, Dr. Parbati Saha, Dr. Alakananda Bhattacharya and Dr. Sanjukta Pal, the four recent Ph. D. recipients, for sharing their knowledge on different areas of computational intelligence over the past 5/6 years. Dr. Biswajit Paul and Dr. Srikant Patnaik, two senior ex-doctoral students of the author also helped him improving his mathematical skill in the subject through many hours of discussions. Mr. Diptendu Bhattacharya, Ms. Sheli Sinha Choudhuri and Ms. Indrani Chakraborty, the current Ph.D. students of the author always inspire him to successfully complete the book.

The teachers and current colleagues of the author also played vital roles in the preparation of the manuscript. The author owes a deep gratitude to Professor A. K. Mandal of ETCE department, J. U., for teaching him the subject of AI from the first principles and providing him moral support as a teacher, Ph.D. thesis advisor and colleague. The author is also indebted to Professor A. K. Nath of the same department, who has recently retired from his service, for encouraging him to write books and spending long hours in valuable

discussions. The author would like to thank his teacher Professor A. B. Roy of the department of Mathematics, Jadavpur University, who inspired his writing skill, which later enabled him to write this book. The author gratefully acknowledges the contribution of his one-time project supervisor Professor T. K. Ghosal of Electrical Engineering department, Jadavpur University for his constructive criticism, which helped him develop a habit of checking a thought twice before deliberating. In a personal meeting of the author with Professor Ghosal in the late eighties, Professor Ghosal stressed the need for stability analysis of fuzzy/ non-monotonic systems, which later provoked the author to study the dynamic behavior of those systems. Most of the author's current research work is centered around the then thoughts of Professor Ghosal. In connection with his research work under the *University Potential for Excellence Program in Cognitive Science*, the author came in touch with a great personality. She is Professor Amita Chatterjee, the coordinator of the above program. During the preparation of the manuscript, Professor Chatterjee shared her knowledge on Cognitive Science with the author for several hours. Her inspiration and cooperation helped the author in many ways to complete the book.

The author gratefully acknowledges the contribution of a number of international researchers, who directly or indirectly helped him in many ways to write this book. The author gratefully acknowledges the collaborative technical support he received from Professor Lakhmi C. Jain of University of South Australia, Adelaide in connection with the preparation of the book. Professor Henry Hexmoor of University of Arkansas provided him both technical and moral support in all respects in connection with the publication of the book. Professor Carl G. Looney of University of Nevada, the father of Fuzzy Petri Nets, extended all the support that the author needed in connection with the preparation of the book. The author would also like to thank Professor Witold Pedrycz of University of Alberta, Canada for his inspiration and encouragement in connection with writing this book. Dr. Uday Chakraborty of University of Missouri, St. Louis also helped the author in many ways in writing the book.

Among his colleagues and well-wishers, the author would like to mention the name of the following personalities: Professor P. K. Sinha Roy of the Institute of Engineering and Management, Salt Lake, and Dr. B. Gupta, Dr. Subir. K. Sarkar, Dr. Chandan K. Sarkar and Dr. Iti Saha Mishra of ETCE Department, Jadavpur University. The author is indebted to many other personalities including Professor D. Ghosh Dastidar and Professor M. K. Roy of the Information Technology department, Jadavpur University, who taught the author the fundamentals of algorithms and programming languages. His friends Dr. Bijoy Bhattacharya of Production Engineering department and Dr. Dipak Laha of Mechanical Engineering department always encouraged him to write books.

The author would like to mention his sincere thanks to Mr. Gourishankar Chattopadhyay and Ms. Ishita Chattopadhyay, who always wished him a successful career since his childhood. He would also like to thank his one time

favorite friend Ganesh (Gabu), who always inspired the author to fix his goals ahead.

The author would like to acknowledge the cooperation he received from the Acquisition Editor and staff members of Springer-Verlag publisher.

The author also acknowledges the financial and technical support he received from 2 UGC-sponsored projects: i) AI and Expert Systems Applied to Imaging and Robotics, and ii) University with Potential for Excellence Program in Cognitive Science.

Lastly, the author expresses his deep gratitude to his parents: Mr. Sailen Konar and Mrs. Minati Konar, who played a significant role in his success as a teacher, researcher and author of several books. He would also like to thank his brother Sanjoy, who always inspired the author to fulfill his ambitions. He also wishes to thank his wife Srilekha for relieving the author from the ins and outs of the household responsibilities, without which the book of this much volume could not be completed.

The author mourns the sudden death of his father-in-law: Mr. Sahadeb Samanta, who could have been much happy to see the book in print.

The author finally remembers the charming faces of his son Dipanjan, his nephew Arpan and niece Samiparna, whose love and imagination created inspiration and encouragement for the successful completion of the book.

Robotics and Vision Laboratory  
ETCE department  
Jadavpur University

Amit Konar

August 15, 2004.

# Contents

## Chapter 1: An Introduction to Computational Intelligence

- 1.1 Artificial Intelligence- a Brief Review 1
- 1.2 Pitfalls of the Traditional AI 3
- 1.3 Computational Intelligence- an Emergence of a New Computational Paradigm 4
- 1.4 Computational Intelligence- a Formal Definition 7
- 1.5 Soft Computing- Definitions 8
- 1.6 Fundamental Elements of Soft Computing 9
  - 1.6.1 The Logic of Fuzzy Sets 9
  - 1.6.2 Computational Models of Neural Nets 10
  - 1.6.3 Genetic Algorithms 17
  - 1.6.4 Belief Networks 20
- 1.7 Computational Learning Theory 23
- 1.8 Synergism in Soft Computing 25
  - 1.8.1 Neuro-Fuzzy Synergism 26
  - 1.8.2 Neuro-GA Synergism 26
  - 1.8.3 Fuzzy-GA Synergism 26
  - 1.8.4 Neuro-Belief Network Synergism 26
  - 1.8.5 GA-Belief Network Synergism 27
  - 1.8.6 Neuro-Fuzzy-GA Synergism 27
- 1.9 Conclusions 27
  - Exercise 27
  - References 29

## Chapter 2: Fuzzy Sets and Relations

- 2.1 Conventional Sets 37
- 2.2 Fuzzy Sets 38
- 2.3 Membership Functions 40
- 2.4 Continuous and Discrete Membership Functions 42
- 2.5 Typical Membership Functions 43
  - 2.5.1 The  $\gamma$ -Function 43
  - 2.5.2 The s-Function 44
  - 2.5.3 The L-Function 45
  - 2.5.4 The Triangular Membership Function 46
  - 2.5.5 The  $\Pi$ -Function 47
  - 2.5.6 The Gaussian Membership Function 47
- 2.6 Operations on Fuzzy Sets 48
  - 2.6.1 Fuzzy T-Norm 48
  - 2.6.2 Fuzzy S-Norm 49
  - 2.6.3 Fuzzy Complement 50
- 2.7 Basic Concepts Associated with Fuzzy Sets 50
- 2.8 Extension Principle of Fuzzy Sets 52

- 
- 2.9 Fuzzy Relations 54
  - 2.10 Projection of Fuzzy Relations 56
  - 2.11 Cylindrical Extension of Fuzzy Relations 57
  - 2.12 Fuzzy Max-Min and Max-Product Composition Operation 58
  - 2.13 Fuzzy Linguistic Hedges 60
  - 2.14 Summary 62
    - Exercise 63
    - References 66

## **Chapter 3: Fuzzy Logic and Approximate Reasoning**

- 3.1 Production Systems 67
- 3.2 Conflict Resolution in Production Systems 69
- 3.3 Drawbacks of Traditional Production Systems 69
- 3.4 Fuzzy Implication Rules 70
- 3.5 Fuzzy Implication Relations 71
- 3.6 Fuzzy Logic 73
  - 3.6.1 Typical Propositional Inference Rules 73
  - 3.6.2 Fuzzy Extension of the Inference Rules 74
- 3.7 The Composition Rule of Inference 75
  - 3.7.1 Computing Fuzzy Inferences in GMP 75
  - 3.7.2 Computing Fuzzy Inferences Using GMT 78
  - 3.7.3 Computing Fuzzy Inferences Using GHS 79
- 3.8 Approximate Reasoning with Multiple Antecedent Clauses 80
- 3.9 Approximate Reasoning with Multiple Rules 83
- 3.10 Scope of Parallelism in Approximate Reasoning Using Mamdani Implication Function 86
- 3.11 Realization of Fuzzy Inference Engine on VLSI Architecture 87
- 3.12 Approximate Reasoning with Multiple Rules Each with Multiple Antecedent Clauses 89
- 3.13 Fuzzy Abductive Reasoning 90
- 3.14 Conclusions 93
  - Exercise 93
  - References 95

## **Chapter 4: Fuzzy Logic in Process Control**

- 4.1 Process Control 97
- 4.2 Advantages of Fuzzy Control 99
- 4.3 Typical Fuzzy Control Systems 99
- 4.4 Architecture of Typical Fuzzy control Systems 101
- 4.5 Reasoning in Mamdani Type Fuzzy Control Systems 101
- 4.6 Reasoning in T-S Fuzzy Control Systems 106
- 4.7 Stability Analysis of Dynamic Systems Using Lyapunov Energy Functions 109
- 4.8 Stability Analysis of T-S Fuzzy Systems 110
- 4.9 Application in Power Control of a Nuclear Reactor 114

4.10 Defuzzification Techniques 119

4.11 Conclusions 120

    Exercise 120

    References 122

## **Chapter 5: Fuzzy Pattern Recognition**

5.1 Introduction 125

5.2 The Fuzzy C-Means Clustering Algorithm 127

5.3 Image Segmentation Using Fuzzy C-Means  
    Clustering Algorithm 132

5.4 Conclusions 136

    Exercise 136

    References 137

## **Chapter 6: Fuzzy Databases and Possibilistic Reasoning**

6.1 Introduction to Relational Database Systems 139

    6.1.1 The Relational Model 140

6.2 Issues in Relational database Design 143

    6.2.1 Lossless Join Decomposition 143

    6.2.2 Functional Dependency Preservation 144

6.3 Possibility Distribution and Fuzzy Sets 146

6.4 Fuzziness in Relational Models 147

    6.4.1 Type-1 Fuzzy Relational Data Model 147

    6.4.2 Type-2 Fuzzy Relational Data Model 149

6.5 Fuzzy Relational Operators 152

    6.5.1 Projection of Fuzzy Relations 152

    6.5.2 Cylindrical Extension of Fuzzy Relations 153

    6.5.3 Natural Join of Fuzzy Relations 154

    6.5.4 Lossy Fuzzy Joins 155

6.6 Fuzzy Integrity Constraints 155

    6.6.1 Conjunction of Fuzzy Propositions 156

    6.6.2 The Modifier Rule 157

    6.6.3 Possibility Distribution for Conditional  
        Fuzzy Propositions 158

6.7 Fuzzy Functional Dependency 158

    6.7.1 Equality as a Fuzzy Relation 158

    6.7.2 Representing Functional Dependency Using  
        Equality Relation 160

6.8 Fuzzy Lossless Join 161

6.9 Design of Fuzzy Relational Databases 162

6.10 Conclusions 162

    Exercise 163

    References 165

## **Chapter 7: Introduction to Machine Learning Using Neural Nets**

- 7.1 Biological Neural Networks 167
- 7.2 Artificial Neural Networks 169
- 7.3 Principles of Learning in a Neural Net 170
- 7.4 Stability and Convergence 179
- 7.5 Three Important Theorems for Stability Analysis of Neural Dynamics 180
- 7.6 Conclusions 193
  - Exercise 184
  - References 194

## **Chapter 8: Supervised Neural Learning Algorithms**

- 8.1 Introduction 197
- 8.2 McCulloch-Pitts Model 198
- 8.3 The Perceptron Learning Model 201
  - 8.3.1 Linear Classification by Perceptrons 203
  - 8.3.2 Multilayered Perceptron Classifier 207
- 8.4 Widrow-Hoff's ADALINE Model 209
- 8.5 The Back-propagation Learning Algorithm 217
- 8.6 The Radial Basis Function Neural Net 223
- 8.7 Modular Neural Nets 225
- 8.8 Conclusions 230
  - Exercise 230
  - References 233

## **Chapter 9: Unsupervised Neural Learning Algorithms**

- 9.1 Introduction 237
- 9.2 Recurrent Neural Nets 238
  - 9.2.1 Discrete Hopfield Network 239
  - 9.2.2 Continuous Hopfield Neural Net 241
  - 9.2.3 A Near-Equilibrium Consideration for Classification by Continuous Hopfield Nets 245
  - 9.2.4 Optimization Using Hopfield (Continuous) Neural Nets 246
  - 9.2.5 Application of Hopfield Neural Nets in Engineering Problems 248
  - 9.2.6 Boltzman Machines 252
- 9.3 Bi-directional Associative Memory 253
- 9.4 Adaptive Resonance Theory 256
- 9.5 Fuzzy Associative Memory 260
- 9.6 Discussions 262

Exercise 263  
References 265

## **Chapter 10: Competitive Learning Using Neural Nets**

- 10.1 Introduction 267
- 10.2 Two Layered Recurrent Networks for Competitive Learning 270
- 10.3 Components of a Competitive Learning Network 272
  - 10.3.1 The Pre-Processing Layer 272
  - 10.3.2 The Instar Connectivity from Neurons of the Input to the Output Layer 273
  - 10.3.3 Competitive Learning in the Output Layer 274
- 10.4 Hebbian Learning in the Competitive Layer 276
- 10.5 Analysis of Pattern Clustering Network 279
- 10.6 Principal Component Analysis 281
- 10.7 Self-Organizing Feature Map 282
- 10.8 Application in Face Recognition 284
- 10.9 Conclusions 289
- Exercise 289
- References 292

## **Chapter 11: Neuro-dynamic Programming by Reinforcement Learning**

- 11.1 Introduction 295
- 11.2 Formulation of the Reinforcement Learning Paradigm 298
- 11.3 Q-Learning 302
- 11.4 Convergence of Q-Learning Algorithm 307
- 11.5 Nondeterministic Rewards and Actions 308
- 11.6 Temporal Difference Learning 310
- 11.7 Neural Reinforcement Learning 311
- 11.8 Multi-Agent Reinforcement Learning 315
- 11.9 Conclusions 317
- Exercise 318
- References 321

## **Chapter 12: Evolutionary Computing Algorithms**

- 12.1 Introduction 323
- 12.2 Genetic Algorithm: How does it work? 324
- 12.3 Deterministic Explanation of Holland's Observation 331
- 12.4 Stochastic Explanation of GA 332
- 12.5 The Markov Model for Convergence Analysis 335
- 12.6 Application of GA in Optimization Problems 339
- 12.7 Application of GA in Machine Learning 341
  - 12.7.1 GA as an Alternative to Back-propagation Learning 342

---

12.7.2	Adaptation of the Learning Rule/ Control Law by GA	342
12.8	Application of GA in Intelligent Search	345
12.8.1	Navigational Planning of Robots	345
12.9	Genetic Programming	346
12.10	Conclusions	348
	Exercise	349
	References	350

## **Chapter 13: Belief Calculus and Probabilistic Reasoning**

13.1	Introduction	353
13.2	Elements of Probability Theory	354
13.2.1	Bayes' Law on Conditional Probability	356
13.3	Belief Propagation on a Causal Tree	361
13.4	Pearl's Belief Propagation Scheme on a Polytree	369
13.5	Dempster-Shafer Theory for Uncertainty Management	372
13.6	Conclusions	377
	Exercise	377
	References	391

## **Chapter 14: Reasoning in Expert Systems Using Fuzzy Petri Nets**

14.1	Introduction	394
14.2	Imprecision Management in an Acyclic FPN	396
14.2.1	Formal Definitions and the Proposed Model	396
14.2.2	Proposed Model for Belief Propagation	396
14.2.3	Proposed Algorithm for Belief Propagation	398
14.3	Imprecision and Inconsistency Management in a Cyclic FPN	404
14.3.1	Proposed Model for Belief Revision	404
14.3.2	Stability Analysis of the Belief Revision Model	405
14.3.3	Detection and Elimination of Limit cycles	411
14.3.4	Nonmonotonic Reasoning in an FPN	414
14.4	Conclusions	415
	Exercise	419
	References	419

## **Chapter 15: Image Matching Using Fuzzy Moment Descriptors**

15.1	Introduction	423
15.2	Image Features and their Membership Distributions	425
15.2.1	Fuzzy Membership Distributions	426
15.2.2	Fuzzy Production Rules	428
15.3	Fuzzy Moment Descriptors	429

15.4	Image Matching Algorithm	431
15.5	Rotation and Size Invariant Matching	433
15.6	Noise Insensitive Matching	433
15.7	Computer Simulation	434
15.8	Implication of the Results	435
15.9	Template Matching Using Interleaved Search	435
15.10	Computer Simulation of Template Matching	439
15.11	Human Mood Detection from Facial Expressions	440
15.11.1	Image Segmentation and Localization of Facial Components	441
15.11.2	Facial Extracts and their Measurements for Mood Analysis	442
15.12	Conclusions	446
	Exercise	447
	References	450

## **Chapter 16: Behavioral Synergism of Soft Computing Tools**

16.1	Introduction	453
16.2	Neuro-Fuzzy Synergism	455
16.2.1	Weakly Coupled Neuro-Fuzzy Systems	455
16.2.2	Tightly Coupled Neuro-Fuzzy Systems	456
16.3	Fuzzy-GA Synergism	459
16.4	Neuro-GA Synergism	460
16.4.1	Adaptation of a Neural Learning Algorithm Using GA	460
16.5	GA-Belief Network Synergism	462
16.6	A Case Study of Synergism of 2 Neural Topology and GA	462
16.6.1	The Problem	462
16.6.2	Clustering by TASONN	463
16.6.3	Continuous Hopfield Net- A Review	463
16.6.4	Perception-to-Action Transformation	464
16.6.5	Optimization of the Energy Function Using GA	465
16.6.6	Experimental Details	466
16.7	Conclusions and Future Directions	468
	Exercise	469
	References	475

## **Chapter 17: Object Recognition from Gray Images Using Fuzzy ADALINE Neurons**

17.1	Introduction	477
17.2	Proposed Model of ADALINE	480
17.3	Stability Analysis for Convergence of the ADALINE Model	482
17.4	Training of the Proposed Neural Net	484
17.4.1	Training Algorithm of ADALINES	484
17.4.2	Training of the Neural Net	487

---

17.4.3	Training with Multiple Input-Output Patterns	488
17.5	Translation Rotation and Size Invariant Gray Pattern Recognition	490
17.5.1	Design of Translational Invariance Network	491
17.5.2	Design of Rotational Invariance Network	492
17.5.3	Design of Size Invariance Network	493
17.6	Conclusions	493
	Exercise	494
	References	495

## **Chapter 18: Distributed Machine Learning Using Fuzzy Cognitive Maps**

18.1	Introduction	497
18.2	Axelrod's Cognitive Maps	498
18.3	Kosko's Model	500
18.4	Kosko's Extended Model	503
18.5	Adaptive FCMs	504
18.6	Zhang, Chen and Bezdek's Model	505
18.7	Pal and Konar's Model	507
18.8	Conclusions	513
	Exercise	513
	References	517

## **Chapter 19: Machine Learning Using Fuzzy Petri Nets**

19.1	Introduction	521
19.2	The Proposed Model for Cognitive Reasoning	523
19.2.1	Encoding of Weights	524
19.2.2	The recall Model	524
19.3	State-Space Formulation	526
19.3.1	State-Space Model for Belief Updating	527
19.3.2	State-Space Model for FTT Updating of Transitions	527
19.3.3	State-Space Model for Weights	528
19.4	Stability Analysis of the Cognitive Model	528
19.5	Computer Simulation	532
19.6	Implication of the Results	535
19.7	Knowledge Refinement by Hebbian Learning	535
19.7.1	The Encoding Model	535
19.7.2	The Recall/ Reasoning Model	537
19.7.3	Case Study by Computer Simulation	537
19.7.4	Implication of the Results	542
19.8	Conclusions	542
	Exercise	544
	References	545

**Chapter 20: Computational Intelligence in Tele-communication Networks**

- 20.1 Introduction 547
- 20.2 Network Routing Using Genetic Algorithms 549
  - 20.2.1 Path-Genetic Operators 551
  - 20.2.2 Fitness Evaluation 553
  - 20.2.3 The Genetic Based Routing Algorithm 553
- 20.3 Computational Intelligence in Network Congestion Control 554
  - 20.3.1 Fuzzy Congestion Controller 555
- 20.4 Computational Intelligence in Handling the Call Admission Control Problem 560
  - 20.4.1 Call Admission Control Using Neural Networks 561
  - 20.4.2 Input/Outputs of Neural Nets Used in Call Admission Control 562
- 20.5 Intelligent Traffic Control 565
- 20.6 Conclusions 566
  - Exercise 567
  - References 570

**Chapter 21: Computational Intelligence in Mobile Robotics**

- 21.1 Introduction 571
- 21.2 Path Planning of a Mobile Robot Using Neural Nets 573
  - 21.2.1 Generation of Training Instances 574
  - 21.2.2 Configuring the Neural Nets 580
  - 21.2.3 Parameters Used for Performance Evaluation of the Neural Path-Planning Algorithms 582
  - 21.2.4 Experimental Results- a Benchmark Analysis 584
  - 21.2.5 Implication of the Results 589
- 21.3 Object Localization 589
- 21.4 Target Tracking and Interception by Mobile Robots Using Kalman Filtering 592
  - 21.4.1 Measurements of the Input to Kalman Filter 595
  - 21.4.2 Extended Kalman Filter- an Overview 596
  - 21.4.3 Predicting Target Position Using Extended Kalman Filter 597
  - 21.4.4 Use of the Back-propagation Neural Net 600
  - 21.4.5 Experimental Results 600
- 21.5 Conclusions and Future Directions 601
  - Exercise
  - References

---

**Chapter 22: Emerging Areas of Computational Intelligence**

- 22.1 Introduction 611
- 22.2 Artificial Life 613
  - 22.2.1 The Model of Artificial Fish Positions, Speed and Heading 613
  - 22.2.2 Repulsion, Attraction and Aligning 614
  - 22.2.3 Body Size and Form 616
- 22.3 Particle Swarm Optimization 617
  - 22.3.1 Particle Swarm Size 619
  - 22.3.2 The Split Swarm Algorithm 619
- 22.4 Artificial Immune Systems 620
  - 22.4.1 Biological Communication among Different Species of Antibodies 621
  - 22.4.2 A Simple Model of the Immune System 622
- 22.5 Fuzzy Chaos Theory 624
- 22.6 Rough Sets 627
  - 22.6.1 Rough-Fuzzy Sets 629
- 22.7 Granular Computing 630
  - 22.7.1 Rough Inclusion and Indiscernibility 632
  - 22.7.2 Classifier Design 632
- 22.8 Redefining Computational Intelligence 633
- 22.9 Summary 634
  - Exercise 635
  - References 640

**Chapter 23: Research Problems for Graduate Thesis and Pre-Ph D Preparatory Courses**

- 23.1 Problem 1: Computing Max-Min Inverse Fuzzy Relation 643
- 23.2 An Outline to the Solution of Problem 1 644
- 23.3 Tutorial Assignment for Graduate Students 645
- 23.4 Problem 2: Reasoning with Fuzzy Petri Nets 646
  - 23.4.1 Tutorial Problems on Fuzzy Petri Nets 646
  - 23.4.2 Research Problems on Fuzzy Petri Nets 647
- 23.5 Problem 3: Spoken Word Reconstruction from Lip Stylus Sequences 649
- 23.6 Problem 4: Fuzzy State Equations and stability Analysis 650
- 23.7 An Outline to the Solution of Problem 4 651
- 23.8 Graduate Students' Assignments on Stability of Fuzzy State Equations 652
- 23.9 Problem 5: Fuzzy Data Mining 652
- 23.10 An approach to the Solution of Problem 5 653
- 23.11 Graduate Assignments on Fuzzy Data Mining 655
- 23.12 Problem 6: Selected Items in Signal Processing and Communication Engineering 655

23.12.1 System Modeling	656
23.12.2 Non-linear Prediction Problem	657
23.12.3 Inverse Modeling	658
23.12.4 Adaptive Channel Equalization	659
23.13 Problem 7: Handling Large Training Instances Using Back-Propagation-RBF Synergism	660
23.14 Problem 8: Macro Cell Placement and Routing in VLSI Design Using Genetic Algorithms	662
23.15 A Possible Solution to Macro Cell Placement and Routing	663
23.16 Problem 9: Drug Classification Problem Using Artificial Neural Networks	664
References	667

## **Appendix A: Sample Run of Programs Included in the CD**

A.1 Detection of Mouth Opening of a Person from his/ her Facial Image	669
A.2 A Program for Image Matching Using Fuzzy Moment Descriptors	674
A.3 GA in Path Planning of a Mobile Robot	676
A.4 The FPNreas1 Program	677
A.5 The FPNreas2 Program	683
A.6 The FPNreas3 Program	684
A.7 Fuzzy Chaos Program	686
A.8 The New_Petri3 Program to Study Controllability on FPN	688

## **Appendix B: Evolutionary Algorithms of Current Interest**

B.1 Ant Colony Systems: an Overview	691
B.1.1 The ACO Algorithm	693
B.1.2 Solving the classical TSP Problem by ACO	693
B.1.3 Application Domain of ACO	695
B.2 Differential Evolution	696
B.2.1 Outline of the Algorithm	696
B.2.2 Application Domain of DE and Recent Research	698
References	698

**Index** 701

**About the Author** 707

# 1

# An Introduction to Computational Intelligence

*The chapter provides an introduction to computational intelligence. It begins with a thorough review of the underlying principles of artificial intelligence, and examines the scope of computational intelligence in overcoming the limitations of the traditional AI. The chapter then briefly introduces various tools of computational intelligence such as fuzzy logic, neural network, genetic algorithm, belief network, chaos theory, computational learning theory and artificial life. The synergistic behavior of the above tools on many occasions far exceeds their individual performance. A discussion on the synergistic behavior of neuro-fuzzy, neuro-GA, neuro-belief and fuzzy-belief network models is also included in the chapter. A list of tutorial problems is appended at the end of the chapter to build up students' ability in handling real world problems.*

## **1.1 Artificial Intelligence- a Brief Review**

Artificial intelligence (AI) aims at emulating human intelligence on machines so as to enable these act and think like the human beings. It is a vast discipline of

knowledge that includes reasoning, machine learning, planning, intelligent search and perception building.

Reasoning attempts to satisfy a prescribed goal of a problem using a set of supplied facts and a pre-defined knowledge base. The knowledge base consists of a set of IF-THEN rules or graph-theoretic structures representing the knowledge of skilled persons in a specialized domain. An interpreter or inference engine is used to select the appropriate rule for firing at a given situation, and reasoning is continued in the direction of the prescribed goal.

Learning is an inherent instinct of animals. Because of learning animals can act properly in known situations and inadvertently in unknown situations. But how do people perform correct actions in known situations? Psychologists believe that people memorize situation-action pairs in their brain [40]. Learning thus can be defined as a process of encoding the situation-action pairs on memory, so that on excitation of appropriate situation the memory should be able to recall the correct actions.

Machines too can be empowered with the skill of learning. But how can we represent learning in machines? The process of learning is realized on machines either by training the machine with known situation-action pairs or by allowing the machine to adapt the parameters of a prescribed learning rule in a trial sense. The trial adaptation of the parameters of a learning rule definitely requires enormous amount of time. To save learning epochs sometimes a penalty-reward scheme is incorporated so as to make the machines aware of its performance in trial adaptations.

Planning is concerned with determining the sequence of steps in solving a problem. To be more specific, given a knowledge base and a set of facts, planning calls for sequencing the step of rule firing, so that at least one such sequence leading to the goal exists. In case optional sequences leading to the goal are available, strategies are used to identify the best sequence.

Most of the classical AI problems are formulated as a state-space search problem [65]. The term "state" usually refers to a particular problem instance, and the term "space" stands for collection of states. Usually, one or more initial states are first defined arbitrarily as a guess to the solutions. If the guess is same as the goal, there is no need for a move, otherwise a move in the state-space is caused by applying operators to the existing states. The new states thus generated are compared with the supplied goal. If the goal is detected among the new states, then the search process is terminated, else the new states are defined as the existing state and the search is continued. Occasionally, the search cost is too expensive because of the presence of a series of operators on the path leading to the goal. Under such circumstance, the user is given the choice to terminate further search in the state-space.

Understanding scenes from images and sentences from voice/ text is a classical problem of human perception. Realizing the power of perception on machines requires a thorough formalization of the principles of 3D vision and natural language understanding. A machine with good level of visual perception requires relatively lessor sensors for understanding a scene. Similarly, a machine with a good level of linguistic perception can recover missing words/ phrases in a given sentence. Perception building is currently gaining importance for its increasing applications in mobile robotics, language translation and other on-line man-machine interfaces.

There exist a vast literature on AI [29-33], [50], [54], [65] that covers various techniques of knowledge representation, reasoning, [72-73] machine learning, image and language understanding [8], planning, intelligent search and their realizations [8], [38], [66]. A detailed discussion on these topics goes beyond the scope of this book. Interested readers may consult any standard textbook [40] for a comprehensive discussion on the above topics.

## 1.2 Pitfalls of the Traditional AI

Traditional problem solving methods in AI are concerned with representation of the problem states by symbols, and construction of a set of rules to describe the transitions in problem states. The states of the problem are then matched against the IF part of the IF-THEN rules, and on successful matching the selected rule is fired causing a transition of the existing state to a new state as obtained from the THEN part of the fired rule. To keep the rules firable until the goal is found, the knowledge base is usually enriched with a large number of rules. A large knowledge base, however, calls for more search time and thus is responsible for degradation in efficiency of a reasoning system. One approach to circumvent this problem is to organize the knowledge base with fewer rules but to allow **partial matching** of the problem states with the IF-part of the rules. The *logic of fuzzy sets* [34], [62], [86-88] that we shall introduce in the next chapter is capable of such partial matching.

Traditional AI is very good in inductive [61] and analogy-based learning [85] but it is inefficient to realize supervised learning [63-64], [80-83]. In supervised learning, the trainer provides a number of training input/ output instances for the learning system. The learning system has to adapt its internal parameters so as to generate the correct output instance in response to a given input instance. *Neural network models* [2], [5], [10-12], [21], [23-24], [35-36] that we shall introduce in this book can perform supervised learning very well. It may be added here that approximately 2/3-rd of the commercial applications of machine learning fall within the category of supervised learning.

Except the heuristic search algorithm, traditional AI is not much competent to handle real world optimization problems. But the need for optimization of

system parameters and resources in design, synthesis, diagnosis and scheduling problems, for example, is gradually increasing. *Genetic algorithm* (GA) [49-52], [75], [78-79], fortunately, is a new tool that has a good potential in optimizing the parameters of intelligent systems. It has proved itself successful in optimization of rules in an expert system [40], tuning of parameters [48] in a learning system for an optimal system performance and determining the optimal order of events in a scheduling problem [40]. The complete list of application areas of GA will be as large as this book itself, some of which will be discussed in the subsequent parts of the book.

Decision making in a real-time system largely depends on the available facts, their level of precision, the knowledge base and its soundness. A sound knowledge base should not generate false inferences, but the degree of precision of the inferences may degrade when the input facts are less precise. However, when the facts are available from multiple sources, their level of precision can be improved through a process of **data fusion**. Traditional AI is not concerned with data fusion. Fortunately, there is a vast contemporary literature on data fusion technology [1]. The classical *Bayesian statistics* [18], *Dempster-Shafer theory* [68-70], *Pearl's belief networks* [56-57], *Kalman filtering* [9] and *neural network* based methods [28] are some of the well-known techniques of data fusion. These tools and techniques have successfully been used in many industrial autonomous systems. For example, noisy sensory data received by various transducers of a mobile robot are fused to take decisions about its direction of motion in a constrained obstacle-map [40]. The other application areas of data fusion technique include object recognition from multi-sensory noisy data, prediction of earthquake, storm or heavy rain from multi-sensory measurements and automated diagnosis of a complex system from noisy measurements.

### **1.3 Computational Intelligence- an Emergence of New Computational Paradigms**

Traditional AI was incompetent to serve the increasing demand of search, optimization and machine learning in i) information systems with large biological and commercial databases and ii) factory automation for steel, aerospace, power and pharmaceutical industries. The shortcomings of AI became more and more pronounced with successive failures of the decade long Japanese project on *Fifth Generation Computing Machines*. Almost at the same time, the contemporary models of non-traditional machine intelligence such as rough sets, fuzzy logic, artificial neural networks, genetic algorithms, belief networks, computational learning theory and chaos theory could prove their theoretical basis. The failure of classical AI opened up new avenues for the non-conventional models in various engineering applications. These computational tools gave rise to a new discipline called **computational intelligence**.

To understand the scope of this new discipline let us first have a look at the definitions. Prof. James (Jim) Bezdek, the father of fuzzy pattern recognition theory, defined a computationally intelligent system [7] as follows:

*"A system is **computationally intelligent** when it: deals with only numerical (low level) data, has pattern recognition components, does not use knowledge in the AI sense; and additionally when it (begins to) exhibit i) computational adaptivity, ii) computational fault tolerance, iii) speed approaching human-like turnaround and iv) error rates that approximate human performance."*

What does the above definition infer? It states that a computationally intelligent system should be characterized with the capability of computational adaptation, fault tolerance, high computational speed and less error-prone to noisy information sources. Computational adaptation means that the system should be capable of changing its parameters following some guidelines (of optimizing criteria) and depending on the temporal changes in its input and output instances. Most of the ANN models satisfy this characteristic. Computational fault tolerance is a general characteristic of a parallel and distributed system. In a parallel and/ or distributed system, computational resources like variables, procedures and software tools are usually replicated at the distributed units of the computing system. So, a damage of a few units usually does not cause malfunctioning of the entire system, as the same resources are available at other units. While ANN and fuzzy logic have their inherent characteristics of fault tolerance, GA and belief networks too can be configured in a parallel fashion to provide users the benefit of fault tolerance.

The other important aspects of Bezdek's definition include high computational speed and less error rates like the human beings. It is true that a high computational speed may sometimes yield a poor accuracy in the results. Fuzzy logic and neural nets that support a high degree of parallelism usually have a fast response to input excitations. Further, unlike conventional production (rule based) system where a single rule only is fired at a time, fuzzy logic allows firing of a large number of rules ensuring partial matching of the available facts with the antecedent clauses of those rules. Thus the reasoning capability of fuzzy logic is humanlike, and consequently it is less error-prone. An ANN also allows firing of a number of neurons concurrently and thus has a high computational speed; it usually adapts its parameters by satisfying a set of constraints that minimizes the error rate. The parallel realization of GA and belief networks for the same reason have a good computational speed, and their inherent information filtering behavior maintain accuracy of their resulting outcome.

The first clause of Bezdek's definition emphasizes that a computationally intelligent system should deal with low level (numerical) information, should include a pattern recognition component and should not use knowledge in the AI

sense. Since fuzzy logic, ANN and GA deal with numerical data, and all of them have the capability of pattern recognition and have a significant difference with respect to the traditional AI, they are undoubtedly the ideal members of the computational intelligence family. Belief networks that can also do some pattern recognition tasks and deal with low level numerical information, unfortunately, fall in the margin area of AI and computational intelligence. So, whether this can be regarded as a member of the computational intelligence family is questionable.

In an attempt to define computational intelligence [47], Robert J. Marks clearly mentions the name of the constituent members of the family. According to him

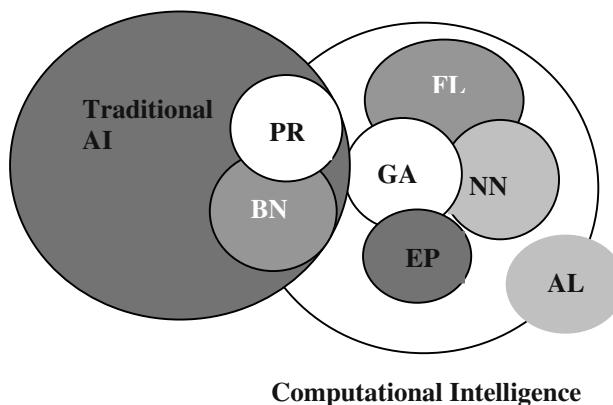
*"... neural networks, genetic algorithms, fuzzy systems, evolutionary programming and artificial life are the building blocks of computational intelligence."*

Out of these 5 members the last 2 emerged very recently and require a brief introduction. Roughly speaking, an **evolutionary programming** [13-17], [19-20] is a new programming paradigm where a number of individual computer programs evolve using the concepts of genetic algorithm. Devised by Koza [44] in 1992, this is a challenging area of research and has drawn interest of a good many number of researchers. An **artificial life** is also an emerging discipline that is based on the assumption that physical and chemical laws are good enough to explain the intelligence of the living organisms. Langton defines artificial life [45] as

*"....an inclusive paradigm that attempts to realize lifelike behavior by imitating the processes that occur in the development or mechanics of life."*

Let us now summarize what exactly we understand by the phrase computational intelligence. Fig.1.1 outlines the topics that share some ideas of this new discipline.

It is clear from the Fig. 1.1 that the primary constituents of computational intelligence are fuzzy logic, ANN and GA. Belief networks and probabilistic reasoning fall in the intersection of traditional AI and the computational intelligence. Computational intelligence and the physico-chemical laws share artificial life. The definition of computational intelligence, however, needs more formalization.



PR= Probabilistic reasoning, BN= Belief networks, FL= Fuzzy logic, NN= Neural nets, GA= Genetic algorithms, EP= Evolutionary programming, AL= Artificial life.

**Fig. 1.1:** The building blocks of computational Intelligence. It includes GA, FL, NN and EP. AL is shared by Physico-chemical laws (not shown in the figure) and computational intelligence jointly. PR and BN fall in the intersection of traditional AI and computational intelligence.

## 1.4 Computational Intelligence- a Formal Definition

Bezdek [7], Marks [47], Pedrycz [58-59] and others have defined computational intelligence in different ways depending on the then developments of this new discipline. A intersection of these definitions will surely focus to fuzzy logic, ANN and GA, but a union (and generalization) of all these definitions includes many other subjects like **rough set theory**, **chaos theory** and **computational learning theory**. Further, computational intelligence being an emerging discipline should not pinpoint to a limited number of topics only, it should rather have a scope to expand in diverse directions and to merge with other existing disciplines. Keeping this in mind, we can define computational intelligence *as the computational models and tools of intelligence capable of inputting raw numerical sensory data directly, processing them by exploiting the representational parallelism and pipelining of the problem, generating reliable & timely responses and withstanding high fault tolerance*.

The above definition emphasizes 4 major aspects. First a computationally intelligent system should be able to receive and interpret raw sensory data. This tests the capability of the system to handle numerical data. Secondly, it must have the capability to detect the possible pipelining and parallelism in the underlying problem. For example, consider a simple problem of computing  $z =$

( $a + b$ )/ ( $a - b$ ) with real data a and b. Here, the system must first identify from the problem definition that the computation of ( $a + b$ ) and ( $a - b$ ) can be carried out in parallel, and the computation of the result z can be performed in a pipeline following the computation of ( $a + b$ ) and ( $a - b$ ). Fuzzy and neural systems have their inherent parallelism and pipelining and thus they can process data at a high speed. The third issue considered in the above definition is the power of generating reliable and timely responses. This in other words means that the computational errors should be less and the responses should be timely. The last issue refers to high fault tolerance, which in other words require multiplicity of computational resources in the realization of the computational models.

## 1.5 Soft Computing- Definitions

Shortly after the birth of computational intelligence in early 90's, the researchers took much interest in studying the synergistic behavior of the underlying computational tools. Being inspired by the synergism of the computational tools Prof. Zadeh in 1992 coined a name- **Soft Computing** to this emerging discipline. In Zadeh's own words:

1. *"Soft computing is an emerging approach to computing which parallels the remarkable ability of the human mind to reason and learn in an environment of uncertainty and imprecision" [88].*
2. *"Soft computing is not a homogeneous body of concepts and techniques. Rather it is a collection of methodologies, which in one form or another reflect the guiding principle of soft computing: exploit the tolerance for imprecision, uncertainty, and partial truth to achieve tractability, robustness, and low solution cost. Viewed in a slightly different perspective, soft computing is a consortium of methodologies which, either singly or in combination, serve to provide effective tools for the development of intelligent systems" [58].*
3. *.....a recent trend to view fuzzy logic (FL), neurocomputing (NC), genetic computing (GC) and probabilistic computing (PC) as an association of computing methodologies falling under the rubric of so-called soft computing. The essence of soft computing is that its constituent methodologies are for the most part complementary and synergistic rather than competitive. A concomitant of the concept of soft computing is that in many situations it is advantageous to employ FL, NC, GC and PC in combination rather than isolation [59].*

It is thus clear from the above definitions that in case a single computational tool is used in solving a problem, soft computing is same as computational intelligence. But the basic difference between Soft computing and computational intelligence lies in the synergistic power of the former through fusion of 2 or

more computational models/techniques. Let us now briefly review the individual members of soft computing and their possible forms of synergism.

## 1.6 Fundamental Elements of Soft Computing

In this section we briefly outline the common soft-computing models based on their fundamental characteristics.

### 1.6.1 The Logic of Fuzzy Sets

Proposed by Zadeh in 1965 [87], this logic is an extension of the classical propositional and predicate logic that rests on the principles of the binary truth functionality. For instance, let us consider the well-known *modus ponens* rule in propositional logic, given by

$$\begin{array}{c} p \\ p \rightarrow q \\ \hline q \end{array}$$

where  $p$  and  $q$  are binary propositions (i.e. facts with binary truth values) and  $\rightarrow$  denotes a 'if-then' operator. The rule states that given  $p$  is true and 'if  $p$  then  $q$ ' is true, it can be inferred that  $q$  is true. But what happens when the truth or falsehood of  $p$  cannot be guaranteed? There are three situations that may arise in this context. First, we may take support or refutation of other statements in the system, if any, to determine the truth or falsehood of  $p$ . If majority support for  $p$  is obtained, then  $p$  may be accepted to be true, else  $p$  is assumed to be false. This idea led to the foundation of a new class of logic, called **non-monotonic logic** [3]. But what happens when the exact binary truth functional value of  $p$  cannot be ascertained. This called for the other two situations. Suppose  $p$  is partially true. This can be represented by a truth functional value in between 0 and 1, and this idea was later formalized as the basis of **multi-valued logic**. The second situation thus presumes a non-binary truth functional value of  $p$  and attempts to infer a truth functional value of  $q$  in the range [0,1]. Now, consider the third situation where a fact approximately same as  $p$ , called  $p'$ , is available and the nearest rule whose antecedent part partially matches with  $p'$  is  $p \rightarrow q$ . Thus, formally

$$\begin{array}{c} p' \\ p \rightarrow q \\ \hline q' \end{array}$$

where  $q'$  is the inferred consequence. This partial matching of  $p'$  with  $p$ , and thus generating the inference  $q'$  comes under the scope of **fuzzy logic**. The truth functional value of the propositions here lies in the same interval of [0,1] like that of multi-valued logic. Thus in one sense fuzzy logic is a multi-valued logic. But the most pertinent feature of fuzzy logic for which it receives so much attention is its scope of **partial matching**, as illustrated with the last example. Another interesting situation occurs when a number of rules' antecedent parts match partially with the fact  $p'$ . The resulting inference, say  $q_{final}$ , under this circumstance is determined by the composite influence of all these rules. So, in any real world system, the inferences guided by a number of rules follow a middle decision trajectory over time. This particular behavior of following a middle decision trajectory [41] is humanlike and is a unique feature of fuzzy logic, which made it so attractive!

Very recently Prof. Zadeh highlighted another important characteristic [59] of fuzzy logic that can distinguish it from other multi-valued logics. He called it **f.g- generalization**. According to him any theory, method, technique or problem can be fuzzified (or **f-generalized**) by replacing the concept of a crisp set with a fuzzy set. Further, any theory, technique, method or problem can be granulated (or **g-generalized**) by partitioning its variables, functions and relations into granules (information cluster). Finally, we can combine f-generalization with g-generalization and call it f.g- generalization. Thus ungrouping a information system into components by some strategy and regrouping them into clusters by some other strategy can give rise to a new kind of information sub-systems. Determining the strategies for ungrouping and grouping, however, rests on the designer's choice. The philosophy of f.g- generalization undoubtedly will re-discover fuzzy logic in a new form.

## 1.6.2 Computational Models of Neural Nets

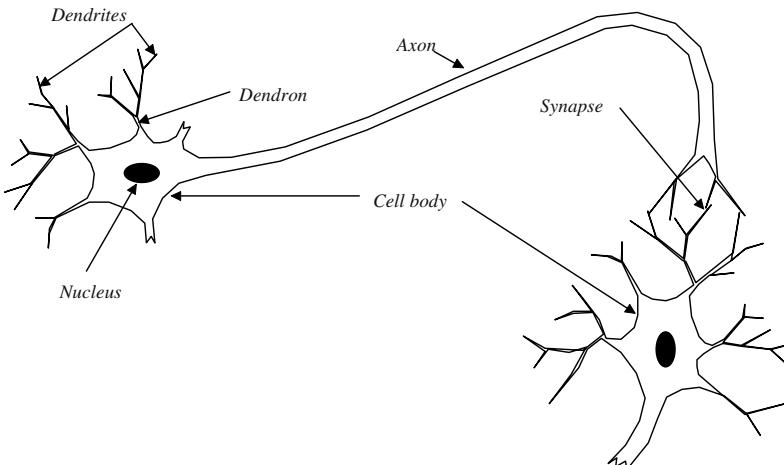
Neurons are the fundamental building blocks of a biological nervous system. A neuron has four main structural components: [53], [67] the dendrites, the cell body, the axon and the synapse. The dendrites act as receptors, thereby receiving signals from several neighboring neurons and passing these on to a little thick fiber, called dendron. In other words, dendrites are the free terminals of dendrons. The received signals collected by different dendrons are processed within the cell body and the resulting signal is transferred through a long fiber called axon. At the other end of the axon, there exists an inhibiting unit called synapse. This unit controls the flow of neuronal current from the originating neuron to receiving dendrites of neighborhood neurons. A schematic diagram, depicting the above concept is presented in Fig.1.2.

An artificial neural net is an electrical analogue of a biological neural net [23]. The cell body in an artificial neural net is modeled by a linear activation function. The *activation function* in general, attempts to enhance the signal

contribution received through different dendrons. The axon behaves like a signal conductive (resistive) device. The synapse in the artificial neural net is modeled by a non-linear *inhibiting function*, for limiting the amplitude of the signal processed at cell body.

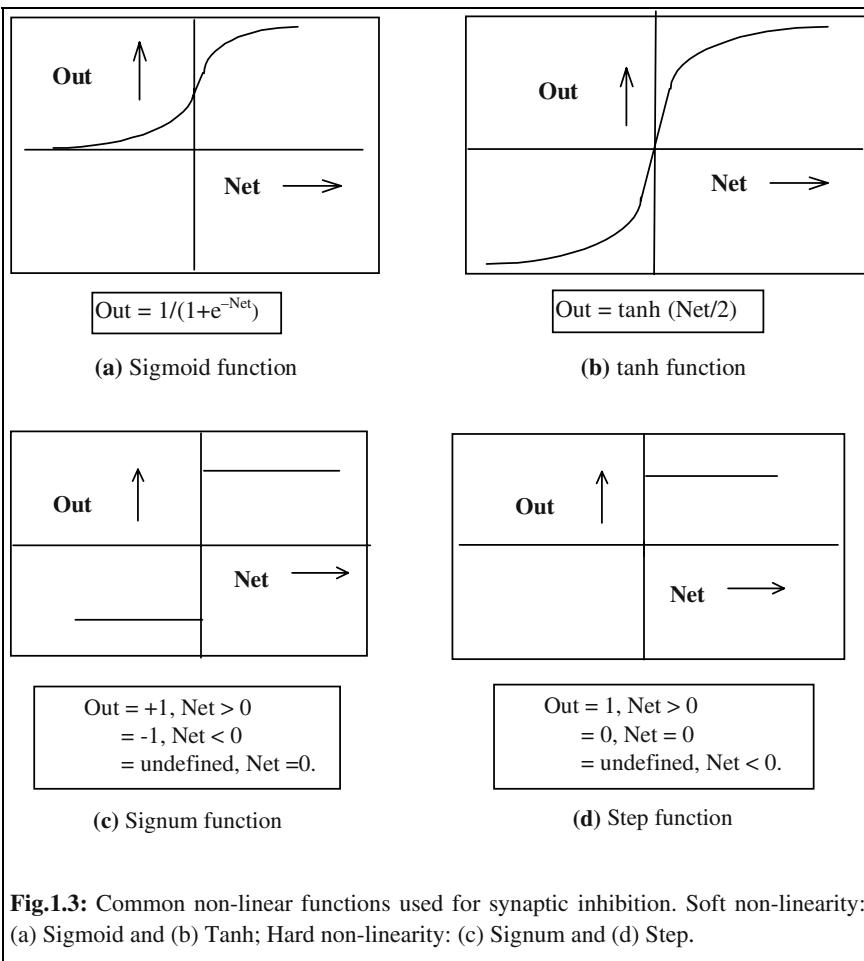
The most common non-linear functions used for synaptic inhibition are:

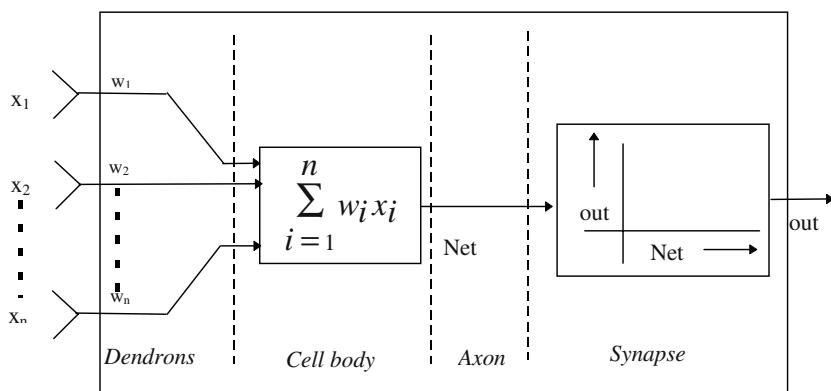
- sigmoid function
- tanh function
- signum function
- step function.



**Fig. 1.2:** A biological neuron comprising of a cell body, several dendrons, an axon and a synapse. The terminals of a dendron are called dendrites. The dendrites of a neuron transfer signal to dendrites of a second neuron through a synaptic gap.

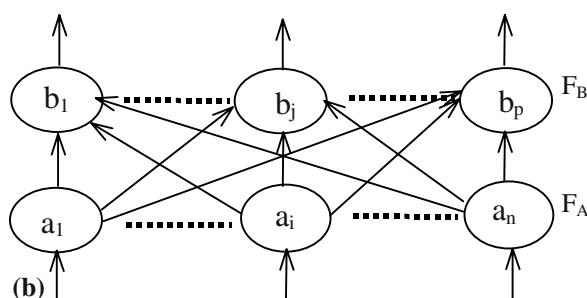
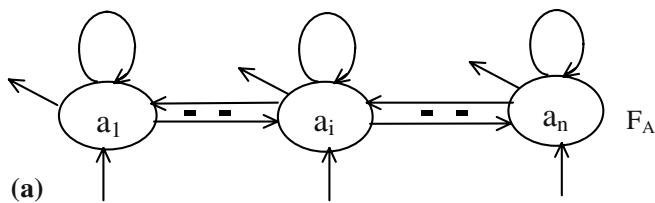
Sigmoid and tan hyperbolic (tanh) functions are grouped under soft non-linearity, whereas signum and step functions under hard type non-linearity. These functions are presented graphically in Fig. 1.3 for convenience. The schematic diagram of an artificial neuron, based on the above modeling concept is presented in Fig. 1.4.

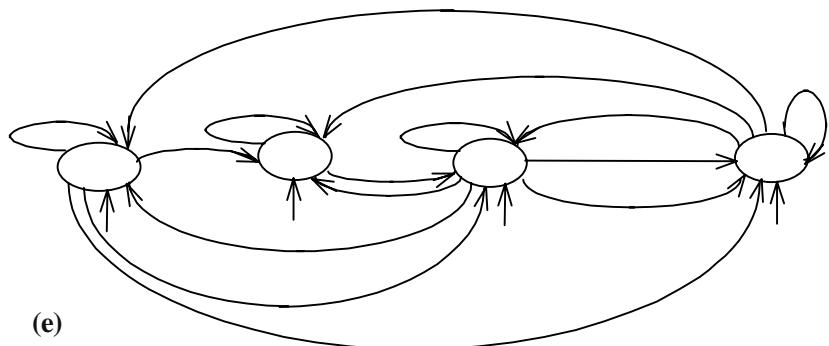
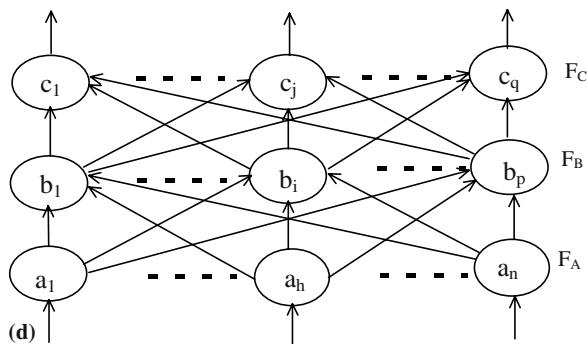
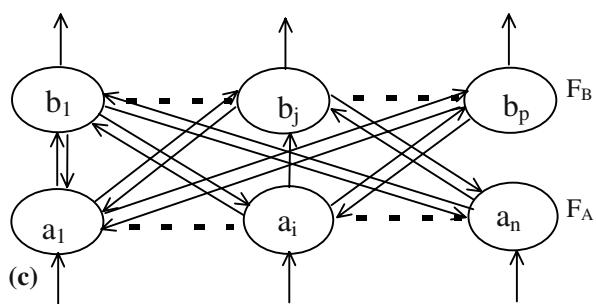




**Fig. 1.4:** An electrical equivalent of the biological neuron.

Depending on the nature of problems, the artificial neural net is organized in different structural arrangements (topologies). Common topologies (vide Fig. 1.5) are:





**Fig.1.5:** Common topologies of Artificial neural net: (a) single layered recurrent net with lateral feedback, (b) two layered feed-forward structure, (c) two layered structure with feedback, (d) three layered feed-forward structure, (e) a recurrent structure with self-loops.

- Single layered recurrent net with lateral feedback structure,
- Two layered feed-forward structure,
- Two layered feedback structure,
- Three layered feed-forward structure and
- Single layered recurrent structure.

The single layered recurrent net with lateral feedback topology was proposed by Grossberg [22], which has successfully been applied for classifying analog patterns. The feed-forward neural nets are most common structures for the well-known back-propagation algorithm [63]. Two layered feedback structure, on the other hand have been used by Carpenter and Grossberg [10] for realization of adaptive resonance theory [11] and by Kosko [42-43] for realization of bi-directional associative memory. The last class of topology shown in Fig. 1.5 (e) represents a recurrent net with feedback. Many cognitive nets [39] employ such topologies. Another interesting class of network topologies, where each node is connected to all other nodes bi-directionally and there is no direct self-loop from a node to itself, has been used by Hopfield in his studies [26-27]. We do not show the figure for this topology, as the readers by this time can draw it themselves for their satisfaction.

Fuzzy logic, which has proved itself successful especially in reasoning and unsupervised classification, however, is not directly amenable for automated machine learning [50]. Artificial neural nets (ANN) on the other hand can learn facts (represented by patterns) and determine the inter-relationship among the patterns. The process of determining the interrelationships among patterns, in general, is informally called **encoding**. Some ANN parameters, usually the weights and the thresholds, are generally encoded to represent the interrelationship among the patterns. The process of determining the inter-related pattern(s) from one given pattern using the encoded network parameters is usually called **recall**. Depending on the type and characteristics of the encoding and recall process, machine learning techniques are categorized into four major heads: i) supervised learning, ii) unsupervised learning, and iii) reinforcement learning and competitive learning.

In a **supervised learning** system, there is a trainer who provides the input and the corresponding target (output) patterns. A learning algorithm is employed to determine a unique set of network parameters that jointly satisfy the input-output interrelationship of each two patterns. After the encoding process discussed above is over the network on excitation [55] with an unknown input pattern can generate its corresponding output pattern. The ANN when trained by

a supervised learning algorithm, thus, behaves like a multi-input-multi output function approximator.

A common question that naturally arises: how does a supervised learning system actually work? Generally, the network is initially assigned a random set of parameters such as weights and thresholds. Suppose that we want to train the network with a single input-output pattern. What should we do? We supply the network with our input pattern, and let the network generate its output pattern. The generated output pattern is compared with the target pattern. The difference of these two patterns result in an error vector. A supervised learning algorithm is then employed to adjust the network parameters using the error vector. For multiple input-output patterns, the error vector for each pattern set is determined and a function of these error vectors is used to adjust the network parameters. There exist quite a large number of supervised learning algorithms using neural nets. The most popular among them is the Back-propagation algorithm. We shall discuss this algorithm later.

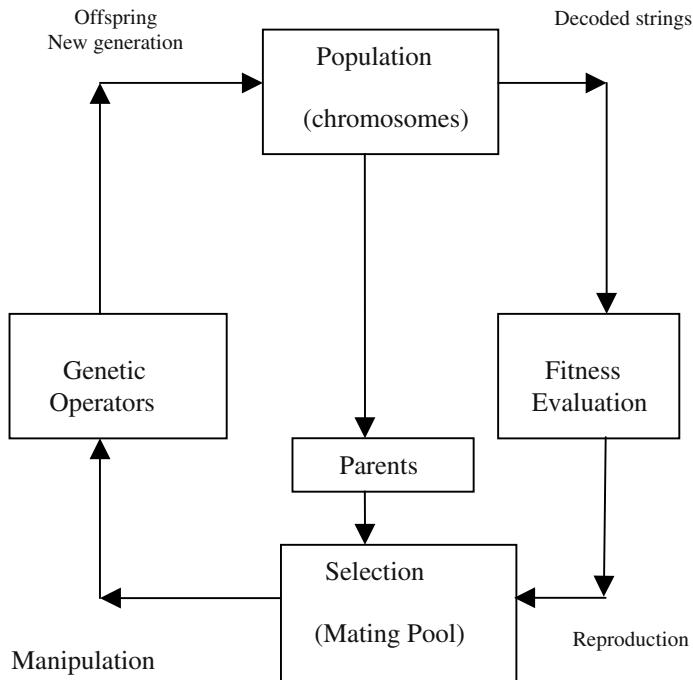
An **unsupervised learning system** employs no teacher, and thus the interrelation among the patterns is not known. Generally in an unsupervised learning process, one or more input patterns is automatically mapped to one pattern cluster. The encoding process in an unsupervised learning system varies greatly from one system to another. Most systems, however, employ a recursive learning rule that autonomously adjusts the network parameters for attaining some criteria like minimization of the network energy states. Among the unsupervised learning systems, Hopfield nets and associative memory are most popular. We shall discuss these learning systems in a separate chapter later.

The third category of the learning system that bridges the gap between the supervised and the unsupervised learning is popularly known as the **reinforcement learning** [84]. This learning scheme employs an **internal critic** that examines the response of the environment in turn of the action of the learning system on the environment. If the response is in favor of the goal, then the action is **rewarded** otherwise it is **penalized**. Determination of the status of the action: reward or penalty may, however, require quite a long time, until the goal is reached. Q-learning is one of the most common reinforcement learning techniques, and we will shall take up the issues of Q-learning in a separate chapter.

The last category of learning is well known as **competitive learning**. Neurons in this scheme compete with one another to satisfy a given goal. One typical scheme of competitive learning will be taken up in a separate chapter.

### 1.6.3 Genetic Algorithms

A Genetic algorithm (GA) is a stochastic algorithm [4] that models the evolutionary process of biological species through natural selection. Proposed by Holland [25] in early 60's, this algorithm is gaining its importance for its wide acceptance in solving three classical problems, such as learning, search and optimization. A number of researchers throughout the world have developed their own ways to prove the convergence of the algorithm. Among these the work by Goldberg [22], De Jong [17], Davis [16], Muehlenbein [52], Chakraborti [13-15], Fogel [53], and Vose [78-79] need special mention. In this section we shall briefly outline this algorithm with an example.



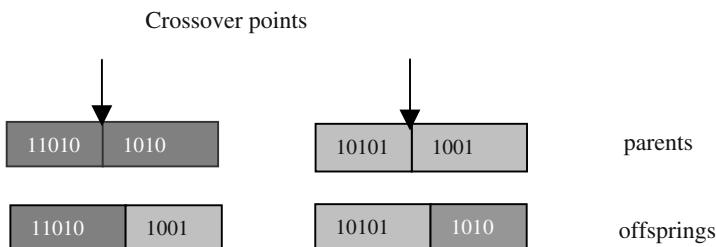
**Fig. 1.6:** The cycle of genetic algorithms.

A GA operates through a simple cycle of stages [54]:

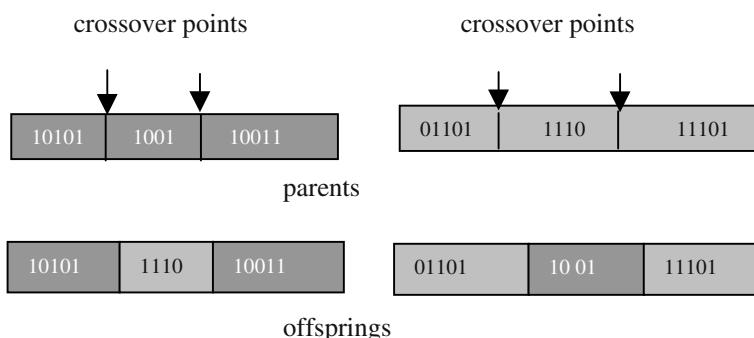
- i) Creation of a “population” of strings,
- ii) Evaluation of each string,
- iii) Selection of best strings and
- iv) Genetic manipulation to create new population of strings.

The cycle of a GA is presented in Fig. 1.6.

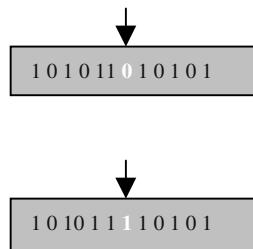
Each cycle in GA produces a new generation of possible solutions for a given problem. In the first phase, an initial population, describing representatives of the potential solution, is created to initiate the search process. The elements of the population are encoded into bit-strings, called chromosomes. The performance of the strings, often called fitness, is then evaluated with the help of some functions, representing the constraints of the problem. Depending on the fitness of the chromosomes, they are selected for a subsequent genetic manipulation process. It should be noted that the **selection** process is mainly responsible for assuring survival of the best-fit individuals. After selection of the population strings is over, the genetic manipulation process consisting of two steps is carried out. In the first step, **crossover** operation that recombines the bits (genes) of each two selected strings (chromosomes) is executed. Various types of crossover operators are found in the literature. The single point and two points crossover operations are illustrated in Fig. 1.7 and 1.8 respectively. The **crossover points** of any two chromosomes are selected randomly. The second step in the genetic manipulation process is termed **mutation**, where the bits at one or more randomly selected positions of the chromosomes are altered (Fig. 1.9). The mutation process helps to overcome trapping at local maxima. The offsprings produced by the genetic manipulation process are the next population to be evaluated.



**Fig. 1.7:** A single point crossover after the 3-<sup>rd</sup> bit position from the L.S.B.



**Fig. 1.8:** Two point crossover: one after the 4<sup>th</sup> and the other after the 8<sup>th</sup> bit positions from the L.S.B.



**Fig. 1.9:** Mutation of a chromosome at the 5<sup>th</sup> bit position.

**Example 1.1:** The GA cycle is illustrated in this example for maximizing a function  $f(x) = x^2$  in the interval  $0 \leq x \leq 31$ . In this example the fitness function is  $f(x)$  itself. The larger is the functional value, the better is the fitness of the string. In this example, we start with 4 initial strings. The fitness value of the strings and the percentage fitness of the total are estimated in Table 1.1. Since fitness of the second string is large, we select 2 copies of the second string and one each for the first and the fourth string in the mating pool. The selection of the partners in the mating pool is also done randomly. Here in Table 1.2, we selected partner of string 1 to be the 2-nd string and partner of the 4-th string to be the 2nd string. The crossover points for the first-second and second-fourth strings have been selected after 0-th and 2-nd bit positions respectively in Table 1.2. The second generation of the population without mutation in the first generation is presented in Table 1.3.

**Table 1.1:** Initial population and their fitness values

string no.	initial population	x	f(x)	strength fitness (% of total)
1	01101	13	169	14.4
2	11000	24	576	49.2
3	01000	08	64	5.5
4	10011	19	361	30.9
Sum-fitness		1170	100.00	

**Table 1.2:** Mating pool strings and crossover

String No.	Mating Pool	Mates string	Swapping	New population
1	01101	2	0110[1]	01100
2	11000	1	1100[0]	11001
2	11000	4	11[000]	11011
4	10011	2	10[011]	10000

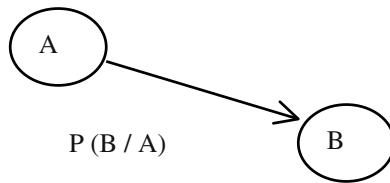
**Table 1.3:** Fitness value in second generation

Initial population	x	f(x) (fitness)	strength (% of total)
01100	12	144	8.2
11001	25	625	35.6
11011	27	729	41.5
10000	16	256	14.7
Sum-fitness =		1754	100.00

In this example, we did not consider mutation operation. In case, mutation is applied, we have to employ it after crossover. After computation of fitness by Table 1.3, we repeat the next GA cycle and continue until fitness of one chromosome comes within a desired tolerance level.

### 1.6.3 Belief Networks

A Bayesian belief network [56-57] is represented by a directed acyclic graph or tree, where the nodes denote the events and the arcs denote the cause-effect relationship between the parent and the child nodes. Each node, here, may assume a number of possible values. For instance, a node A may have n number of possible values, denoted by  $A_1, A_2, \dots, A_n$ . For any two nodes, A and B, when there exists a dependence  $A \rightarrow B$ , we assign a conditional probability matrix  $[P(B/A)]$  to the directed arc from node A to B. The element at the  $j^{\text{th}}$  row and  $i^{\text{th}}$  column of  $P(B/A)$ , denoted by  $P(B_j/A_i)$ , represents the conditional probability of  $B_j$  assuming the prior occurrence of  $A_i$ . This is described in Fig.1.10.



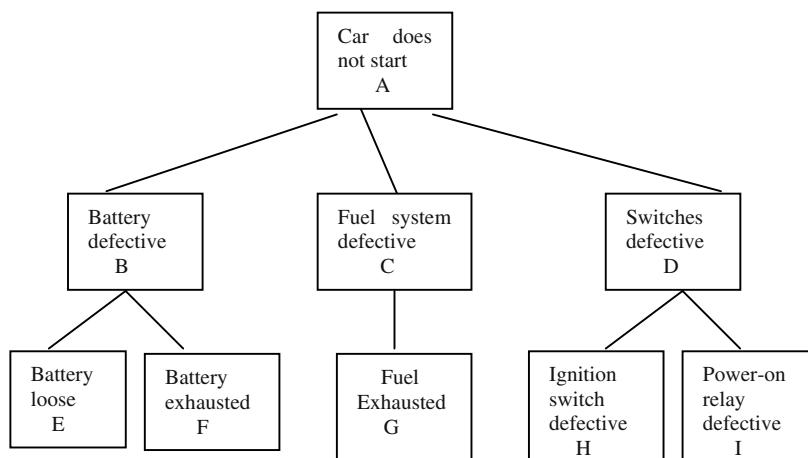
**Fig. 1.10:** Assigning a conditional probability matrix in the directed arc connected from A to B.

Given the probability distribution of A, denoted by  $[P(A_1) \ P(A_2) \ \dots \ P(A_n)]$ , we can compute the probability distribution of event B by using the following expression:

$$\begin{aligned}
 \mathbf{P(B)} &= [P(B_1) \ P(B_2) \ \dots \ P(B_m)]_{1 \times m} \\
 &= [P(A_1) \ P(A_2) \ \dots \ P(A_n)]_{1 \times n} \cdot [\mathbf{P(B / A)}]_{n \times m} \\
 &= [\mathbf{P(A)}]_{1 \times n} \cdot [\mathbf{P(B / A)}]_{n \times m}
 \end{aligned} \tag{1.1}$$

We now illustrate the computation of  $P(B)$  with an example.

**Example 1.2:** Consider a Bayesian belief tree describing the possible causes of a defective car.



**Fig. 1.11:** A diagnostic tree for a car.

Here, each event in the tree (Fig. 1.11) can have two possible values: true or false. Thus the matrices associated with the arcs will have dimensions (2 x 2). Now, given  $P(A) = [P(A = \text{true}) \ P(A = \text{false})]^T$ , we can easily compute  $P(B)$ ,  $P(C)$ ,  $P(D)$ ,  $P(E)$ , ...,  $P(I)$  provided we know the transition probability matrices connected with the links. As an illustrative example, we compute  $P(B)$  with  $P(B/A)$  and  $P(A)$ .

$$\text{Let } P(A) = [P(A = \text{true}) \ P(A = \text{false})]^T$$

$$= [ \begin{array}{cc} 0.7 & 0.3 \end{array} ]$$

		$B_j \rightarrow$	
		$B = \text{true}$	$B = \text{false}$
$P(B/A) = A = \text{true}$	$A = \text{true}$	0.8	0.2
	$A = \text{false}$	0.4	0.6

$$\text{So, } P(B) = P(A) \cdot P(B/A) = [ 0.68 \ 0.32 ]$$

One interesting property of Bayesian network is that we can compute the probability of the joint occurrence easily with the help of the topology. For instance, the probability of joint occurrence of A, B, C, D, E, F, G, H, I (see Fig. 1.11) is given by

$$\begin{aligned} & P(A, B, C, D, E, F, G, H, I) \\ &= P(A/B) \cdot P(A/C) \cdot P(A/D) \cdot P(B/E, F) \cdot P(C/G) \cdot P(D/H, I) \end{aligned} \quad (1.2)$$

Further, if E and F are independent, and H and I are independent, the above result reduces to

$$P(A/B) \cdot P(A/C) \cdot P(A/D) \cdot P(B/E) \cdot P(B/F) \cdot P(C/G) \cdot P(D/H) \cdot P(D/I).$$

Thus, given A,B,C,...,H all true except I, we would substitute the conditional probabilities for  $P(A = \text{true} / B = \text{true})$ ,  $P(A = \text{true} / C = \text{true})$ ....and finally  $P(D = \text{true} / I = \text{false})$  in the last expression to compute  $P(A = \text{true}, B = \text{true}, \dots, H = \text{true}, I = \text{false})$ .

Judea Pearl [56-57], [71] proposed a scheme for propagating beliefs of evidence in a Bayesian network. We shall first demonstrate his scheme with a Bayesian tree like that in Fig. 9.11. It may, however, be noted that like the tree of Fig. 9.11 each variable, say A, B,..., need not have only two possible values. For example, if a node in a tree denotes German Measles (GM), it could have three possible values like severe-GM, little-GM, moderate-GM.

In Pearl's scheme for evidential reasoning, he considered both the causal effect and the diagnostic effect to compute the **belief function** at a given node in the Bayesian belief tree. For computing belief at a node, say  $V$ , he partitioned the tree into two parts: i) the subtree rooted at  $V$  and ii) the rest of the tree. Let us denote the subset of the evidence, residing at the subtree of  $V$  by  $e_v^-$  and the subset of the evidence from the rest of the tree by  $e_v^+$ . We denote the belief function of the node  $V$  by  $\text{Bel}(V)$ , where it is defined as

$$\begin{aligned}\text{Bel}(V) &= P(V/e_v^+, e_v^-) \\ &= P(e_v^-/V) \cdot P(V/e_v^+) / \alpha \\ &= \lambda(V) \prod(V) / \alpha\end{aligned}\quad (1.3)$$

$$\left. \begin{array}{l} \text{where, } \lambda(V) = P(e_v^-/V), \\ \prod(V) = P(V/e_v^+), \end{array} \right\} \quad (1.4)$$

and  $\alpha$  is a normalizing constant, determined by

$$\alpha = \sum_{v \in \{\text{true, false}\}} P(e_v^-/V) \cdot P(V/e_v^+) \quad (1.5)$$

It seems from the last expression that  $v$  could assume only two values: true and false. It is just an illustrative notation. In fact,  $v$  can have a number of possible values.

Pearl designed an interesting algorithm for belief propagation in a causal tree. He assigned *a priori* probability of one leaf node, say node  $H$  in Fig. 1.11, to be defective and then propagated the belief from this node to its parent and then from the parent to the grandparent, until the root is reached. Then he considered a downward propagation of belief from the root to its children, and from each child node to its children, and so on until the leaves are reached. The leaf having the highest belief is then assigned *a priori* probability and the whole process described above is repeated. Pearl has shown that after a finite number of up-down traversal on the tree, a *steady-state* condition is reached following which a particular leaf node in all subsequent up-down traversal yields a maximum belief with respect to all other leaves in the tree. The leaf thus selected is considered as the defective item.

## 1.7 Computational Learning Theory

The main question on machine learning is how does one know that his learning algorithm has generated a concept, appropriate for predicting the future correctly? For instance, in inductive learning [50] how can we assert that our hypothesis  $h$  is sufficiently close to the target function  $f$ , when we do not know

f? These questions can be answered with the help of computational learning theory.

The principle of computational learning theory states that any hypothesis, which is sufficiently incorrect, will be detected with a high probability after experimenting with a small number of training instances. Consequently, a hypothesis that is supported by a large number of problem instances will be unlikely to be wrong and hence should be **Probably Approximately Correct (PAC)**.

The PAC learning was defined in the last paragraph w.r.t. training instances. But what about the validity of PAC learning on the test set (not the training set) of data. The assumption made here is that the training and the test data are selected randomly from the population space with the same probability distribution. This in PAC learning theory is referred to as **stationary assumption**.

In order to formalize the PAC learning theory, we need a few notations [68].

Let  $X$  = exhaustive set of examples,

$D$  = distribution by which the sample examples are drawn,

$m$  = cardinality of examples in the training set,

$H$  = the set of possible hypotheses, and

$f$  = function that is approximated by the hypothesis  $h$ .

We now define an error of hypothesis  $h$  by

$$\text{Error}(h) = P(h(x) \neq f(x) \mid x \in D). \quad (1.6)$$

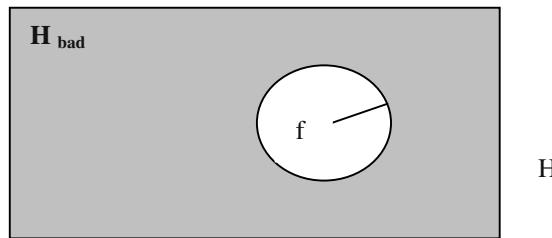
A hypothesis  $h$  is said to be **approximately correct** when

$$\text{error}(h) \leq \epsilon, \quad (1.7)$$

where  $\epsilon$  is a small positive quantity.

When an approximate hypothesis  $h$  is true, it must lie within the  $\epsilon$ -ball around  $f$ . When  $h$  lies outside the  $\epsilon$ -ball we call it a bad hypothesis [65].

Now, suppose a hypothesis  $h_b \in H_{\text{bad}}$  is supported by first  $m$  examples. The probability that a bad hypothesis is consistent with an example  $\leq (1-\epsilon)$ . If we consider  $m$  examples to be independent, the probability that all  $m$  samples will be consistent with hypothesis  $h_b$  is  $\leq (1-\epsilon)^m$ . Now, if  $H_{\text{bad}}$  has to contain a consistent hypothesis, at least one of the hypothesis of  $H_{\text{bad}}$  should be consistent. The probability of this happening is bounded by the sum of individual probabilities.



**Fig. 17:** The ball around  $f$ .

$$\text{Thus } P(\text{a consistent } h_b \in H_{\text{bad}}) \leq |H_{\text{bad}}| (1 - \epsilon)^m \quad (1.8)$$

$$\leq |H| (1 - \epsilon)^m \quad (1.9)$$

where  $|H_{\text{bad}}|$  and  $|H|$  denotes the cardinality of them respectively. If we put a small positive upper bound  $\delta$  to the above quantity: we find

$$|H| (1 - \epsilon)^m \leq \delta \quad (1.10)$$

$$\Rightarrow m \geq (1/\epsilon) [\ln(1/\delta) + \ln(H)] \quad (1.11)$$

Consequently, if a learning algorithm asserts an hypothesis that is supported by  $m$  number of examples, then it must be correct with a probability  $\geq(1-\delta)$ , when the error is  $\leq \epsilon$ . So, we can call it probably approximately correct.

## 1.8 Synergism in Soft Computing

The characteristics of the soft computing tools reveal that each tool has its own pros and cons. Fuzzy logic, for example, is good in approximate reasoning, but does not have much scope in machine learning or optimization applications. ANN on the other hand has a bigger role in machine learning. GA can be employed in intelligent search, optimization and some machine learning applications. Belief networks can be applied in diagnosis of systems from noisy sensory data. A co-operative synthesis of these tools thus may give rise to better computational models that can complement the limitation of one tool by the judicious use of the others.

Depending on the behavioral compatibility of the soft computing tools, we can classify their synergism into the following types. Each type again may be sub-divided into **strongly coupled** and **weakly coupled** synergism [37]. In a

---

strongly coupled synergism, the individual tools are mixed in an inseparable manner, whereas in a weakly coupled synergism the individual tools have their structural/ modular identity. More details on this issue will be discussed in a separate chapter.

### **1.8.1 Neuro-Fuzzy Synergism**

Synergism of this type is needed to use the joint benefits of neural nets and fuzzy logic in a common platform [77]. A blending of neural nets and fuzzy logic facilitates the users with a system capable of i) machine learning from approximate data/ knowledge and/or ii) approximate reasoning using the knowledge acquired through machine learning. Neuro-fuzzy synergism may also be incorporated into an on-line knowledge acquisition system for automatically acquiring knowledge and/or its parameters, [40] such as certainty factors, membership distributions or conditional probabilities, while reasoning.

### **1.8.2 Neuro-GA Synergism**

Neural nets and GA together can give rise to different forms of synergism. Most of the Neuro-GA synergism is centered on the adaptation of the weights of a neural net by using GA. As a specific example, GA can be employed to determine the weights of a Hopfield neural net [76]. The energy function of the Hopfield net under this circumstance may be regarded as the fitness function of the GA [40]. GA can also be used to determine the weights of a feed-forward neural net. A mean-square-error sum of the neurons at the output layer may, under this circumstance, be used as the fitness function. The training thus imparted to the neural net occasionally outperforms the typical back-propagation algorithm for the possibility of its getting trapped at *local minima* [63].

### **1.8.3 Fuzzy-GA Synergism**

Common forms of fuzzy-GA synergism include optimization of the parameters of a fuzzy system by using GA. A fuzzy system employs a fuzzifier to map the real world signal amplitudes into a range of [0, 1]. This is usually realized by a specialized curve called membership distributions. The x-axis of the membership distribution is the signal amplitude, while its y-axis denotes the fuzzy scale. Choice of the distributions has a significant impact in the response of a fuzzy system. The membership distributions in a fuzzy system may be adapted by a GA for optimization in the performance of a fuzzy system.

### **1.8.4 Neuro-Belief Network Synergism**

This is a relatively a new type of synergism and much discussion on this is not readily available in the current literature. A commonsense reasoning reveals 2

possible types of synergism under this category. Firstly, neural networks may be employed to determine the conditional probabilities in a belief network from the measurement data of known case histories. Secondly, a specialized type of belief computation, called the Dempster-Shfer theory, may by applied to fuse multi-sensory data, which consequently may be used as the training instances of a supervised neural net [74].

### 1.8.5 GA-Belief Network Synergism

Like Neuro-Belief network synergism, this too is new and unfortunately this has not been reported elsewhere. GA may be used to adapt the conditional probabilities of a belief network, and a performance criterion may be defined to test the success of the known case histories with the presumed conditional probabilities in each GA cycle until convergence of the GA occurs. In the coming future, GA-belief network synergism will be applied extensively in fault detection of complex engineering systems.

### 1.8.6 Neuro-Fuzzy-GA Synergism

This can be configured in different forms. One simple configuration of this includes a neural net as a pattern classifier, trained with fuzzy membership distributions, which has been pre-tuned by a GA. Among the other configurations, GA may be employed in a tightly coupled neuro-fuzzy system to optimize its parameters. GA can also be used to determine the best set of training instances of a tightly coupled neuro-fuzzy system.

## 1.9 Conclusions

The chapter introduced different computational models of machine intelligence and discussed their scope of possible synergism. It is clear from the discussions that fuzzy logic is a fundamental tool for reasoning with approximate data and knowledge. Neural network plays a significant role in machine learning and GA has an extensive application in intelligent search and optimization problems. Belief networks are capable of propagating beliefs of an event node based on the probabilistic support of its cause and effect nodes in the causal tree/graph. Computational learning theory provides an answer to the correctness of a learning algorithm. The chapter ended with a list of possible synergism of 2 or more computational models that fall under the rubric of soft computing.

## Exercises

1. Formulate the classical water-jug problem presented below by GA.

Given 2 jugs of content 4 liters and 3 liters. There is no marking in the jugs. You need to separate 2 liters of water by using these 2 jugs only.

[**Hints:** Define operators like transferring water from one jug to another, evacuating a jug, filling in a jug and the like.....]

2. Show that the derivative of a sigmoid function:

$$\text{OUT} = 1 / (1 + \exp(-\text{NET}))$$

is given by

$$\partial \text{OUT} / \partial \text{NET} = \text{OUT} (1 - \text{OUT})$$

and hence show that at the origin ( $\text{NET}$ ,  $\text{OUT}$ ) = (0, 0), the slope of this function is 1/4. Also show that when  $\text{NET}$  approaches  $+\infty$  or  $-\infty$ , the slope approaches 0.

3. Show graphically that the straight line

$$y = x + 3$$

classifies the point sets  $A = \{(1, 5), (2, 6), (3, 7), (4, 9), (5, 11)\}$  and

$$B = \{(1, 3), (2, 4), (3, 5), (4, 6), (5, 7)\}$$

into 2 classes.

4. Prove that the point sets A and B generated by

$$A = \{(x, y): y = x^2 + 3\} \text{ and}$$

$$B = \{(x, y): y = x^2 - 3\}$$

can be separated into 2 classes by the straight line:  $y = 2x$  for  $0 \leq x < 3$ .

[**Hints:** Since  $x^2 + 3 - 2x = (x - 1)^2 + 2 > 0$  and  $x^2 - 3 - 2x = (x - 1)^2 - 4 < 0$  for  $0 \leq x < 3$ , therefore,  $y = 2x$  is a classifier for the point sets in A and B for  $0 \leq x < 3$ .]

5. Suppose in designing an engineering system, we have a constraint Z that needs to be maximized in  $-2 \leq x \leq 2$  and  $-2 \leq y \leq 2$ .

$$Z = \frac{x^3 - y^3}{x - y}.$$

- a) Is the function partially differentiable in the prescribed range of parameter x and y?
- b) If no, can we use GA for optimizing Z? Explain in detail how GA can be applied for the above system optimization problem.

**[Hints:** a) Since  $Z(x, y)$  at  $(x, y) = (0, 0)$  is undefined, the function is not continuous, and thus not differentiable. b) GA can be applied to optimize the function. We may consider two fields  $x, y$  of the chromosomes and use  $Z$  as the fitness function. Take precaution that the search is restricted to  $-2 \leq x \leq \delta$  &  $\delta \leq x \leq 2$  and  $-2 \leq y \leq -\delta$  &  $\delta \leq y \leq 2$  for  $\delta \rightarrow 0^+ = 10^{-3}$ .]

## References

- [1] Abidi, M. A. and Gonzalez, R. C. (Eds.), *Data Fusion in Robotics and Machine Intelligence*, Academic Press, San Diego, CA, 1992.
- [2] Anderson, J. A., "A simple neural network generating an associative memory," *Mathematical Biosciences*, vol. 14, pp. 197-220, 1972.
- [3] Antoniou, G., *Nonmonotonic Reasoning*, MIT Press, 1997.
- [4] Back, T., Fogel, D. B. and Michalewicz, Z., *Handbook of Evolutionary Computation*, IOP and Oxford University Press, Bristol, UK, 1997.
- [5] Bender, E. A., *Mathematical Methods in Artificial Intelligence*, IEEE Computer Society Press, Los Alamitos, pp. 589-593, 1996.
- [6] Bezdek, J. C., *Fuzzy Mathematics in Pattern Classification*, Ph.D. thesis, Applied Mathematics Center, Cornell University, Ithaca, 1973.
- [7] Bezdek, J. C., "What is Computational Intelligence?" In *Computational Intelligence Imitating Life*, Zurada, J. M., Marks, R. J. and Robinson, C. J. (Eds.), *IEEE Press*, NY, pp. 1-12, 1994.
- [8] Biswas, B., Konar, A. and Mukherjee, A. K., "Image Matching with Fuzzy Moment Descriptors," *Engineering Applications of Artificial Intelligence*, vol. 14, pp. 43-49, 2001.

- 
- [9] Broida, T. J., Kinematic and Statistical Models for Data Fusion Using Kalman Filtering, In *Data Fusion in Robotics and Machine Intelligence*, Abidi, M. A. and Gonzalez, R. C. (Eds.), Academic Press, San Diego, CA, 1992.
  - [10] Carpenter, G. A. and Grossberg, S., “A massively parallel architecture for a self-organizing neural pattern recognition machine,” *Computer Vision, Graphics and Image Processing*, vol. 37, pp. 54-115, 1987.
  - [11] Carpenter, G. A. and Grossberg, S., “ART2: Self-organization of stable category recognition codes for analog input patterns,” *Applied Optics*, vol. 23, pp. 4919-4930, Dec. 1987.
  - [12] Castro, J. L., Carlos, M. J., and Benitez, J. M., “Interpretation of Artificial Neural Networks by Means of Fuzzy Rules,” *IEEE Trans. on Neural Networks*, vol. 13, no. 1, Jan. 2002.
  - [13] Chakraborty, U. K. and Dastidar, D. G., “Using reliability analysis to estimate the number of generations to convergence in genetic algorithm,” *Information Processing Letters*, vol. 46, pp. 199-209, 1993.
  - [14] Chakraborty, U. K. and Muehlenbein, H., “Linkage equilibrium and genetic algorithms,” *Proc. 4<sup>th</sup> IEEE Int. Conf. on Evolutionary Computation*, Indianapolis, pp. 25-29, 1997.
  - [15] Chakraborty, U. K., Deb, K. and Chakraborty, M., “Analysis of selection algorithms: A Markov chain approach,” *Evolutionary Computation*, vol. 4, no. 2, pp. 133-167, 1996.
  - [16] Davis, T. E. and Principa, J. C., “A Markov chain framework for the simple Genetic Algorithm,” *Evolutionary Computation*, vol. 1, no. 3, pp. 269-288, 1993.
  - [17] De Jong, K. A., *An analysis of behavior of a class of genetic adaptive systems*, Doctoral dissertation, University of Michigan, 1975.
  - [18] Elfes, A., Multi-source Spatial Fusion Using Bayesian Reasoning, In *Data Fusion in Robotics and Machine Intelligence*, Abidi, M. A. and Gonzalez, R. C. (Eds.), Academic Press, San Diego, CA, 1992.
  - [19] Filho, J. L. R. and Treleven, P. C., *Genetic Algorithm Programming Environment*, IEEE Computer Society Press, pp. 28-43, June 1994.
  - [20] Fogel, D. B., *Evolutionary Computation*, IEEE Press, Piscataway, NJ, 1995.

- [21] Fu, Li M, *Neural Networks in Computer Intelligence*, McGraw-Hill, NY, 1994.
- [22] Goldberg, D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.
- [23] Haykin, S., *Neural Networks: A Comprehensive Foundation*, Prentice-Hall, NJ, 1999.
- [24] Hertz, J., Krogn, A. and Palmer, G. R., *Introduction to the Theory of Neural Computation*, Addison-Wesley, Reading, MA, 1990.
- [25] Holland, J. H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.
- [26] Hopfield, J. J., “Neural networks with graded response have collective computational properties like those of two state neurons,” *Proc. of the National Academy of Sciences*, vol. 81, pp. 3088-3092, May 1984.
- [27] Hopfield, J. “Neural nets and physical systems with emergent collective computational abilities,” *Proc. of the National Academy of Sciences*, vol. 79, pp. 2554-2558, 1982.
- [28] Huntsberger, T. L., Data Fusion: A Neural Network Implementation, In *Data Fusion in Robotics and machine Intelligence*, Abidi, M. A. and Gonzalez, R. C. (Eds.), Academic Press, San Diego, CA, 1992.
- [29] Jain, L. C. and Lazzerini, B. (Eds.), *Knowledge-Based Intelligent Techniques in Character Recognition*, CRC Press, Boca Raton, 1999.
- [30] Jain, L. C. and Martin, and N. M. (Eds.), *Fusion of Neural Networks, Fuzzy Sets and Genetic Algorithms: Industry Applications*, CRC Press, 1998.
- [31] Jain, L. C. and Silva, C. W. De. (Eds.), *Intelligent Adaptive Control: Industrial Applications*, CRC press, Boca Raton, 1998.
- [32] Jain, L. C. (Ed.), *Intelligent Biometric Techniques in Fingerprint and Face Recognition*, CRC Press, Boca Raton, 1999.
- [33] Jamshidi, M., Titli, A., Zadeh, L. and Boverie, S. (Eds.), *Applications of Fuzzy Logic: Towards High Machine Intelligence Quotient Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1997.
- [34] Klir, G. J. and Yuan, B., *Fuzzy Sets and Fuzzy Logic: Theory and Applications*, Prentice-Hall, NJ, 1995.

- 
- [35] Kohonen, T., Barna, G. and Chrisley, R., "Statistical pattern recognition using neural networks: Benchmarking studies," *IEEE Conf. on Neural Networks*, San Diego, vol. 1, pp. 61-68.
  - [36] Kohonen, T., *Self-organization and Associative Memory*, Springer-Verlag, Berlin, 1989.
  - [37] Konar, A. and Jain, L. C., An introduction to computational intelligence paradigms, In *Practical Applications of Computational Intelligence Techniques*, Jain, L. C. and Wilde, P. D. (Eds.), Kluwer, 2001.
  - [38] Konar, A. and Mandal, A. K., "Uncertainty Management in Expert Systems using Fuzzy Petri Nets," *IEEE Trans. on Knowledge and Data Engineering*, vol. 8, no. 1, 1996.
  - [39] Konar, A. and Pal, S., Modeling cognition with fuzzy neural nets, In *Neural Network Systems: Techniques and Applications*, Leondes, C. T. (Ed.), Academic Press, New York, pp. 1341-1391, 1999.
  - [40] Konar, A., *Artificial Intelligence and Soft Computing: Behavioral and Cognitive modeling of the Human Brain*, CRC Press, Boca Raton, 1999.
  - [41] Kosko, B., *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*, Prentice-Hall, Englewood Cliffs, NJ, 1991.
  - [42] Kosko, B., "Adaptive bi-directional associative memories," *Applied Optics*, vol. 26, pp. 4947-4960, 1987.
  - [43] Kosko, B., "Bi-directional associative memories," *IEEE Trans. on Systems, Man and Cybernetics*, vol. SMC-18, pp. 49-60, Jan 1988.
  - [44] Koza, J. R., *Genetic Programming: On the programming of Computers by Means of Natural Selection*, MIT Press, 1992.
  - [45] Langton, C. G. (Ed.), *Artificial Life*, vol. 6, Addison-Wesley, Reading, MA, 1989.
  - [46] Luo, Fa- Long and Unbehauen, R., *Applied Neural Networks for Signal Processing*, Cambridge University Press, London, pp. 1-31, 1997.
  - [47] Marks, R. J., "Intelligence: Computational versus Artificial," *IEEE Trans. on Neural Networks*, vol. 4, pp. 737-739, 1993.

- [48] Mc Donell, J. R., "Control" in *Handbook of Evolutionary Computation*, Back, T., Fogel, D. B. and Michalewicz, Z. (Eds.), IOP and Oxford University Press, New York, 1998.
- [49] Michalewicz, Z, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Berlin, 1992.
- [50] Mitchell, M. M., *Machine Learning*, McGraw-Hill, New York, pp. 81-127, 1997.
- [51] Mitchell, M., *An introduction to Genetic Algorithms*, MIT Press, Cambridge, MA, 1996.
- [52] Muehlenbein, H. and Chakraborty, U. K., "Gene pool recombination genetic algorithm and the onemax function," *Journal of Computing and Information Technology*, vol. 5, no. 3, pp. 167-182, 1997.
- [53] Pal, S. K. and Mitra, S., *Neuro-Fuzzy Pattern Recognition: Methods in Soft Computing*, John Wiley & Sons, Inc., 1999.
- [54] Patterson, D. W., *Introduction to Artificial Intelligence and Expert Systems*, Prentice-Hall, Englewood Cliffs, NJ, pp. 107-119, 1990.
- [55] Paul, B., Konar, A. and Mandal, A. K., "Fuzzy ADALINEs for gray image recognition," *Neurocomputing*, vol. 24, pp. 207-223, 1999.
- [56] Pearl, J., "Distributed revision of composite beliefs," *Artificial Intelligence*, vol. 33, pp. 173-213, 1987.
- [57] Pearl, J., "Fusion, propagation and structuring in belief networks," *Artificial Intelligence*, vol.29, pp. 241-288, 1986.
- [58] Pedrycz, W., *Fuzzy Sets Engineering*, CRC Press, Boca Raton, FL, pp. 73-106, 1996.
- [59] Pedrycz, W. and Gomide, F., *An Introduction to Fuzzy Sets: Analysis and Design*, MIT Press, 1998.
- [60] Peng, Y. and Reggia, J. A., "A probabilistic causal model for diagnostic problem solving," *IEEE Trans. on Systems, Man and Cybernetics*, SMC-17, no. 3, pp. 395-408, May-June 1987.
- [61] Quinlan, J. R., "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81-106.
- [62] Ross, T. J., *Fuzzy Logic with Engineering Applications*, McGraw-Hill, 1995.

- 
- [63] Rumelhart, D. E. and McClelland, J. L., *Parallel Distributed Processing: Exploring in the microstructure of cognition*, MIT Press, Cambridge, MA, 1986.
  - [64] Rumelhart, D. E., Hinton, G. E. and Williams R. J., "Learning representations by back-propagation errors," *Nature*, vol. 323, pp. 533-536, 1986.
  - [65] Russel, S. and Norvig, P., *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Englewood Cliffs, NJ, pp. 598-644, 1995.
  - [66] Saha, P. and Konar, A., "A heuristic approach for computing the max-min inverse fuzzy relation," *Int. J. of Approximate Reasoning*, vol. 30, pp. 131-147, 2002.
  - [67] Schalkoff, R. J., *Artificial Neural Networks*, McGraw-Hill, New York, pp. 146-188, 1997.
  - [68] Shafer, G. and Logan, R., "Implementing Dempster's rule for hierarchical evidence," *Artificial Intelligence*, vol. 33, pp. 271-298, 1987.
  - [69] Shafer, G., *A Mathematical Theory of Evidence*, Princeton University Press, Princeton, NJ, 1976.
  - [70] Shenoy, P. P. and Shafer, G., "Propagating belief functions with local computations," *IEEE Expert*, pp. 43-52, Fall 1986.
  - [71] Shoham, Y., *Artificial Intelligence Techniques in PROLOG*, Morgan Kaufmann, San Mateo, CA, pp. 183-185, 1994.
  - [72] Shortliffe, E. H. and Buchanan, B. G., "A model of inexact reasoning," *Mathematical Biosciences*, vol. 23, pp. 351-379, 1975.
  - [73] Shortliffe, E. H., *Computer based medical consultations: MYCIN*, American Elsevier, New York, 1976.
  - [74] Sil, J. and Konar, A., "Reasoning Using a Probabilistic Predicate Transition Net Model," *Int. J. of Modeling and Simulations*, vol. 21, no. 2, pp. 155-168, 2001.
  - [75] Srinivas, M. and Patnaik, L. M., "Genetic search: Analysis using fitness moments," *IEEE Trans. on Knowledge and Data Engg.*, vol. 8, no. 1, pp. 120-133, 1996.

- [76] Tank, D. W. and Hopfield, J. J., "Simple neural optimization networks: An A/D converter, signal decision circuit and a linear programming circuit," *IEEE Trans. on Circuits and Systems*, vol. 33, pp. 533-541, 1986.
- [77] Teodorescu, H. N., Kandel, A. and Jain, L. C., Eds., *Fuzzy and Neuro-Fuzzy systems in Medicine*, CRC Press, London, 1999.
- [78] Vose, M. D. and Liepins, G. E., "Punctuated equilibrium in genetic search," *Complex Systems*, vol. 5, pp. 31-44, 1991.
- [79] Vose, M. D., *Genetic Algorithms*, MIT Press, 1999.
- [80] Wasserman, P. D., *Neural Computing: Theory and Practice*, Van Nostrand Reinhold, New York, pp. 49-85, 1989.
- [81] Widrow, B. and Hoff, M. E., Adaptive Switching Circuits, in *1960 IRE WESCON Convention Record*, Part 4, pp. 96-104, NY, 1960.
- [82] Widrow, B., "Generalization and information storage in networks of ADALINE neurons," in *Self-organizing systems*, Yovits, M. C., Jacobi, G. T. and Goldstein, G. D., Eds., pp. 435-461, 1962.
- [83] Williams, R. J., "On the use of back-propagation in associative reinforcement learning," in *IEEE Int. Conf. on Neural Networks*, NY, vol. 1, pp. 263-270, 1988.
- [84] Williams, R. J. and Peng, J., "Reinforcement learning algorithm as function optimization," Proc. of *Int. Joint Conf. on Neural Networks*, NY, vol. II, pp. 89-95, 1989.
- [85] Winston, P., *Learning Structural Descriptions from Examples*, Ph.D. Dissertation, MIT Technical Report AI-TR-231, 1970.
- [86] Zadeh, L. A., "Outline of a new approach to the analysis of complex systems and decision processes," *IEEE Trans. Systems, Man and Cybernetics*, vol. 3, pp. 28-45, 1973.
- [87] Zadeh, L. A., Fuzzy Sets, *Information and Control*, vol. 8, pp. 338-353, 1965.
- [88] Zadeh, L. A., Fuzzy logic, neural networks and soft computing, one page course announcement of CS 294-4, Spring 1993, University of California at Berkeley, November 1992.
- [89] Zimmerman, H. J., *Fuzzy Set Theory and Its Applications*, Kluwer Academic, Dordrecht, The Netherlands, pp. 131-162, 1996.

# 2

# Fuzzy Sets and Relations

*The chapter provides an introduction to fuzzy sets, fuzzy relations and some elementary fuzzy operators such as t-norm, s-norm, max-min composition and max-product composition operators. The extension principle of fuzzy sets and the concept of projection and cylindrical extension have been outlined in the chapter with examples. A brief introduction to fuzzy linguistic variables and fuzzy hedges is also given at the end of the chapter.*

## 2.1 Conventional Sets

Mathematicians define **sets** as a collection of objects having one or more common characteristics. The objects that belong to a set are called **members/elements**. The characteristics used to define a set should be sufficient to identify its members. For example, persons enrolled for a course may together be called a STUDENT set. We call it a set as we can easily determine whether a person is a student by checking his/her name in the registrar book. LARGE RIVERS of a country, however, should not be called a set unless we clearly define their minimum length to be considered as LARGE.

Let  $A$  be a set and  $x$  be a member of  $A$ , we can denote this by the following notation:

$$x \in A \quad (2.1)$$

Also let  $y$  be a member of set  $B$ , such that for all  $y$ ,  $y$  is also a member of  $A$ . Then  $B$  is called the subset of  $A$ , denoted by

$$B \subseteq A. \quad (2.2)$$

If  $B$  is a subset of  $A$  but every element of  $A$  are not present in  $B$  then  $B$  is called the **proper subset** of  $A$ , denoted by

$$B \subset A. \quad (2.3)$$

If every element contained in  $B$  is an element of  $A$  and every element contained in  $A$  is an element of  $B$ , then set  $A$  and set  $B$  are **equal**, given by

$$A = B. \quad (2.4)$$

For any 2 sets  $A$  and  $B$  if there exist at least one common element  $x$  of both  $A$  and  $B$ , then we say that

$$x \in (A \cap B), \quad (2.5)$$

where  $\cap$  denotes a logical **intersection operation**.

For any 2 sets  $A$  and  $B$  if there exists at least one element  $x$ , such that  $x$  is a member of  $A$  or  $B$ , then we say that

$$x \in (A \cup B), \quad (2.6)$$

where  $\cup$  denotes a **union operation**.

A **universal set  $U$**  in a particular domain is a set that includes all possible members of that domain. In other words, all sets in a given domain are subsets of the universal set  $U$ .

## 2.2 Fuzzy Sets

In a conventional set, the condition defining the set boundaries is very rigid. For example, consider a universal set AGE, OLD, VERY OLD, YOUNG, CHILD and BABY are subsets of the universal set AGE. The conventional approach to define these sets are illustrated below:

$$\text{BABY} = \{\text{age} \in \text{AGE}: 0 \text{ year} \leq \text{age} < 1 \text{ year}\},$$

$$\text{CHILD} = \{\text{age} \in \text{AGE}: 1 \text{ year} \leq \text{age} \leq 10 \text{ years}\},$$

$$\text{YOUNG} = \{\text{age} \in \text{AGE}: 19 \text{ years} \leq \text{age} \leq 40 \text{ years}\},$$

$$\text{OLD} = \{\text{age} \in \text{AGE}: 60 \text{ years} \leq \text{age} < 80 \text{ years}\},$$

and  $\text{VERY OLD} = \{\text{age} \in \text{AGE}: 80 \text{ years} \leq \text{age} < 120 \text{ years}\}$ .

In the above definitions age is a variable that may presume any value in the range [0, 120] years. It is clear from the definitions that the boundary of each set is distinct. Thus an age=11 months 29 days is a member of the set BABY, but once it is 1 year it falls in the set CHILD. Thus there is a sharp demarcation in the boundary definition of the sets BABY and CHILD at age=1 year. Measurements in a real world system being highly imprecise, such a sharp demarcation of 2 set boundaries may cause a wrong allocation of the members to a given set.

Another characteristic of a conventional set includes assignment of a grade of membership 1 to all its members and 0 to all its non-members. The following connotation is used to describe that the membership of an element  $x$  in a set  $A$  is 1, and the membership of a non-element  $y$  in the set  $A$  is 0.

$$\mu_A(x) = 1 \quad (2.7)$$

$$\mu_A(y) = 0 \quad (2.8)$$

A **fuzzy set** extends the binary membership: {0,1} of a conventional set to a spectrum in the interval of [0, 1]. Further, unlike a conventional set, all elements of the universal set  $U$  are members of a given set  $A$ . Thus for each element  $x \in U$ ,

$$0 \leq \mu_A(x) \leq 1. \quad (2.9)$$

It needs mention here that as all elements of a universal set  $U$  are members of a given fuzzy set  $A$ , therefore, 2 fuzzy sets  $A$  and  $B$  may have an overlap in the boundary definitions. For example, in contrast to the respective conventional sets: BABY, CHILD, YOUNG, OLD and VERY OLD, the corresponding fuzzy sets allow any age in the interval [0, 120] years as a member of each of the above sets but with different memberships in [0, 1]. As a specific instance, the age 22 is a member of all the fuzzy sets but the membership of age (=22) to belong to the sets BABY, CHILD, YOUNG, OLD and VERY OLD respectively are 0.001, 0.01, 1.00, 0.60 and 0.20. The above example makes sense in the line of reasoning that an age of 22 corresponds to a young person, so the membership of age (=22) to be young is high (1.00). The relative grading of the other memberships thus can be easily understood from the usual meaning of the terms BABY, CHILD, OLD and VERY OLD.

A fuzzy set thus can be formally defined as follows.

**Definition 2.1:** A *fuzzy set A* is a set of ordered pairs, given by

$$A = \{(x, \mu_A(x)) : x \in X\} \quad (2.10)$$

where  $X$  is a universal set of objects (also called the universe of discourse) and  $\mu_A(x)$  is the grade of membership of the object  $x$  in  $A$ . Usually,  $\mu_A(x)$  lies in the closed interval of  $[0, 1]$ .

It may be added here that some authors [7] relax the range of membership from  $[0, 1]$  to  $[0, R_{\max}]$  where  $R_{\max}$  is a positive finite real number. One can easily convert  $[0, R_{\max}]$  to  $[0, 1]$  by dividing the membership values in the range  $[0, R_{\max}]$  by  $R_{\max}$ .

There are other notations of fuzzy sets as well. For instance, the ordered pair  $(x, \mu_A(x))$  in the definition of fuzzy set is also written as  $x/\mu_A(x)$  or  $\mu_A(x)/x$  as well. Let the elements of set  $X$  be  $x_1, x_2, \dots, x_n$ . Then the fuzzy set  $A \subseteq X$  is denoted by any of the following nomenclature.

$$A = \{(x_1, \mu_A(x_1)), (x_2, \mu_A(x_2)), \dots, (x_n, \mu_A(x_n))\}, \text{ or}$$

$$A = \{x_1/\mu_A(x_1), x_2/\mu_A(x_2), \dots, x_n/\mu_A(x_n)\}, \text{ or}$$

$$A = \{x_1/\mu_A(x_1) + x_2/\mu_A(x_2) + \dots + x_n/\mu_A(x_n)\}, \text{ or}$$

$$A = \{\mu_A(x_1)/x_1 + \mu_A(x_2)/x_2 + \dots + \mu_A(x_n)/x_n\}, \text{ or}$$

$$A = \{\mu_A(x_1)/x_1, \mu_A(x_2)/x_2, \dots, \mu_A(x_n)/x_n\}.$$

In this book we used the last option. The details of membership function  $\mu_A(x)$  is formalized below.

## 2.3 Membership Functions

The grade of membership  $\mu_A(x)$  maps the object or its attribute  $x$  to positive real numbers in the interval  $[0, 1]$ . Because of its mapping characteristics like a function, it is called **membership function**. A formal definition of the membership function is given below for the convenience of the readers.

**Definition 2.2:** A *membership function  $\mu_A(x)$*  is characterized by the following mapping:

$$\mu_A: x \rightarrow [0, 1], x \in X \quad (2.11)$$

where  $x$  is a real number describing an object or its attribute and  $X$  is the universe of discourse and  $A$  is a subset of  $X$ .

A question that naturally arises is: how to construct a membership function? The following examples provide a thorough insight to the selection of the membership functions.

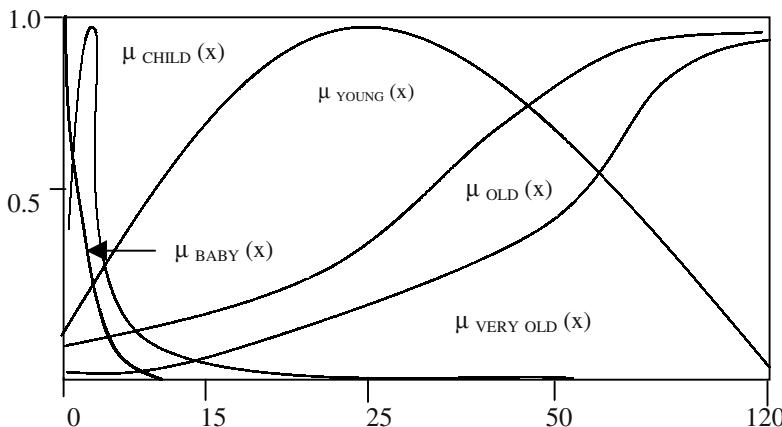
**Example 2.1:** Consider the problem of defining BABY, CHILD, YOUNG, OLD and VERY OLD by membership functions. The closer the age of a person to 0, the higher is his/her membership to be a BABY. So, if  $x$  is the age of the person, we can define BABY as follows:

$$\text{BABY} = \{(x, \mu_{\text{BABY}}(x)) \text{ where } \mu_{\text{BABY}}(x) = \exp(-x)\}. \quad (2.12)$$

Thus as  $x \rightarrow 0$ ,  $\mu_{\text{BABY}}(x) \rightarrow 1$ . Further, as  $x$  increases,  $\mu_{\text{BABY}}(x)$  decreases exponentially. The membership function  $\mu_{\text{BABY}}(x)$  can also be designed to have a controlled decrease with increasing  $x$  by including a factor  $\alpha$  to  $x$  in  $\exp(-x)$ . Thus,

$$\mu_{\text{BABY}}(x) = \exp(-\alpha x) \text{ for } \alpha > 0. \quad (2.13)$$

Larger the value of  $\alpha$ , the higher is the falling rate of  $\mu_{\text{BABY}}(x)$  over  $x$ . In a similar manner we can define the membership functions for CHILD, YOUNG, OLD and VERY OLD fuzzy sets. But before representing them mathematically let us take a look at them.



**Fig. 2.1:** Membership curves for the fuzzy sets: BABY, CHILD, YOUNG, OLD and VERY OLD. The x-axis denotes the age in years and the y-axis denotes the memberships of the given fuzzy sets at different ages.

The membership curves for the fuzzy sets: BABY, CHILD, YOUNG, OLD and VERY OLD are shown in Fig. 2.1. The curve for CHILD fuzzy set has the peak at some age slightly greater than 0 and has a sharp fall off around the peak. The

logical interpretation of this directly follows from the meaning of the word child. The membership curve for the fuzzy set YOUNG has a peak at age  $x=25$  and falls off very slowly on both sides around the peak. As youth is the most charming period of the human beings, we prefer to call people YOUNG even if they are away from 25 on either side. If the readers' view is different they can allow a sharp falloff of the curve around the age  $x=25$ . One interesting point to note about the OLD and VERY OLD membership curves is that OLD curve throughout has a higher membership than the VERY OLD curve until both saturate at age  $x = 120$  years. This is meaningful because if someone is called VERY OLD then he must be OLD, but the converse may not be true.

There are many ways to represent the membership functions shown in Fig. 2.1 by mathematical functions. One such representation is given below:

$$\mu_{\text{CHILD}}(x) = ax^2/(1+bx^2 + cx), \quad a, b, c > 0. \quad (2.14)$$

$$\mu_{\text{YOUNG}}(x) = \exp [ - (x - 25)^2 / 2 \sigma^2 ], \quad \sigma > 0 \quad (2.15)$$

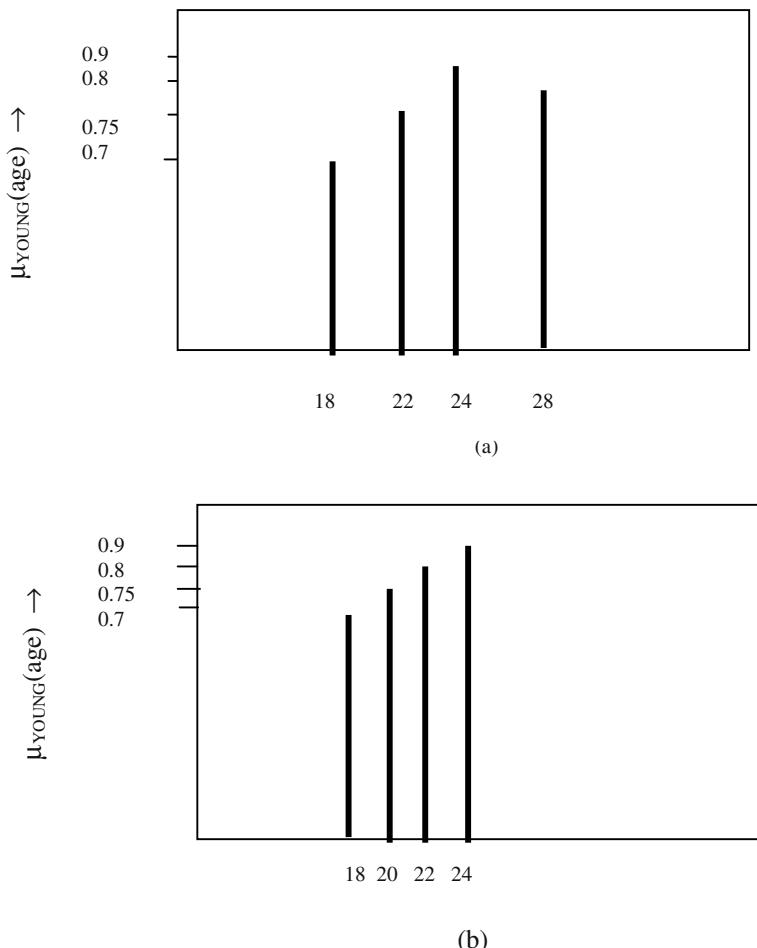
$$\mu_{\text{OLD}}(x) = 1 - \exp (-dx^2), \quad d > 0, \text{ and} \quad (2.16)$$

$$\mu_{\text{VERY OLD}}(x) = 1 - \exp(-dx), \quad d > 0. \quad (2.17)$$

The parameters  $a$ ,  $b$ ,  $c$  and  $d$  in the above membership functions are selected intuitively by the experts based on their subjective judgement in the respective domains. Tuning of these parameters is needed to control the curvature and sharp changes on the curves around some selected  $x$ -values.

## 2.4 Continuous and Discrete Membership Functions

The **universe of discourse** (or simply the **universe**) of a fuzzy set may exist for both discrete and continuous spectrum. For example, the roll number of students in a class is a discrete universe. On the other hand the height of persons is a continuous universe as height may take up any values between 4' to 8'. It may be mentioned here that a continuous universe is sometimes sampled at regular or irregular intervals for using it as a discrete universe. The membership curve of YOUNG in Fig. 2.1 may be, for instance, discretized at age  $x= 18, 22, 24, 28$ . This is an example of **non-uniform/ irregular sampling** as the intervals of sampling 18-22, 22-24, 24-28 are unequal. The membership curve of YOUNG may alternatively be sampled at a regular interval of age  $x=18, 20, 22, 24$ , say. This is an example of **uniform/ regular** sampling. Fig. 2.2(a) and (b) describe the instances of the non-uniform and uniform sampling of the YOUNG membership curve.



**Fig. 2.2:** (a) Non-uniform and (b) uniform sampling of the YOUNG membership curve.

## 2.5 Typical Membership Functions

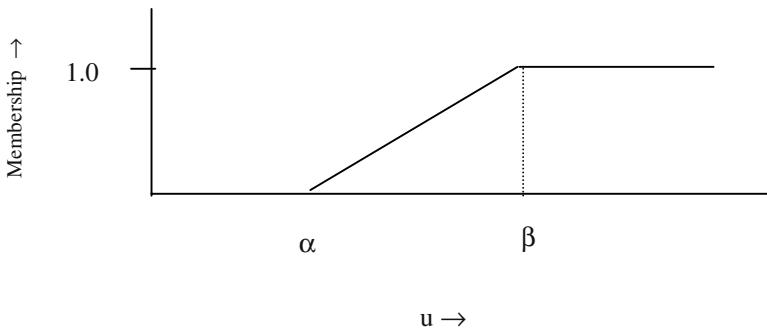
In theory, membership functions usually can take any form. But in most practical applications, triangular, Gaussian (bell-shaped), S-function and  $\gamma$ -functions are commonly used. In this section, these 4 functions are outlined.

### 2.5.1 The $\gamma$ -Function

This function has 2 parameters  $\alpha$  and  $\beta$ . It is formally defined by

$$\left. \begin{array}{ll} \gamma(u; \alpha, \beta) = 0, & u \leq \alpha, \\ = (u - \alpha)/(\beta - \alpha), & \alpha < u \leq \beta, \\ = 1 & u > \beta \end{array} \right\} \quad (2.18)$$

Fig. 2.3 describes the graphical representation of the  $\gamma$ -function.



**Fig. 2.3:** The membership curve for the  $\gamma$ -function.

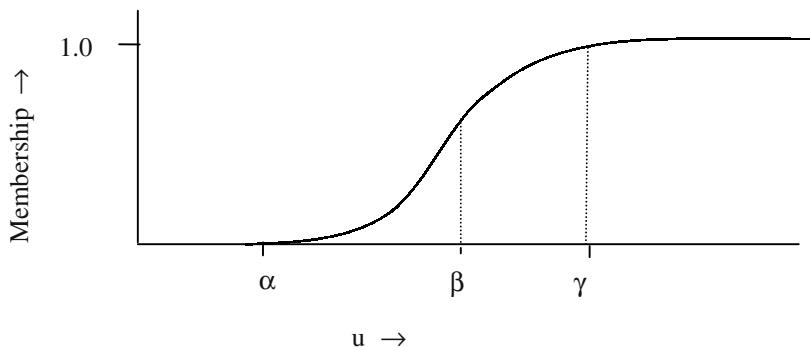
The membership function OLD in our previous example can be, for instance, described by the  $\gamma$ -function. Suppose we call someone OLD with some positive membership if his age exceeds 60 and call someone OLD with membership =1.0 when his age attains 90. So, the  $\gamma$ -function in the present context should be  $\gamma(\text{age}; 60, 90)$ .

## 2.5.2 The s-Function

This function is a smooth version of the  $\gamma$ -function mentioned above. It is formally defined as follows.

$$\left. \begin{array}{ll} S(u; \alpha, \beta, \gamma) = 0, & u \leq \alpha \\ = 2[(u - \alpha)/(\gamma - \alpha)]^2, & \alpha < u \leq \beta \\ = 1 - 2[(u - \gamma)/(\gamma - \alpha)]^2, & \beta < u \leq \gamma \\ = 1, & u > \gamma \end{array} \right\} \quad (2.19)$$

Generally,  $\beta = (\alpha + \gamma)/2$  is considered in most applications of fuzzy logic. One typical form of the S-function is presented in Fig. 2.4 below.



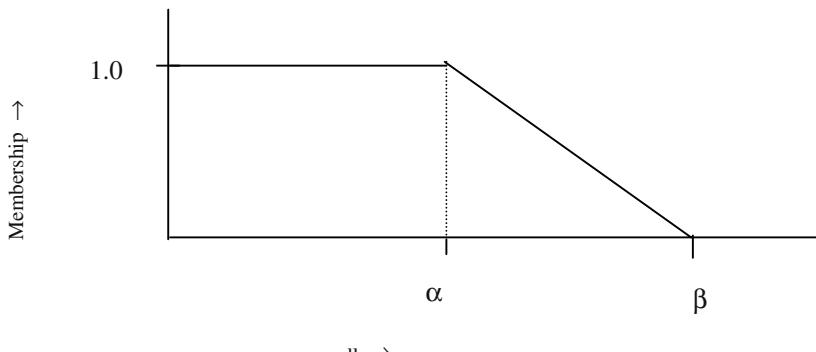
**Fig. 2.4:** One typical S-function.

The S-function also may be used to represent the OLD membership function. As the slope of the function at  $u = \alpha$  is very small, we can select a smaller age  $u$  to represent the OLD membership function.  $S(\text{age}; 40, 60, 90)$  thus is one choice for the membership function OLD.

### 2.5.3 The L-Function

This function is the converse of the typical  $\gamma$ -function. It can be mathematically expressed as

$$\left. \begin{aligned} L(u; \alpha, \beta) &= 1, & u < \alpha \\ &= (\alpha - u) / (\beta - \alpha), & \alpha \leq u \leq \beta \\ &= 0, & u > \beta \end{aligned} \right\} \quad (2.20)$$



**Fig. 2.5:** One typical form of the L-function.

One typical form of the L-function is presented in Fig. 2.5. L-functions are generally used to represent the fuzzy linguistic *positive small*. Suppose  $u$  is a fuzzy variable which should essentially have a positive value. Now, as  $u$  increases its membership should decrease. As a second example, suppose we are interested to describe the average intensity of the pixels (points) in an image by a fuzzy linguistic: *not very dark*. So, until the average intensity exceeds  $\alpha$  ( $=50$ , say), its membership of being not very dark is 1 and falls off if the average intensity exceeds  $\alpha$ .

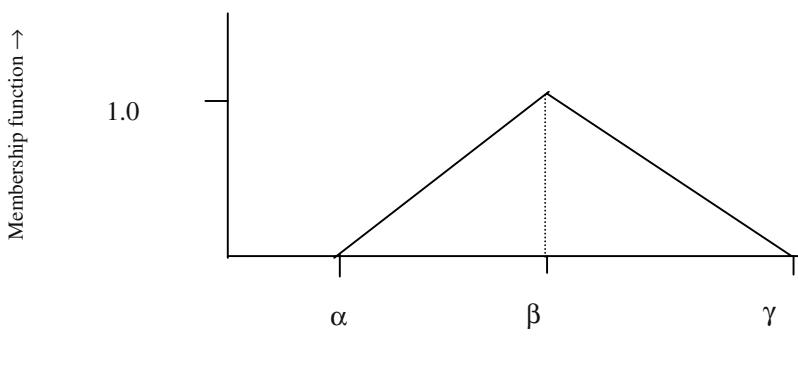
#### 2.5.4 The Triangular Membership Function

The triangular membership function, also called the bell-shaped function with straight lines, can be formally defined as follows:

$$\begin{aligned} \Lambda(u; \alpha, \beta, \gamma) = 0, & \quad u \leq \alpha \\ = (u - \alpha) / (\beta - \alpha), & \quad \alpha < u \leq \beta \\ = (\alpha - u) / (\beta - \alpha), & \quad \beta < u \leq \gamma \\ = 0, & \quad u > \gamma \end{aligned} \quad \left. \right\} \quad (2.21)$$

One typical plot of the triangular membership function is given in Fig. 2.6.

The YOUNG membership function, for instance, can be represented by the triangular membership function. We can set age  $\alpha = 20$ ,  $\beta = 25$  and  $\gamma = 30$  as the typical parameters for the YOUNG membership function in order to represent it by a triangular membership function.



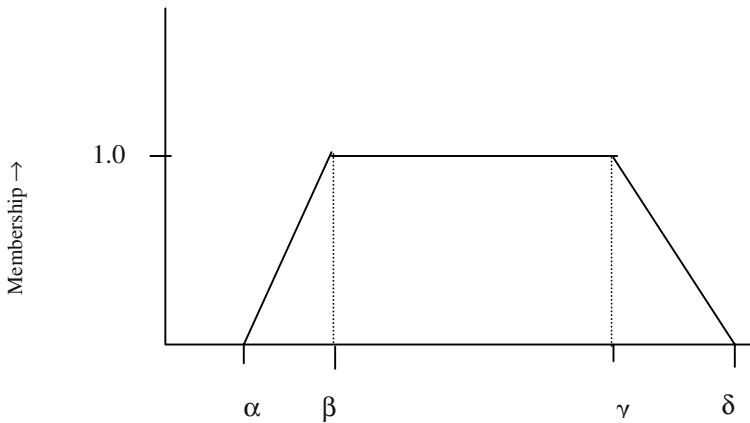
**Fig. 2.6:** One typical form of the triangular membership function.

### 2.5.5 The $\Pi$ -function

The  $\Pi$ -function can be formally described as follows:

$$\begin{aligned} \Pi(u; \alpha, \beta, \gamma, \delta) &= 0, & u \leq \alpha \\ &= (u - \alpha)/(\beta - \alpha), & \alpha < u \leq \beta \\ &= 1, & \beta < u \leq \gamma \\ &= (\gamma - u)/(\delta - \gamma), & \gamma < u \leq \delta \\ &= 0, & u > \delta. \end{aligned} \quad \left. \right\} \quad (2.22)$$

One typical plot of the  $\Pi$ -function is given in Fig. 2.7. The  $\Pi$ -function is used to represent the fuzzy linguistic: *neither so high nor so low*. For example suppose we want to express that today is *neither so hot nor so cold*. This can be represented by a fuzzy membership curve plotted against temperature. It may be noted that for temperature below a threshold  $th_1$  and above a threshold  $th_2$ , the membership of the said curve should be close to one and it should have falloffs below  $th_1$  and above  $th_2$ . Thus a  $\Pi$ -function is an ideal choice for the representation of the fuzzy linguistic *neither so hot nor so cold*.



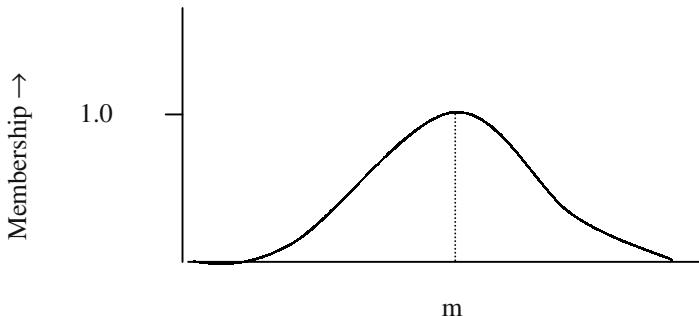
**Fig. 2.7:** One typical form of the  $\Pi$ -function.

### 2.5.6 The Gaussian Membership Function

A Gaussian membership function is defined by

$$G(u; m, \sigma) = \exp [-\{(u - m)/\sqrt{2\sigma}\}^2] \quad (2.23)$$

where the parameters  $m$  and  $\sigma$  control the center and width of the membership function. A plot of the Gaussian membership function is presented in Fig. 2.8.



**Fig. 2.8:** One typical form of the Gaussian function.

The Gaussian membership function has a wide application in the literature on fuzzy sets and systems. The YOUNG membership function illustrated earlier, for instance, can also be described by a Gaussian membership function with mean  $m=22$  years, say. Smaller the value of variance of the curve, higher is its sharpness around the mean.

## 2.6 Operation on Fuzzy Sets

Unlike conventional sets, the operations on fuzzy sets are usually described with reference to membership functions. Among the common operations on fuzzy sets fuzzy T-norm, fuzzy S-norm and fuzzy complementation need special mention. They are outlined below.

### 2.6.1 Fuzzy T-Norm

For any 2 fuzzy sets  $A$  and  $B$  under a common universe of discourse  $X$ , the intersection of the fuzzy sets, characterized by a T-norm operator, is given by

$$\mu_{A \cap B}(x) = T(\mu_A(x), \mu_B(x)). \quad (2.24)$$

For any membership values  $a, b, c$  and  $d$ , the T-norm operator  $T$  can be formally defined as follows.

$$T(0, 0) = 0, T(a, 1) = T(1, a) = a \quad (\text{boundary})$$

$$T(a, b) \leq T(c, d) \text{ if } a \leq c \text{ and } b \leq d \quad (\text{monotonicity})$$

$$T(a, b) = T(b, a) \quad (\text{commutativity})$$

$$T(a, T(b, c)) = T(T(a, b), c) \quad (\text{associativity})$$

**Definition 2.3:** A function  $T: [0, 1] \times [0, 1] \rightarrow [0, 1]$  that satisfies the above 4 characteristics is called a  $T$ -norm.

**Example 2.2:** The following are the examples of the typical  $T$ -norm function.

$$\text{a) Minimum: } T_{\min}(a, b) = \min(a, b) \quad (2.25)$$

$$\text{b) Algebraic product: } T_{ap}(a, b) = a.b \quad (2.26)$$

$$\text{c) Einstein product: } T_{ep}(a, b) = ab / \{2 - (a + b - ab)\} \quad (2.27)$$

$$\text{d) Drastic product: } T_{dp}(a, b) = \begin{cases} a & \text{if } b=1 \\ b & \text{if } a=1 \\ 0 & \text{otherwise.} \end{cases} \quad (2.28)$$

## 2.6.2 Fuzzy S-Norm

For any 2 fuzzy sets  $A$  and  $B$  under a common universe  $X$ , the union of fuzzy sets, characterized by a  $S$ -norm ( $T$ -co-norm) operator is given by

$$\mu_{A \cup B}(x) = S(\mu_A(x), \mu_B(x)). \quad (2.29)$$

For any membership values  $a, b, c$  and  $d$ , the  $S$ -norm operator  $S$  can be formally defined as follows.

$$S(1, 1) = 1, S(a, 0) = S(0, a) = a \quad (\text{boundary})$$

$$S(a, b) \leq S(c, d) \text{ if } a \leq c \text{ and } b \leq d \quad (\text{monotonicity})$$

$$S(a, b) = S(b, a) \quad (\text{commutativity})$$

$$S(a, S(b, c)) = S(S(a, b), c) \quad (\text{associativity})$$

**Definition 2.3:** A function  $S: [0, 1] \times [0, 1] \rightarrow [0, 1]$  that satisfies the above 4 characteristics is called a  $S$ -norm.

**Example 2.3:** The following are the examples of the typical  $S$ -norm function.

$$\text{a) Maximum: } S_{\max}(a, b) = \max(a, b) \quad (2.30)$$

$$\text{b) Algebraic sum: } S_{as}(a, b) = a + b - ab \quad (2.31)$$

c) Einstein sum:  $S_{es}(a, b) = (a + b) / (1 + ab)$  (2.32)

d) Drastic sum:  $S_{ds}(a, b) = \begin{cases} a & \text{if } b=0 \\ b & \text{if } a=0 \\ 1 & \text{otherwise.} \end{cases}$  (2.33)

### 2.6.3 Fuzzy Complement

Given a fuzzy set A under the universal set X, fuzzy complementation over set A is a mapping that transforms the membership function of A into the membership function of the complement of A, denoted by  $A^c$ . Mathematically, the complementation function c is given by

$$c[\mu_A(x)] = \mu_{A^c}(x) \quad (2.34)$$

The fuzzy complementation function should essentially satisfy the following 2 criteria:

1.  $c(0) = 1$  and  $c(1) = 0$  (boundary condition)

2. For any 2 fuzzy memberships a and b,

if  $a < b$  then  $c(a) \geq c(b)$  (non-increasing condition)

**Definition 2.4:** Any function  $c: [0, 1] \rightarrow [0, 1]$  that satisfies the above 2 characteristics is called fuzzy complementation.

**Example 2.4:** The following functions are typical examples of fuzzy complementation.

a)  $c[\mu_A(x)] = 1 - \mu_A(x)$  (2.35)

b)  $c_\lambda(a) = (1 - a) / (1 + \lambda a)$  (Sugeno class of complements) (2.36)

where for each value of the parameter  $\lambda$  in  $(-1, \infty)$  we obtain a particular fuzzy complement.

c)  $c_w(a) = (1 - a^w)^{1/w}$  (Yager class of complements) (2.37)

where for each value of w in  $(0, \infty)$  we obtain a particular fuzzy complement.

## 2.7 Basic Concepts Associated with Fuzzy Sets

This section introduces some elementary concepts associated with fuzzy sets. They are outlined below with examples.

**Definition 2.5:** The **support** of a fuzzy set A is the set of all points x in X such that  $\mu_A(x) > 0$ . Formally,

$$\text{Support}(A) = \{x \mid \mu_A(x) > 0\} \quad (2.38)$$

**Definition 2.6:** The **core** of a fuzzy set A is the set of points x in X such that  $\mu_A(x) = 1$ . Formally,

$$\text{Core}(A) = \{x \mid \mu_A(x) = 1\} \quad (2.39)$$

**Definition 2.7:** A fuzzy set with non-empty core is called **normal**. In other words, A is normal if  $\exists x, \mu_A(x) = 1$ .

**Definition 2.8:** A **crossover point** denotes a point x in X where  $\mu_A(x) = 0.5$ . Mathematically,

$$\text{Crossover}(A) = \{x \mid \mu_A(x) = 0.5\}. \quad (2.40)$$

**Definition 2.9:** The  **$\alpha$ -cut**, also called  **$\alpha$ -level**, of a fuzzy set A is a crisp set denoted by  $A_\alpha$  is given by

$$A_\alpha = \{x \mid \mu_A(x) \geq \alpha\}. \quad (2.41)$$

**Definition 2.10:** The **strong  $\alpha$ -cut**, also called **the strong  $\alpha$ -level**, of a fuzzy set A is a crisp set denoted by  $A_\alpha^+$  is given by

$$A_\alpha^+ = \{x \mid \mu_A(x) > \alpha\}. \quad (2.42)$$

**Example 2.5:** For the fuzzy set A = {0.1/x<sub>1</sub>, 0.5/x<sub>2</sub>, 0.7/x<sub>3</sub>, 0.9/x<sub>4</sub>, 1.0/x<sub>5</sub>, 0.5/x<sub>6</sub>}

Support (A)= {x<sub>1</sub>, x<sub>2</sub>, x<sub>3</sub>, x<sub>4</sub>, x<sub>5</sub>, x<sub>6</sub>} since for all the elements x<sub>1</sub>, x<sub>2</sub>, x<sub>3</sub>, x<sub>4</sub>, x<sub>5</sub>, x<sub>6</sub> of set A the membership values are greater than 0.

Core (A) = {x<sub>5</sub>} since  $\mu_A(x_5) = 1$ .

Crossover point (A) = {x<sub>2</sub>, x<sub>6</sub>} since both at x= x<sub>2</sub> and x<sub>6</sub> the membership value is 0.5.

$A_\alpha \mid_{\alpha=0.7} = \{x_1, x_3, x_4, x_5\}$  because for all these elements the membership values are greater than or equal to 0.7.

$A_\alpha^+ \mid_{\alpha=0.7} = \{x_1, x_4, x_5\}$  because for all these elements the membership values are greater than 0.7.

**Definition 2.11:** A fuzzy set  $A$  is convex if and only for any 2 points  $x_1$  and  $x_2$  and a real scalar  $\lambda$  in  $[0, 1]$ ,

$$\mu_A(\lambda x_1 + (1 - \lambda) x_2) \geq \min [\mu_A(x_1), \mu_A(x_2)] \quad (2.43)$$

**Definition 2.12:** A **fuzzy number**  $A$  is a fuzzy set on the real line that satisfies the conditions for convexity and normality.

**Definition 2.13:** A fuzzy set  $A$  is **symmetric** around a point  $x = c$  if

$$\mu_A(c + x) = \mu_A(c - x) \text{ for all } x \text{ belonging to } X.$$

A Gaussian type membership function, for instance, is symmetric around the mean value of the function.

## 2.8 Extension Principle of Fuzzy Sets

Let  $f(.)$  be a mapping function from fuzzy universal set  $X$  to fuzzy universal set  $Y$ , and  $A$  and  $B$  are subsets of  $X$  and  $Y$  respectively. Let the fuzzy set  $A$  be given by

$$A = \{\mu_A(x_1)/x_1, \mu_A(x_2)/x_2, \dots, \mu_A(x_n)/x_n\}. \quad (2.44)$$

If there is a one to one mapping from  $x_i$  to  $y_i = f(x_i)$  then  $B$  is given by

$$\begin{aligned} B = f(A) &= \{\mu_A(x_1)/f(x_1), \mu_A(x_2)/f(x_2), \dots, \mu_A(x_n)/f(x_n)\} \\ &= \{\mu_A(x_1)/y_1, \mu_A(x_2)/y_2, \dots, \mu_A(x_n)/y_n\}. \end{aligned} \quad (2.45)$$

But if many to one mapping exists from set  $X$  to  $Y$  then a maximum of the memberships of  $f(x_i), f(x_j), \dots, f(x_k)$ , where  $f(x_i) = f(x_j) = \dots = f(x_k)$  should be taken. Formally, for many to one mapping from set  $X$  to  $Y$ ,

$$\mu_B(y) = \max [\mu_A(x) : x \in f^{-1}(y)] \quad (2.46)$$

**Example 2.6:** The computation of  $B = f(A)$  is illustrated in this example. Let  $A = \{0.2/(-1), 0.4/(-2), 0.6/(1), 0.8/(2), 0.9/(3)\}$ , and  $f(x) = x^2$ . Here, since  $f(-1) = f(1) = 1$ , and  $f(-2) = f(2) = 4$ ,

$$\begin{aligned} \mu_B(f(-1)) &= \mu_B(f(1)) = \max [\mu_A(x) |_{x=1}, \mu_A(x) |_{x=-1}] \\ &= \max[0.6, 0.2] \end{aligned}$$

$$=0.6.$$

Similarly,  $\mu_B(f(-2)) = \mu_B(f(2)) = \max [0.4, 0.8]$

$$=0.8.$$

Consequently,  $B = f(A) = \{0.6/(1^2), 0.8/(2^2), 0.9/(3^2)\}$

$$=\{0.6/1, 0.8/4, 0.9/9\}.$$

So far we discussed mapping from a one dimensional space X to another one dimensional space Y. In general the mapping can be defined from an n dimensional product space  $X_1 \times X_2 \times X_3 \times \dots \times X_n$  to a single universe Y. Here  $X_1, X_2, \dots, X_n$  denote fuzzy universes. The mapping function in the present context is denoted by  $f(x_1, x_2, \dots, x_n)$ , where  $x_1 \in X_1, x_2 \in X_2, \dots, x_n \in X_n$ . If  $A_1, A_2, \dots, A_n$  are n fuzzy sets in  $X_1, X_2, \dots, X_n$  respectively, then extension principle asserts that the fuzzy set B induced by the mapping from  $A_1, A_2, \dots, A_n$  is given by

$B = \{\mu_B(y) / y, \text{ where } y = f(x_1, x_2, \dots, x_n)\}$  with

$$\begin{aligned} \mu_B(y) &= \max_{(x_1, x_2, \dots, x_n) \in f^{-1}(y)} [\min \{ \mu_{A_1}(x_1), \mu_{A_2}(x_2), \dots, \mu_{A_n}(x_n) \}], \text{ if } f^{-1}(y) \neq \text{null set}, \\ &= 0, \text{ otherwise.} \end{aligned} \quad (2.47)$$

**Example 2.7:** In this example we illustrate the extension principle for a function f of 2 variables  $x_1$  and  $x_2$ . Let  $f(x_1, x_2) = x_1 + x_2$ , and

$$A_1 = \{0.2/-1, 0.4/0, 0.6/1\} \text{ and } A_2 = \{0.8/-1, 0.6/0, 0.7/1\}.$$

Here,  $X_1 = \{-1, 0, 1\}$  and  $X_2 = \{-1, 0, 1\}$  as well.

Thus  $X_1 \times X_2 = \{(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 0), (0, 1), (1, -1), (1, 0), (1, 1)\}$

Consequently,  $f(-1, 0) = f(0, -1) = 0 + (-1) = -1$ ,

$$f(1, -1) = f(-1, 1) = -1 + 1 = 0,$$

$$f(1, 0) = f(0, 1) = 0 + 1 = 1,$$

$$f(-1, -1) = -2, f(0, 0) = 0, f(1, 1) = 2.$$

So,  $\mu_B(y)$  at  $y = f(-1, 0)$

$$\begin{aligned}
 &= \mu_B(y) \text{ at } y = f(0, -1) \\
 &= \max\{\min(0.2, 0.6), \min(0.4, 0.8)\} = 0.4.
 \end{aligned}$$

Similarly,  $\mu_B(y)$  at  $y = f(1, -1)$

$$\begin{aligned}
 &= \mu_B(y) \text{ at } y = f(-1, 1) \\
 &= \max\{\min(0.6, 0.8), \min(0.2, 0.7)\} = 0.6, \text{ and} \\
 &\mu_B(y) \text{ at } y = f(1, 0) \\
 &= \mu_B(y) \text{ at } y = f(0, 1) \\
 &= \max\{\min(0.6, 0.6), \min(0.4, 0.7)\} = 0.6, \text{ and}
 \end{aligned}$$

$$\begin{aligned}
 &\mu_B(y) \text{ at } y = f(-1, -1) \\
 &= \min(0.2, 0.8) = 0.2, \\
 &\mu_B(y) \text{ at } y = f(0, 0) \\
 &= \min(0.4, 0.6) = 0.4, \\
 &\text{and } \mu_B(y) \text{ at } y = f(1, 1) \\
 &= \min(0.6, 0.7) = 0.6.
 \end{aligned}$$

Consequently,  $B = \{0.4/f(-1, 0), 0.6/f(-1, 1), 0.6/f(1, 0), 0.2/f(-1, -1), 0.4/f(0, 0), 0.6/f(1, 1)\} = \{0.4/-1, 0.6/0, 0.6/1, 0.2/-2, 0.4/0, 0.6/2\}$

## 2.9 Fuzzy Relations

Let  $X$  and  $Y$  be two arbitrary universal sets in the real plane. A fuzzy relation between sets  $X$  and  $Y$  is given by

$$R(x, y) = \{\mu_R(x, y) / (x, y) \mid (x, y) \in X \times Y\} \quad (2.48)$$

where  $\mu_R(x, y)$  denotes the membership of relation  $R(x, y)$ . The following example illustrates a fuzzy relation.

**Example 2.8:** Let  $X = \{x_1, x_2, x_3\}$  and  $Y = \{y_1, y_2\}$  and suppose we are interested in determining a fuzzy relation  $R(x, y)$ , where the distance between  $x$  and  $y$  is

close to zero for all  $x \in X$  and for all  $y \in Y$ . One typical function that can describe this is

$$R(x, y) = \exp[-(x-y)^2]. \quad (2.49)$$

How does the above expression describe the fuzzy relation 'x is close to y'? The answer to this follows from the explanation below.

When  $x = y$ ,  $R(x, y) = 1$ ; and when the difference between  $x$  and  $y$  is very large  $R(x, y)$  approaches 0. Thus for a small difference between  $x$  and  $y$  the  $R(x, y)$  is large and close to 1, and for a large difference between  $x$  and  $y$  the  $R(x, y)$  is close to 0. To be specific, let  $X = \{1, 2, 3\}$  and  $Y = \{1, 2\}$ . Then we can express  $R(x, y)$  by

$$\begin{aligned} R(x, y) &= \{\exp[-(1-1)^2]/(1, 1), \exp[-(1-2)^2]/(1, 2), \exp[-(2-1)^2]/(2, 1), \\ &\quad \exp[-(2-2)^2]/(2, 2), \exp[-(3-1)^2]/(3, 1), \exp[-(3-2)^2]/(3, 2)\} \\ &= \{1.0/(1, 1), 0.43/(1, 2), 0.43/(2, 1), 1.0/(2, 2), 0.16/(3, 1), 0.43/(3, 2)\}. \end{aligned}$$

Generally, a relational matrix is used to describe a fuzzy relation. For instance, the fuzzy relation: 'x is close to y' can be described as

$$R(x, y) = \begin{array}{c} y \rightarrow \\ \downarrow \end{array} \begin{array}{c} x \\ \downarrow \\ \begin{array}{|c|c|c|} \hline & \mathbf{1} & \mathbf{2} \\ \hline \mathbf{1} & 1.0 & 0.43 \\ \hline \mathbf{2} & 0.43 & 1.0 \\ \hline \mathbf{3} & 0.16 & 0.43 \\ \hline \end{array} \end{array}$$

The membership values  $\mu_R(x, y)$  in the matrix is shown by faint numbers and the x- and y- values are denoted by bold numbers. Representation of a fuzzy relation by matrices has many advantages to be explored gradually.

The fuzzy relation introduced above represents relationship between 2 fuzzy sets. So, it is called *binary fuzzy relation*. A generalized fuzzy relation on the other hand represents relationship among many fuzzy sets. A formal definition of a (generalized) fuzzy relation is presented below.

**Definition 2.14:** A **fuzzy relation** is a fuzzy set defined in the Cartesian product of crisp sets  $X_1, X_2, \dots, X_n$ . A fuzzy relation  $R(x_1, x_2, \dots, x_n)$  thus is defined as

$$R(x_1, x_2, \dots, x_n)$$

$$= \{\mu_R(x_1, x_2, \dots, x_n) / (x_1, x_2, \dots, x_n) \mid (x_1, x_2, \dots, x_n) \in X_1 \times X_2 \times \dots \times X_n\} \quad (2.50)$$

where  $\mu_R: X_1 \times X_2 \times \dots \times X_n \rightarrow [0, 1]$ .

A binary fuzzy relation is a special case of the generalized fuzzy relation where instead of  $n$  universes we need only 2 universes.

## 2.10 Projection of Fuzzy Relations

Let us first consider a binary fuzzy relation  $R(x, y)$  defined on the Cartesian product of the universes  $X$  and  $Y$ . The projection of  $R(x, y)$  on  $X$ , denoted by  $R_1$  is given by

$$\mu_{R1}(x) = \max_{y \in Y} [\mu_R(x, y)] \quad (2.51)$$

Similarly, the projection of  $R(x, y)$  on  $Y$ , denoted by  $R_2$  is given by

$$\mu_{R2}(y) = \max_{x \in X} [\mu_R(x, y)] \quad (2.52)$$

**Example 2.9:** This example illustrates the principle of projection of a fuzzy relation  $R(x, y)$  on 2 fuzzy universes  $X$  and  $Y$ . We take the fuzzy relation constructed in the last section. The projection of  $R(x, y)$  on  $X$  universe is computed by determining the maximum element in each row, and its dimension will be same as the number of columns in  $R$ . The projection of  $R(x, y)$  on  $Y$  universe is also computed similarly by determining the maximum element in each column of  $R$  and its dimension will be same as the number of rows in  $R$ .

		$y \rightarrow$	1	2	
$x \downarrow$					
			1.0	0.43	1.0
1			0.43	1.0	1.0
2			0.16	0.43	1.0
3					0.43

1.0	1.0	X-projection
-----	-----	--------------

Y-projection
--------------

**Fig. 2.9:** Illustrating X- and Y- projection of a fuzzy relation.

Until now we considered projection of a binary fuzzy relation. Now, we extend the principle of projection for generalized fuzzy relations.

**Definition 2.15:** *Projection of a fuzzy relation  $R(x_1, x_2, \dots, x_n)$  on to  $X_i \times X_j \times \dots \times X_k$  for any  $i, j$  and  $k$  in  $[1, n]$  is defined as a fuzzy relation  $R_p$  where*

$$R_p(x_i, x_j, \dots, x_k) = \{ \max_{x_i \in X_i, x_j \in X_j, \dots, x_k \in X_k} \mu_R(x_i, x_j, \dots, x_k) / (x_i, x_j, \dots, x_k) \}. \quad (2.53)$$

## 2.11 Cylindrical Extension of Fuzzy Relations

Informally, cylindrical extension from a X-projection means filling all the columns of the relational matrix by the X-projection. Similarly cylindrical extension from a Y-projection means filling all the rows of the relational matrix by the Y-projection. Examples illustrating the construction of cylindrical extension from the X- and the Y-projections are given in Fig. 2.10 and Fig. 2.11 respectively.

1.0	1.0
1.0	1.0
0.43	0.43

Cylindrical extension                            X-projection

**Fig. 2.10:** Construction of cylindrical extension from X-projection.

Cylindrical extension

1.0	1.0
1.0	1.0
1.0	1.0

1.0                                            Y-projection

**Fig. 2.11:** Construction of cylindrical extension from the Y-projection.

A cylindrical extension of a fuzzy relation  $R_1(x)$  thus can be defined on  $X \times Y$  as a binary fuzzy relation given by

$$R_{1c}(x, y) = \{\mu_{R_1}(x)/ (x, y)\}. \quad (2.54)$$

The suffix  $c$  of  $R_1$  in the above expression denotes its cylindrical extension. A cylindrical extension from a projected fuzzy relation of  $m$  dimension,  $m > 1$  is presented below.

**Definition 2.16:** Let  $R_1$  be a fuzzy relation in  $X_i \times X_j \times \dots \times X_k$  for any  $(i, j, \dots, k)$  in  $[1, n]$ . The cylindrical extension of  $R_1$  to  $X_1 \times X_2 \times \dots \times X_n$  is a fuzzy relation  $R_{1c}$  given by

$$R_{1c}(x_1, x_2, \dots, x_n) = \{\mu_{R_1}(x_1, x_2, \dots, x_n)/ (x_1, x_2, \dots, x_n)\} \quad (2.55)$$

## 2.12 Fuzzy Max-Min and Max-Product Composition Operation

Let us consider 2 fuzzy relations  $R_1$  and  $R_2$  defined on  $X \times Y$  and  $Y \times Z$  respectively. The *max-min composition* of  $R_1$  and  $R_2$  is a fuzzy set defined by

$$R_3 = R_1 \circ R_2$$

$$= \{\mu_{R_3}(x, z)/ (x, z)\}$$

$$\text{where } \mu_{R_3}(x, z) = \max_y \{\min(\mu_{R_1}(x, y), \mu_{R_2}(y, z)) \mid x \in X, y \in Y, z \in Z\}. \quad (2.56)$$

When expressed as relational matrices, the computation of  $R_1 \circ R_2$  is straightforward like matrix multiplication with the replacement of sum by max and product by min operators. The following example illustrates the computation process of the max-min composition operation.

**Example 2.10:** Let  $X = \{x_1, x_2, x_3\}$ ,  $Y = \{y_1, y_2\}$  and  $Z = \{z_1, z_2\}$  be three universe of discourses and  $R_1(x, y)$  and  $R_2(y, z)$  are two fuzzy relations on  $X \times Y$  and  $Y \times Z$  respectively. Suppose  $R_1$  and  $R_2$  denote 'x is close to y' and 'y is close to z' respectively. Given  $R_1$  and  $R_2$  as follows we are interested to compute the relation  $R_3$  that denotes 'x is close to z'.

$$\text{Let } R_1 = \begin{pmatrix} 0.1 & 0.2 \\ 0.4 & 0.5 \\ 0.7 & 0.8 \end{pmatrix} \quad \text{and } R_2 = \begin{pmatrix} 0.9 & 0.8 \\ 0.7 & 0.6 \end{pmatrix}$$

Then  $R_3 = R_1 \circ R_2$

$$\begin{aligned}
 &= \begin{pmatrix} \max\{\min(0.1, 0.9), \min(0.2, 0.7)\} & \max\{\min(0.1, 0.8), \min(0.2, 0.6)\} \\ \max\{\min(0.4, 0.9), \min(0.5, 0.7)\} & \max\{\min(0.4, 0.8), \min(0.5, 0.6)\} \\ \max\{\min(0.7, 0.9), \min(0.8, 0.7)\} & \max\{\min(0.7, 0.8), \min(0.8, 0.6)\} \end{pmatrix} \\
 &= \begin{pmatrix} 0.2 & 0.2 \\ 0.5 & 0.5 \\ 0.7 & 0.7 \end{pmatrix}
 \end{aligned}$$

It may be added that the  $(i, j)$ th element of the relational matrix  $R_3$  indicates the membership of closeness of  $x_i$  and  $z_j$ . By notation  $R_3(i, j) = \mu_{R3}(x_i, z_j)$ .

It is interesting to note that the max-min composition operation satisfies the following properties. Let  $P, Q$  and  $R$  be 3 relational matrices defined on  $X \times Y$ ,  $Y \times Z$  and  $Z \times W$  respectively. Also assume that the matrices composed by  $\circ$  operator have dimensional compatibility.

$$P \circ (Q \circ R) = (P \circ Q) \circ R \quad (\text{associative}) \quad (2.57)$$

$$P \circ (Q \cup R) = (P \circ Q) \cup (P \circ R) \quad (\text{distributive over union}) \quad (2.58)$$

$$P \circ (Q \cap R) \subseteq (P \circ Q) \cap (P \circ R) \quad (\text{weakly distributive over intersection}) \quad (2.59)$$

$$Q \subseteq R \Rightarrow P \circ Q \subseteq P \circ R \quad (\text{monotonic}) \quad (2.60)$$

Besides max-min composition, max-product composition operator is also prevalent in the literature of fuzzy sets. A formal definition of max-product composition operation is introduced below.

Given 2 fuzzy relations  $R_1(x, y)$  and  $R_2(y, z)$  defined on  $X \times Y$  and  $Y \times Z$  respectively. The fuzzy relation  $R_3(x, z)$  defined on  $X \times Z$ , can be obtained by max-product composition operator as outlined below:

$$R_3(x, z) = \{\mu_{R3}(x, z) / (x, z)\}$$

$$\text{where } \mu_{R_3}(x, z) = \max_y [\mu_{R_1}(x, y) * \mu_{R_2}(y, z)] \quad (2.61)$$

The asterisk (\*) in the last expression denotes algebraic multiplication. The max-product composition between  $R_1$  and  $R_2$  is symbolically denoted in this book by an asterisk. So,  $R_3$  can be written as  $R_1 * R_2$ . The computation of  $R_3$  is illustrated below in example 2.9.

**Example 2.11:** Assuming same  $R_1$  and  $R_2$  as in example 2.10, we evaluate  $R_3$  in this example by max-product composition. Replacing the min by product in the relational matrix  $R_3$  (vide example 2.10) we find:

$$R_3 = R_1 * R_2$$

$$\begin{aligned}
&= \left( \begin{array}{cc} \max\{(0.1 * 0.9), (0.2 * 0.7)\} & \max\{(0.1 * 0.8), (0.2 * 0.6)\} \\ \max\{(0.4 * 0.9), (0.5 * 0.7)\} & \max\{(0.4 * 0.8), (0.5 * 0.6)\} \\ \max\{(0.7 * 0.9), (0.8 * 0.7)\} & \max\{(0.7 * 0.8), (0.8 * 0.6)\} \end{array} \right) \\
\\
&= \begin{pmatrix} 0.14 & 0.12 \\ 0.36 & 0.32 \\ 0.63 & 0.56 \end{pmatrix}.
\end{aligned}$$

This is all about the computation of  $R_3$ .

## 2.13 Fuzzy Linguistic Hedges

Fuzzy systems are capable of representing sentences containing terms 'more or less', 'about to', 'very slow', 'a little way' and the like. These are called *linguistic hedges*. Representation of the linguistic hedges by membership functions is necessary for modeling fuzzy systems. In this section we first define *linguistic variables* and then introduce linguistic hedges by membership functions.

---

**Definition 2.17:** A **linguistic variable** is characterized by a 4-tuple:  $\langle x, LV, DR, \mu \rangle$  where

*x is the name of the linguistic variable,*

*LV is the linguistic values that x can take,*

*DR is the dynamic range of the linguistic variable,*

*and  $\mu$  is the membership function of the linguistic variable x for all linguistic values supplied in LV.*

**Example 2.12:** The linguistic variable defined above is illustrated in this example. Let *age* be a linguistic variable. So, by our definition  $x = \text{age}$ . Let us assume that age can assume the following linguistic values: YOUNG, OLD, VERY-OLD etc. So,  $LV = \{\text{YOUNG}, \text{OLD}, \text{VERY-OLD}\}$ . The dynamic range DR for age is  $[0, 120]$  years. We also need to consider the membership functions of age in the fuzzy sets YOUNG, OLD and VERY-OLD. In other words, the membership functions  $\mu_{\text{YOUNG}}(\text{age})$ ,  $\mu_{\text{OLD}}(\text{age})$  and  $\mu_{\text{VERY-OLD}}(\text{age})$  should be defined for the range of age in  $[0, 120]$  years.

Given a linguistic variable  $x = \text{age}$  say. We can qualify YOUNG or OLD further by VERY-YOUNG, VERY-VERY-YOUNG, MORE-OR-LESS-YOUNG, VERY-OLD, NOT-SO-OLD, NOT-SO-YOUNG etc. How can we represent these linguistic hedges by membership functions?

Suppose we know the membership function  $\mu_{\text{YOUNG}}(\text{age})$ , the membership function VERY-YOUNG then can be defined as follows:

$$\mu_{\text{VERY-YOUNG}}(\text{age}) = [\mu_{\text{YOUNG}}(\text{age})]^2. \quad (2.62)$$

Similarly, we can define the membership functions VERY-VERY-YOUNG as

$$\mu_{\text{VERY-VERY-YOUNG}}(\text{age}) = [\mu_{\text{YOUNG}}(\text{age})]^3 \quad (2.63)$$

and MORE-OR-LESS-YOUNG as

$$\mu_{\text{MORE-OR-LESS-YOUNG}}(\text{age}) = [\mu_{\text{YOUNG}}(\text{age})]^{0.5}. \quad (2.64)$$

The following definitions are generally used to represent linguistic hedges by membership functions.

**Definition 2.18:** Let  $\mu_A(x)$  be membership function of a linguistic variable  $x$  in a fuzzy set  $A$ . The operation **concentration (CON)** and **dilation (DIL)** are then defined by the following membership functions:

$$\mu_{\text{CON-}A}(x) = [\mu_A(x)]^2 \quad (2.65)$$

$$\mu_{\text{DIL-}A}(x) = [\mu_A(x)]^{0.5}. \quad (2.66)$$

Fuzzy linguistic hedges such as VERY or VERY-VERY can be best described by the concentration operation. On the other hand, linguistic hedges MORE-OR-LESS, AROUND etc. are usually denoted by the dilation operation.

Another interesting operation, well-known as **contrast intensification** increases the values of  $\mu_A(x)$ , when it is above 0.5 and diminishes those which are below 0.5. The following definition presents one way of contrast intensification.

**Definition 2.19:** The operation **contrast intensification** on a linguistic value  $A$  is defined by the following membership function.

$$\left. \begin{aligned} \mu_{\text{INT-}A}(x) &= 2 [\mu_A(x)]^2 \text{ for } 0 \leq \mu_A(x) < 0.5, \\ &= 1 - 2(1 - \mu_A(x))^2 \text{ for } 0.5 \leq \mu_A(x) \leq 1, \end{aligned} \right\} \quad (2.67)$$

Membership functions of the linguistic variables in a fuzzy set are generally constructed in a manner so as to satisfy the criteria of orthogonality, presented below.

**Definition 2.20:** Let  $T = \{t_1, t_2, \dots, t_n\}$  be a term set of a linguistic variable  $x$  on the universe  $X$ . The term set  $T$  is called orthogonal if it satisfies the following criterion:

$$\sum_{i=1}^n \mu_{t_i}(x) = 1, \quad \forall x \in X, \quad (2.68)$$

where  $t_i$ 's are convex and normal fuzzy sets defined on  $X$ .

**Example 2.13:** Consider a fuzzy set AGE. Let CLOSE-TO-18 and CLOSE-TO-20 and CLOSE-TO-22 be 3 fuzzy linguistic values of the variable age in the range [18, 22] years. We now construct the membership functions of these fuzzy sets in a manner such that sum of the membership of these fuzzy sets over the

age variable is 1 and the fuzzy sets are convex. One way of constructing the above fuzzy sets is illustrated below.

$$\mu_{\text{CLOSE-TO-18}}(\text{age}) = \{1.0/18, 0.0/20, 0.0/22\}$$

$$\mu_{\text{CLOSE-TO-20}}(\text{age}) = \{0.0/18, 1.0/20, 0.0/22\} \text{ and}$$

$$\mu_{\text{CLOSE-TO-22}}(\text{age}) = \{0.0/18, 0.0/20, 1.0/22\}$$

It may be noted that all the above fuzzy sets are convex since each membership function has a single rising peak. Further, sum of the memberships at any age is always 1.

## 2.14 Summary

The chapter presented an introduction to fuzzy sets, fuzzy relations and a few important fuzzy operators and concepts. Among the fuzzy operators, t-norm and s-norm have massive applications in fuzzy reasoning systems. The concept of projection of fuzzy relations also has significant applications in the logic of fuzzy sets. The chapter also introduced different types of membership functions. Selection of these functions in a particular application calls for a domain specific knowledge of the users. For example, the YOUNG fuzzy set can be represented by a Gaussian type membership function, whereas the fuzzy set MODERATE-SPEED of a mechanical moving system can be described by a trapezoidal membership function.

Linguistic variables are of a great concern in designing a fuzzy system. Design of an orthogonal set of linguistic terms is usually very difficult. So, in most cases a near orthogonal linguistic set of terms, where the sum of membership at all values of the dynamic range of the linguistic variables is approximately equal to one, is preferred.

## Exercise

- According to Ohm's law the current passing through a resistive device causes a potential drop across the device, and the drop thus obtained is proportional to the amplitude of the current.

Assume that we have 3 ammeters and 2 voltmeters, which are used in 6 possible combinations to measure the resistance. Let the readings obtained be denoted by a  $(I, V)$  pair in mA and volts respectively, where

$$(I_1, V_1) = (1, 10), \quad (I_2, V_2) = (1.1, 10.1), \quad (I_3, V_3) = (0.9, 9.1),$$

$$(I_4, V_4) = (0.95, 9.52), (I_5, V_5) = (1.15, 11.6) \text{ and } (I_6, V_6) = (1.2, 11.2).$$

Let the absolute value of the resistance identified from its color code be 10K-ohm. Design a fuzzy set that describes GOOD-MEASUREMENT.

**[Hints:** Compute  $R_i = V_i/I_i$  for  $i = 1$  to 6. Determine the absolute value of the deviations  $\text{abs}(R_i - R_{\text{theoretical}})$ . Then the measurement is good for those  $i$  where the absolute value of deviation is small, and those measurements should have a membership close to 1. When the absolute value of deviation is large, the membership should be small. One membership function is given below.

$$\mu_{\text{GOOD-MEASUREMENT}}(R_i)$$

$$= 1 - \left\{ \frac{\text{abs}(R_i - R_{\text{theoretical}})}{\max_i \text{abs}(R_i - R_{\text{theoretical}})} \right\}$$

2. Apply dilation and concentration on the constructed fuzzy set in Exercise 1 to determine the fuzzy sets MORE-OR-LESS-GOOD-MEASUREMENT and VERY-GOOD-MEASUREMENT.
3. Determine the membership distribution of the fuzzy set NEITHER-VERY-GOOD-NOR-VERY-POOR-MEASUREMENT for the problem given in Exercise 1.

**[Hints:** Try with  $\max\{(1-\mu_{\text{VERY-GOOD}}(R_i)), (1-\mu_{\text{VERY-POOR}}(R_i))\}$ .]

4. Given a fuzzy set  $A = \{0.1/1, 0.2/-1, 0.2/2, 0.4/-2, 0.3/3\}$ . Also given a function  $f(x) = x^2$ . Determine  $B = f(A)$  by using the extension principle.

$$\begin{aligned} \text{[Hints: } B=f(A)= & \{ \max(0.1, 0.2)/1^2, \max(0.2, 0.4)/2^2, 0.3/3^2 \} \\ & = \{0.2/1, 0.4/4, 0.3/9\} \] \end{aligned}$$

5. Given  $f(x) = x^2 + 4$ , find  $f(A)$  by extension principle for the following fuzzy set  $A = \{0.1/2, 0.3/-2, 0.6/-3\}$ .

**[Answer:**  $f(A) = \{0.3/8, 0.6/13\}$ ]

6. Find the X- and Y-projections of the following relational matrix.

	x	y→	
	↓		
		1      2      3	
R=	5	0.8   0.9   0.6	
	6	0.2   0.4   0.7	
	7	0.1   0.2   0.5	

[**Answer:** X-projection = {0.9/5, 0.7/6, 0.5/7} and  
Y-projection = {0.8/1, 0.9/2, 0.7/3}.]

7. Verify with an example of two ( $3 \times 3$ ) fuzzy relational matrices that DeMorgan's theorem presented below holds good for the matrices:

$$(R_1 \circ R_2)^c = (R_1^c \Phi R_2^c)^c,$$

where  $R_1$  and  $R_2$  are 2 relational matrices of compatible order,  $c$  denotes the one's complementation operation over the elements of the matrix;  $\circ$  and  $\Phi$  denote max-min composition and min-max composition operator respectively. The min-max composition operator is applied like max-min composition with the replacement of max by min and min by max.

8. x and y are two fuzzy linguistic variables in the same universe U. We define two fuzzy relations to represent that 'x is close to y'. Which of these two relations can represent a sharp estimation of CLOSENESS of x and y?

a)  $R_1_{\text{CLOSE-TO}}(x, y) = \exp[-(x - y)^2]$

b)  $R_2_{\text{CLOSE-TO}}(x, y) = \exp[-\text{abs}(x - y)]$

[**Answer:** (a) because the square term helps falling off the exponential function for a slightly large difference between x and y.]

## References

- [1] Dubois, D. and Prade, H., *Fuzzy sets and Systems: Theory and Applications*, Academic press, NY, 1980.
- [2] Jang, J.-S. R., Sun, C.- T. and Mizutani, E., *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*, Prentice-Hall, Upper Saddle River, NJ, 1997.
- [3] Konar, A., *Artificial Intelligence and Soft Computing: Behavioral and Cognitive Modeling of the Human Brain*, CRC press, Boca Raton, Florida, 1999.
- [4] Pedrycz, W. and Gomide, F., *An Introduction to Fuzzy Sets: Analysis and Design*, MIT Press, Cambridge, Massachusetts, 1998.
- [5] Wang, Li-Xin, *A Course in Fuzzy Systems and Control*, Prentice-Hall International, Upper Saddle River, NJ, 1997.
- [6] Zadeh, L. A., “Fuzzy sets,” *Information and Control*, vol. 8, pp. 338-353, 1965.
- [7] Zimmermann, H. J., *Fuzzy Set Theory and Its Applications*, Kluwer Academic, Dordrecht, The Netherlands, 1991.

# 3

# Fuzzy Logic and Approximate Reasoning

*A production system generally embodies a set of rules, called knowledge base, a set of facts called database and an inference engine (interpreter) for interpretation of the database with the help of the knowledge base. In the process of interpretation of the database, the inference engine generates new inferences. The mechanism of generation of inferences is well known as reasoning in the treaties of knowledge based systems. Reasoning in predicate logic is usually performed by three fundamental rules, such as modus ponens, modus tollens and syllogisms. The chapter extends the scope of reasoning in knowledge based systems through generalization of the above three rules by using the logic of fuzzy sets. Various forms of fuzzy reasoning with single and multiple antecedent clauses have been introduced in the chapter and the scope of one such reasoning scheme on a VLSI engine has been examined. The chapter ends with a discussion on the principles of fuzzy abductive reasoning.*

## 3.1 Production Systems

Since the inception of Artificial Intelligence, production systems have been playing an important role in solving intelligent problems. A typical production system (PS) is represented as a 3-tuple:

$PS = \langle PR, WM, IE \rangle$ .

where

$PR$  denotes a set of If-Then rules, also called production rules;

$WM$  represents a working memory, that holds the current instantiation for the variables present in the If-part (also called antecedent part) of the rules, and

$IE$  represents a inference engine that derives new consequences, also called inferences, by applying the production rules on the data/instantiations available in the  $WM$ .

Generally the IF-part of a production rule contains a number of clauses connected by AND operators. So, the clauses in the IF-part of the rule are also referred to as AND-clauses. A production rule is said to fire when all the AND-clauses present in the rule have matching data in the  $WM$ . On firing of the rule, a new consequence is generated following the Then-part, also called the consequent part, of the rule. The consequence thus generated is added to the list of data items in the  $WM$  for subsequent firing of another rule. Sometimes the consequent part of a rule suggests deletion of one or more data elements from the  $WM$ . On firing of such rules, the data elements as suggested by the rule are deleted from the  $WM$ .

**Example 3.1:** This example illustrates the rule firing and inference generation process in a PS. Consider a production system consisting of 2 production rules:  $PR_1$  and  $PR_2$  and a  $WM$  containing a set of data elements as presented below:

$PR_1$ : IF ( $x$  is a bird)  
THEN ( $x$  can fly).

$PR_2$ : IF ( $x$  is a bird) and  
( $x$  is black) and  
( $x$  has red eyes) and  
( $x$  lays eggs in crows' cage)  
THEN ( $x$  sings in the spring).

$WM = \{cuckoo \text{ is a bird}, cuckoo \text{ is black}, cuckoo \text{ has red eyes},$   
 $cuckoo \text{ lays eggs in crows' cage}, parrot \text{ is a bird}\}$

The IE in the present context matches the IF-part of  $PR_1$  with the data available in the  $WM$ , and finds 2 matching patterns: 'cuckoo is a bird' and 'parrot is a bird'. Consequently,  $PR_1$  is firable for 2 matching patterns and it fires twice

yielding 2 inferences: 'cuckoo can fly' and 'parrot can fly'. The resulting inferences are added to the data elements in the WM. The production system in turn matches the AND clauses of PR<sub>2</sub> one by one in the available data elements of the WM. Since all the matching patterns are found, the rule fires generating the inference 'cuckoo sings in the spring'. The inference is also added to the elements in the WM.

## 3.2 Conflict Resolution in Production Systems

One important aspect of a production system is to fire a single rule at a time. There are, however, occasions when more than one rule has matching patterns in WM and thus is ready for firing concurrently. Conflict resolution strategies are employed to select one of several such rules for firing. The conflict resolution strategies ensure firing of a single rule from multiple firable rules in a production system. The following 3 strategies [2] are generally used for the resolution of conflicts in a production system.

◆ **Refractoriness:** This strategy reveals that a given rule should not be applied on the same set of data again. To implement this, the earlier instantiations of a fired rule should be isolated in some way to prevent firing of an already fired rule with the old data.

◆ **Recency:** This strategy ensures that the inferences generated from the most recently fired rule should be used for firing subsequent rules. The advantage of this is to continue reasoning always at the leading edge of the problem solving process without doubling back to the old data again and again.

◆ **Specificity:** This strategy ensures that in case more than one rule is ready to fire, then the rule having more number of AND-clauses should fire because the latter rule is more specific.

In our previous example, although both the rules PR<sub>1</sub> and PR<sub>2</sub> are ready for firing concurrently (if there is no order restriction in firing of rules), PR<sub>2</sub> being more specific should fire.

## 3.3 Drawbacks of Traditional Production Systems

The principle of rule instantiation and matching in a traditional production system is well known as the *recognition-act cycle*. In the recognition phase, the AND-clauses of the rules are matched with the instantiations present in the working memory. The act phase on the other hand is concerned with firing a

rule and addition of new inferences or deletion of existing inferences from the WM.

Unfortunately, the recognition phase in a practical production system occasionally fails because of the absence of appropriate variable bindings (instantiations) in the WM. Consequently, the recognition-act cycles have a premature termination. In order to continue the recognition-act cycles with the available data of the WM, the rules having partially matched variable bindings in the WM need to be fired. The partial matching of the AND-clauses with the instantiations in the WM can be accomplished by using the logic of fuzzy sets.

For example consider the following production rule that relates the speed of a person with his/ her height.

*PR: IF (height is TALL)  
THEN (speed is HIGH).*

Suppose the WM contains the data: height is VERY-TALL. Then by using fuzzy logic we will be able to infer that 'speed is VERY-HIGH'. It may be noted that the data item of the WM here does not match exactly with antecedent clause of the rule. But the rule still fires and a new consequence is generated. In the subsequent part of this chapter we will discuss the details of the above type of reasoning that allows partial matching [5].

### 3.4 Fuzzy Implication Rules

*Reasoning* informally refers to generation of inferences from a given set of facts and rules. The subject *logic is concerned* with the formalization of methodology and principles of reasoning. Production rules are synonymously called implication rules in traditional logic. Implication rules of traditional logic are extended with fuzzy linguistic variables. In this section, the implication rules in the classical (propositional/ predicate) logic and their extension in fuzzy domain are introduced.

Let us try to formalize the rule: *IF x is a banana and x is yellow THEN x is ripe* in predicate logic. For this formalization, we define 3 predicates  $\text{Banana}(x)$ ,  $\text{Yellow}(x)$  and  $\text{Ripe}(x)$ , where each of these predicates has only 2 possible truth values: true or false. Using a IF-THEN (implication) operator, the above rule can be written as:

$$\text{Banana}(x), \text{Yellow}(x) \rightarrow \text{Ripe}(x).$$

where the comma in the left side of the implication sign ( $\rightarrow$ ) denotes logical conjunction (AND) of the antecedent predicates.

In order to allow instantiation of the antecedent predicates with the contents of the WM, we consider 6 fuzzy sets: YELLOW, VERY-YELLOW, MORE-OR-LESS-YELLOW, RIPE, VERY-RIPE and MORE-OR-LESS-RIPE. Fuzzy extensions of the last rule then may be re-stated as follows:

*Rule 1: IF a banana is YELLOW  
THEN it is RIPE.*

*Rule 2: IF a banana is VERY-YELLOW  
THEN it is VERY-RIPE.*

*Rule 3: IF a banana is MORE-OR-LESS-YELLOW  
THEN it is MORE-OR-LESS-RIPE.*

It may be added here that unlike production systems, the logic of fuzzy sets allows firing of these 3 rules concurrently in presence of a data element concerning color of a banana in the WM. Thus conflict resolution is not employed in fuzzy logic.

Formally, let  $x$  be a linguistic variable in a universe  $X$ , and  $A_1, A_2$  and  $A_3$  are three fuzzy sets under the universe  $X$ . Also assume that  $y$  be another linguistic variable in a universe  $Y$  and  $B_1, B_2$  and  $B_3$  are 3 fuzzy sets under  $Y$ . Then the implication rules between variable  $x$  and  $y$  may be described as

*Rule1: IF  $x$  is  $A_1$  THEN  $y$  is  $B_1$ .*

*Rule2: IF  $x$  is  $A_2$  THEN  $y$  is  $B_2$ .*

*Rule 3: IF  $x$  is  $A_3$  THEN  $y$  is  $B_3$ .*

Suppose the WM contains  $x$  is  $A'$ , where  $A'$  is semantically close to  $A_1, A_2$  and  $A_3$  respectively. In the subsequent sections, we demonstrate the evaluation procedure of  $y$  is  $B'$  from the known membership distributions of  $x$  is  $A_1$ ,  $y$  is  $B_1$ ,  $x$  is  $A_2$ ,  $y$  is  $B_2$ ,  $x$  is  $A_3$ ,  $y$  is  $B_3$  and  $x$  is  $A'$ .

### 3.5 Fuzzy Implication Relations

A fuzzy implication relation [1] for a given rule: IF  $x$  is  $A_i$  THEN  $y$  is  $B_i$  is formally denoted by

$$R_i(x, y) = \{\mu_{R_i}(x, y) / (x, y)\} \quad (3.1)$$

where the membership function  $\mu_{R_i}(x, y)$  is constructed intuitively by many alternative ways. Some typical implication relations are presented below.

◆ **Dienes-Rescher Implication:** This implication relation is a direct consequence of the implication function in classical propositional logic. In the logic of propositions, given 2 propositions  $p$  and  $q$ , then the implication  $p \rightarrow q$  (to be read as *if p then q*) can be proved to be equivalent to  $\neg p \vee q$ , where  $\neg$  and  $\vee$  denote logical negation and OR operation respectively. Inquisitive readers naturally have the question: how it happens? The implication *if p then q* states that *p is true but q is false is impossible*. Logically this means  $p \wedge \neg q$  is false, where  $\wedge$  is a logical AND operator. Alternatively,  $\neg(p \wedge \neg q)$ , which by De Morgan's law yields  $\neg p \vee q$ , is true. Consequently, the result  $p \rightarrow q \equiv \neg p \vee q$  follows, where  $\equiv$  denotes an equivalence operator.

Dienes-Rescher Implication relation extends the classical propositional implication formula by replacing negation by one's complement and logical OR by max operator. Thus, membership function for the implication rule: IF  $x$  is  $A_i$  THEN  $y$  is  $B_i$  is given by

$$\mu_{Ri}(x, y) = \text{Max} [1 - \mu_{Ai}(x), \mu_{Bi}(y)]. \quad (3.2)$$

◆ **Lukasiewicz implication:** The propositional implication function can easily be extended in fuzzy logic by replacing negation by one's complement and logical OR by sum (+) operator. Thus, for the given fuzzy implication rule: IF  $x$  is  $A_i$  THEN  $y$  is  $B_i$ , the membership function of the rule may be stated as  $1 - \mu_{Ai}(x) + \mu_{Bi}(y)$ . Since, the result may exceed 1 we express

$$\mu_{Ri}(x, y) = \text{Min} [1, 1 - \mu_{Ai}(x) + \mu_{Bi}(y)] \quad (3.3)$$

which is known as the **Lukasiewicz implication** function [10].

◆ **Mamdani Implication:** Mamdani proposed the following two implication functions [7, 12].

$$\mu_{Ri}(x, y) = \text{Min} [\mu_{Ai}(x), \mu_{Bi}(y)] \quad (3.4)$$

$$\text{or } \mu_{Ri}(x, y) = \mu_{Ai}(x) \mu_{Bi}(y) \quad (3.5)$$

Mamdani implication functions are most widely used implications in fuzzy systems and fuzzy control engineering. This implication relation is constructed based on the assumption that fuzzy IF-THEN rules are local. For example, consider the implication rule: IF height is TALL THEN speed is HIGH. By Mamdani's implication function, we do not want to mean: IF height is SHORT THEN speed is SLOW. The second rule is rather an example of a non-local rule. The knowledge engineer thus has to decide whether he prefers local or non-local rules. If he prefers local rules then Mamdani's implication relation should be used.

◆ **Zadeh Implication:** Zadeh's implication function [3, 4] is another form of extension of the classical propositional implication function. In the logic of propositions, the propositional implication  $p \rightarrow q$  can alternatively be stated as '*either p and q are true or p is false*'. Thus,  $p \rightarrow q$  is equivalent to  $(p \wedge q) \vee (\neg p)$ .

Representing logical AND by min, logical OR by max and negation by one's complement we can state the membership function of the fuzzy implication rule: IF  $x$  is  $A_i$  THEN  $y$  is  $B_i$  by

$$\mu_{R_i}(x, y) = \text{Max} [\text{Min}(\mu_{A_i}(x), \mu_{B_i}(y)), 1 - \mu_{A_i}(x)] \quad (3.6)$$

This is known as Zadeh implication relation.

◆ **Godel Implication:** The Godel implication is a popular implication formula in traditional logic. A fuzzy extension of the formula for the rule: IF  $x$  is  $A_i$  THEN  $y$  is  $B_i$  can be stated by representing the membership function of the relation by

$$\left. \begin{aligned} \mu_{R_i}(x, y) &= 1 \text{ if } \mu_{A_i}(x) \leq \mu_{B_i}(y) \\ &= \mu_{B_i}(y), \text{ otherwise.} \end{aligned} \right\} \quad (3.7)$$

There are many other implication functions commonly used in the logic of fuzzy sets. A complete list of them is available in any standard textbook on fuzzy sets [10].

## 3.6 Fuzzy Logic

The logic of fuzzy sets, also called fuzzy logic, is an extension of the classical propositional logic from 2 perspectives. First, instead of binary valuation space (truth/falsehood) of the propositional logic, fuzzy logic provides a multi-valued truth-space in  $[0, 1]$ . Secondly, propositional logic generates inferences based on the complete matching of the antecedent clauses with the available data, whereas fuzzy logic is capable of generating inferences even when a partial matching of the antecedent clauses against data elements in WM exists. In this section, we present 3 typical propositional inference rules and describe their possible extensions in fuzzy logic.

### 3.6.1 Typical Propositional Inference Rules

Let  $p$ ,  $q$  and  $r$  be 3 propositions. The following 3 propositional inference rules are commonly used for logical inferencing.

- ◆ **Modus Ponens:** Given a proposition  $p$  and a propositional implication rule  $p \rightarrow q$ , we can derive the inference  $q$ . Symbolically,

$$p \wedge (p \rightarrow q) \Rightarrow q \quad (3.8)$$

where  $\Rightarrow$  denotes a logical provability operator. This above inference rule is well known as modus ponens.

- ◆ **Modus Tollens:** Given a proposition  $\neg q$ , and the implication rule  $p \rightarrow q$ , we can derive the inference  $\neg p$ . Symbolically,

$$\neg q \wedge (p \rightarrow q) \Rightarrow \neg p. \quad (3.9)$$

The above inference rule is known as modus tollens.

- ◆ **Hypothetical Syllogism:** Given 2 implication rules  $p \rightarrow q$  and  $q \rightarrow r$ , then we can easily derive a implication  $p \rightarrow r$ . Symbolically,

$$(p \rightarrow q) \wedge (q \rightarrow r) \Rightarrow p \rightarrow r. \quad (3.10)$$

The above inference rule is popularly known as hypothetical syllogism or chain rule.

### 3.6.2 Fuzzy Extension of the Inference Rules

The logic of fuzzy set provides a general framework for the extension of the above 3 propositional inference rules. Fuzzy extension of modus ponens, modus tollens and hypothetical syllogisms are called generalized modus ponens, generalized modus tollens and generalized hypothetical syllogism respectively.

- ◆ **Generalized Modus Ponens (GMP):** Consider a fuzzy production rule: IF  $x$  is  $A$  then  $y$  is  $B$ , and a fuzzy fact:  $x$  is  $A'$ . The GMP inference rule then infers  $y$  is  $B'$ . Here  $A$ ,  $B$ ,  $A'$  and  $B'$  are fuzzy sets such that  $A'$  is close to  $A$ , and  $B'$  is close to  $B$ . The inference rule also states that the closer the  $A'$  to  $A$ , the closer the  $B'$  to  $B$ . Symbolically, the GMP can be stated as follows:

Given: IF  $x$  is  $A$  THEN  $y$  is  $B$ .

Given:  $x$  is  $A'$

---

Inferred:  $y$  is  $B'$

---

- ◆ **Generalized Modus Tollens (GMT):** Given a fuzzy production rule: IF  $x$  is  $A$  THEN  $y$  is  $B$ , and a fuzzy fact  $y$  is  $B'$ , the GMT then infers  $x$  is  $A'$ , where the more is the difference between  $B'$  and  $B$ , the more is the difference between  $A'$  and  $A$ . Symbolically, the GMT is stated as follows:

Given: IF  $x$  is  $A$  THEN  $y$  is  $B$ .

Given:  $y$  is  $B'$

---

Inferred:  $x$  is  $A'$ .

---

- ◆ **Generalized Hypothetical Syllogism (GHS):** Given 2 fuzzy production rules: IF  $x$  is  $A$  THEN  $y$  is  $B$ , and IF  $y$  is  $B'$  THEN  $z$  is  $C'$ , where  $A$ ,  $B$  and  $C$  are 3 fuzzy sets, and  $B'$  is close to  $B$ . Then the GHS infers the fuzzy fact  $z$  is  $C'$ , where  $C'$  is close to  $C$ . The closer the  $B'$  to  $B$ , the closer the  $C'$  to  $C$ . Symbolically, we can state this rule as follows:

Given: IF  $x$  is  $A$  THEN  $y$  is  $B$ .

Given: IF  $y$  is  $B'$  THEN  $z$  is  $C$

---

Inferred:  $z$  is  $C'$ .

---

In the above definition of the inference rules, we just mentioned that  $A'$  is close to  $A$ ,  $B'$  is close to  $B$  and the like. But we did not mention what we exactly mean by "close to". In fact "close to" can take any of the following forms: VERY, VERY-VERY, MORE-OR-LESS, NOT, ABOUT-TO, AROUND and other fuzzy hedges that means a fuzzy set  $A'$  is approximately similar to  $A$ .

### 3.7 The Compositional Rule of Inference

In this section we present the methodology for the evaluation of fuzzy inferences for GMP, GMT and GHS. This, however, calls for formalization of a fuzzy rule called the compositional rule of inference. The compositional rule of inference is usually applied to 2 fuzzy membership distribution, one of which usually have a smaller number of linguistic variables. *The rule extends the latter membership*

distribution cylindrically, so as to increase its number of linguistic variables to the former distribution. The intersection of the former and the resulting distribution is then projected to desired axes. The whole process is referred to as the compositional rule of inference. How exactly the compositional rule of inference is applied to determine the fuzzy inferences in GMP, GMT and GHS is presented below.

### 3.7.1 Computing Fuzzy Inferences in GMP

Given the rule: IF  $x$  is  $A$  THEN  $y$  is  $B$ , and the observed membership distribution  $x$  is  $A'$ , we by GMP infer:  $y$  is  $B'$ . For evaluation of membership distribution of:  $y$  is  $B'$ ,  $\mu_B'(y)$ , we need to know the membership distribution of  $x$  is  $A'$ ,  $\mu_A'(x)$ , and the membership of the fuzzy relation for the given IF-THEN rule,  $\mu_R(x, y)$ . This is accomplished by applying the compositional rule of inference over  $\mu_A'(x)$  and  $\mu_R(x, y)$ .

Here  $x$  and  $y$  are linguistic variables on universes  $X$  and  $Y$  respectively. Now, in order to apply the compositional rule, we extend  $\mu_A'(x)$  cylindrically and the extended distribution is, suppose,  $\mu_{A\text{CYL}}(x, y)$ . Let the intersection of  $\mu_{A\text{CYL}}(x, y)$  and  $\mu_R(x, y)$  be denoted by  $\mu_{A\text{CYL}\cap R}(x, y)$ . Then  $\mu_B'(y)$  is computed by projecting  $\mu_{A\text{CYL}\cap R}(x, y)$  on  $Y$ . Symbolically, the whole process includes the following 3 steps:

$$\mu_{A\text{CYL}}(x, y) = \mu_A'(x) \quad (3.11)$$

$$\begin{aligned} \mu_{A\text{CYL}\cap R}(x, y) &= \mu_{A\text{CYL}}(x, y) \cap \mu_R(x, y) \\ &= t[\mu_{A\text{CYL}}(x, y), \mu_R(x, y)] \end{aligned} \quad (3.12)$$

$$\mu_B'(y) = \max_{x \in X} t[\mu_{A\text{CYL}}(x, y), \mu_R(x, y)] \quad (3.13)$$

$$= \max_{x \in X} t[\mu_A'(x), \mu_R(x, y)] \quad (3.14)$$

If we take min for t-norm then the above result can directly be obtained for discrete fuzzy system by employing the max-min composition operator as presented below.

$$\mu_B'(y) = \mu_A'(x) \circ \mu_R(x, y). \quad (3.15)$$

where  $\mu_A'(x)$  and  $\mu_R(x, y)$  are row vector and matrices of compatible dimensions.

**Example 3.2:** This example illustrates the computation of membership of the inferences generated using GMP. Let

$$\mu_A'(x) = [0.8 \ 0.9 \ 0.2]$$

and  $\mu_R(x, y) = \begin{pmatrix} 0.8 & 0.6 & 0.5 \\ 0.6 & 0.5 & 0.9 \\ 0.7 & 0.6 & 0.5 \end{pmatrix}$

The membership distribution of the derived inference is given by

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

$$= [0.8 \ 0.6 \ 0.9].$$

The same result can also be obtained by applying the 3 basic steps of the compositional rule of inference. Here,

$$\mu_{A \text{ CYL}}'(x, y) = \begin{pmatrix} 0.8 & 0.8 & 0.8 \\ 0.9 & 0.9 & 0.9 \\ 0.2 & 0.2 & 0.2 \end{pmatrix}$$

$$\mu_{A \text{ CYL} \cap R}'(x, y) = t[\mu_{A \text{ CYL}}'(x, y), \mu_R(x, y)]$$

$$= \begin{pmatrix} 0.8 & 0.8 & 0.8 \\ 0.9 & 0.9 & 0.9 \\ 0.2 & 0.2 & 0.2 \end{pmatrix} \quad t \quad \begin{pmatrix} 0.8 & 0.6 & 0.5 \\ 0.6 & 0.5 & 0.9 \\ 0.7 & 0.6 & 0.5 \end{pmatrix}$$

$$= \begin{pmatrix} 0.8 & 0.6 & 0.5 \\ 0.6 & 0.5 & 0.9 \\ 0.2 & 0.2 & 0.2 \end{pmatrix}$$

Projecting the resulting distribution on Y-axis we find  $\mu_B'(y) = [0.8 \ 0.6 \ 0.9]$  which is same as obtained directly by using the max-min composition operator.

### 3.7.2 Computing Fuzzy Inferences Using GMT

Given a fuzzy production rule: IF  $x$  is  $A$  THEN  $y$  is  $B$ , and a fact  $y$  is  $B'$ , we by GMT infer  $x$  is  $A'$ . In this section we present the principle of determining the membership distribution of  $x$  is  $A'$ ,  $\mu_A'(x)$ , from the membership distribution of  $y$  is  $B'$ ,  $\mu_B'(y)$ , and the membership  $\mu_R(x, y)$  of the fuzzy relation between the antecedent and consequent part of the given rule. The computation involves cylindrical extension of  $\mu_B'(y)$  to  $\mu_{B' CYL}(x, y)$ , then intersection of  $\mu_{B' CYL}(x, y)$  with  $\mu_R(x, y)$  and finally projection of the resulting relation on X-axis. Thus, following the same steps as in the case of GMP, it can easily be shown that

$$\mu_A'(x) = \max_{y \in Y} [\min\{\mu_B'(y), \mu_R(x, y)\}]. \quad (3.16)$$

When  $\mu_B'(y)$  and  $\mu_R(x, y)$  are discrete relations represented by a row vector and a matrix respectively, we can represent the above result by the following max-min composition operation:

$$\mu_A'(x) = \mu_B'(y) \circ [\mu_R(x, y)]^T \quad (3.17)$$

where T denotes the transposition operator over the given relation.

**Example 3.3:** This example illustrates the computation of GMT using max-min composition operator.

We take the same  $\mu_R(x, y)$  as in Example 3.2, and the  $\mu_B'(y)$  we obtained as the result in that example, and plan to determine  $\mu_A'(x)$  by the compositional rule of inference. Thus,

$$\begin{aligned} \mu_A'(x) &= \mu_B'(y) \circ [\mu_R(x, y)]^T \\ &= [0.8 \ 0.6 \ 0.9] \circ \begin{pmatrix} 0.8 & 0.6 & 0.5 \\ 0.6 & 0.5 & 0.9 \\ 0.7 & 0.6 & 0.5 \end{pmatrix}^T \\ &= [0.8 \ 0.9 \ 0.7]. \end{aligned}$$

It may be noted that  $\mu_A'(x)$  thus obtained is not same as that presumed in Example 3.2.

### 3.7.3 Computing Fuzzy Inferences Using GHS

Given two fuzzy production rules: IF  $x$  is A THEN  $y$  is B, and IF  $y$  is B THEN  $z$  is C, we by GHS infer IF  $x$  is A THEN  $z$  is C. Suppose, the membership  $\mu_R(x, y)$  of the relation  $R(x, y)$  and the membership  $\mu_R(y, z)$  of the relation  $R(y, z)$  are supplied, and we want to determine the membership  $\mu_R(x, z)$  of the relation  $R(x, z)$ . We can solve the problem first by extending  $\mu_R(x, y)$  to  $\mu_{R\text{ CYL}}(x, y, z)$ , then by taking intersection of  $\mu_{R\text{ CYL}}(x, y, z)$  with  $\mu_R(y, z)$ , and finally projecting the resulting distribution onto X and Z axes. The whole process can be described by the following expression:

$$\mu_R(x, z) = \max_{y \in Y} t[\mu_R(x, y), \mu_R(y, z)] \quad (3.18)$$

We can represent the above expression in a closed form by using the max-min composition operation:

$$\mu_R(x, z) = \mu_R(x, y) \circ \mu_R(y, z). \quad (3.19)$$

**Example 3.4:** Consider the following two fuzzy production rules (PR):

*PR<sub>1</sub>: IF age is YOUNG THEN digestion-rate is HIGH.*

*PR<sub>2</sub>: IF digestion-rate is HIGH THEN speed is HIGH.*

Now, by GHS we infer: IF age is YOUNG THEN speed is HIGH. Let  $\mu_R(\text{age}, \text{digestion-rate})$  and  $\mu_R(\text{digestion-rate}, \text{speed})$  be the membership distribution for the relation between the antecedent and the consequent part of the first rule and the second rules respectively. We want to determine the membership  $\mu_R(\text{age}, \text{speed})$  of the relation between the antecedent and the consequent part of the derived rule.

Let

$$\mu_R(\text{age}, \text{digestion-rate}) = \begin{array}{c} \text{digestion-rate} \rightarrow \\ \text{age} \rightarrow \end{array} \left[ \begin{array}{ccc} 0.2 & 0.3 & 0.6 \\ 0.4 & 0.6 & 0.5 \\ 0.3 & 0.6 & 0.9 \end{array} \right]$$

and  $\mu_R(\text{digestion-rate}, \text{speed}) = \begin{array}{c} \text{speed} \rightarrow \\ \text{digs-rate} \rightarrow \end{array} \left[ \begin{array}{ccc} 0.8 & 0.7 & 0.5 \\ 0.7 & 0.7 & 0.4 \\ 0.7 & 0.4 & 0.5 \end{array} \right]$ .

Then we can easily evaluate

$$\mu_R(\text{age}, \text{speed}) = \mu_R(\text{age}, \text{digestion-rate}) \circ \mu_R(\text{digestion-rate}, \text{speed})$$

$$\begin{aligned}
 & \text{speed} \rightarrow \\
 & = \text{age} \downarrow \quad \left( \begin{array}{ccc} 0.6 & 0.4 & 0.5 \\ 0.6 & 0.6 & 0.5 \\ 0.7 & 0.6 & 0.5 \end{array} \right)
 \end{aligned}$$

### 3.8 Approximate Reasoning with Multiple Antecedent Clauses

This section provides a general scheme for approximate reasoning with multiple antecedent clauses in the production rule. Consider, for example, the following fuzzy IF-THEN rule with 2 antecedent clauses:

$$\text{IF } x \text{ is } A \text{ and } y \text{ is } B \text{ THEN } z \text{ is } C,$$

where  $x$ ,  $y$  and  $z$  are three linguistic variables in the universes  $X$ ,  $Y$  and  $Z$  respectively, and  $A$ ,  $B$  and  $C$  are three fuzzy sets under the respective universes.

Let  $\mu_A(x)$ ,  $\mu_B(y)$  and  $\mu_C(z)$  be the membership distributions of linguistic variables  $x$ ,  $y$  and  $z$  to belong to  $A$ ,  $B$  and  $C$  respectively. Then the membership distribution of the clause "x is A and y is B" is given by  $t(\mu_A(x), \mu_B(y))$ .

Denoting the above t-norm as the antecedent membership  $AM$  and the membership  $\mu_C(z)$  as the consequent membership  $CM$ , we can define the membership distribution  $\mu_R(x, y; z)$  of the implication rule by

$$\text{Min } [1, 1 - AM + CM] \quad (\text{Lukasiewicz implication})$$

$$\left. \begin{array}{l} \text{AM CM} \\ \text{Min (AM, CM)} \end{array} \right\} \quad (\text{Mamdani implication})$$

$$\text{Max } [1 - AM, CM] \quad (\text{Dienes-Rescher implication})$$

Max [Min (AM, CM), 1 - AM] (Zadeh implication)

$$\left. \begin{array}{l} 1, \text{ if } AM \leq CM \\ CM, \text{ otherwise } \end{array} \right\} \quad (\text{Godel implication})$$

The IF-THEN production rules of the form: IF  $x_1$  is A1 and  $x_2$  is A2.... and  $x_n$  is An THEN y is B with n number of antecedent clauses thus will have the above type of membership distribution where

$$AM = t((\mu_{A1}(x_1), \mu_{A2}(x_2), \dots, \mu_{An}(x_n))) \text{ and } CM = \mu_B(y).$$

The reasoning mechanism for fuzzy production rules with multiple antecedent clauses is illustrated below with a typical GMP.

**Example 3.5:** Given the fuzzy production rule (PR) with two antecedent clauses.

*PR: IF x is A and y is B THEN z is C.*

Suppose, the observed distribution of x is  $A'$ ,  $\mu_A'(x)$ , and y is  $B'$ ,  $\mu_B'(y)$ , are supplied, and we want to determine the membership distribution of z is  $C'$ ,  $\mu_C'(z)$ .

Assuming Lukasiewicz implication function, the membership distribution of the given antecedent-consequent relationship can be described as

$$\mu_R(x, y; z)$$

$$= \text{Min} [1, 1 - AM + CM]$$

$$= \text{Min} [1, 1 - t(\mu_A(x), \mu_B(y)) + \mu_C(z)]$$

$$= \text{Min} [1, 1 - \min(\mu_A(x), \mu_B(y)) + \mu_C(z)] \text{ (Taking min as the t-norm)}$$

If  $R(x, y; z)$  is a discrete relation, we can represent  $\mu_R(x, y; z)$  as a relational matrix. Suppose,  $x \in X = \{x_1, x_2, x_3\}$ ,  $y \in Y = \{y_1, y_2\}$  and  $z \in Z = \{z_1, z_2, z_3\}$ , then  $\mu_R(x, y; z)$  can be described as a  $(6 \times 3)$  matrix with row indices  $x_1y_1, x_1y_2, x_2y_1, x_2y_2, x_3y_1, x_3y_2$  and column indices  $z_1$  and  $z_2$ . We can now determine  $\mu_C'(z)$  by the following composition rule:

$$\mu_C'(z) = t(\mu_A'(x), \mu_B'(y)) \circ \mu_R(x, y; z) \quad (3.20)$$

It is indeed important to note that  $t(\mu_A'(x), \mu_B'(y))$  has 6 components corresponding to  $xy \in \{x_1y_1, x_1y_2, x_2y_1, x_2y_2, x_3y_1, x_3y_2\}$  and it can assume any typical function, such as min.

**Example 3.6:** This example illustrates the numerical computations involved in example 3.5. Consider the fuzzy production rule with 2 antecedents:

*IF height is TALL and weight is MODERATE THEN speed is HIGH.*

Suppose  $\mu_{TALL}(\text{height}) = \{0.5/ 5', 0.8/ 6', 1.0/ 7'\}$ ,

$\mu_{MODERATE}(\text{weight}) = \{0.7/ 45\text{Kg}, 0.9/ 50\text{ Kg}\}$  and

$\mu_{HIGH}(\text{speed}) = \{0.6/ 6\text{m/s}, 0.8/ 8\text{m/s}, 0.5/ 9\text{m/s}\}$  are given.

Suppose,  $\mu_{TALL}'(\text{height}) = \{0.6/ 5', 0.7/ 6', 0.9/ 7'\}$ ,

$\mu_{MODERATE}'(\text{weight}) = \{0.8/ 45\text{Kg}, 0.7/ 50\text{ Kg}\}$

are also given and we want to determine  $\mu_{HIGH}'(\text{speed})$ . It is of course assumed that the fuzzy sets  $TALL' \approx TALL$ ,  $MODERATE' \approx MODERATE$  and  $HIGH' \approx HIGH$ .

For solving the above problem, we first construct the fuzzy relation  $R(\text{height}, \text{weight}; \text{speed})$ , the membership function  $\mu_R(\text{height}, \text{weight}; \text{speed})$  of which is computed in 2 phases.

*First phase: Computation of the t-norm.*

$$AM = t(\mu_{TALL}(\text{height}), \mu_{MODERATE}(\text{weight}))$$

$$= \min(\mu_{TALL}(\text{height}), \mu_{MODERATE}(\text{weight}))$$

$$= \{(0.5 \wedge 0.7)/(5', 45\text{ Kg}), (0.5 \wedge 0.9)/(5', 50\text{ Kg}), (0.8 \wedge 0.7)/(6', 45\text{ Kg}), \\ (0.8 \wedge 0.9)/(6', 50\text{ Kg}), (1.0 \wedge 0.7)/(7', 45\text{ Kg}), (1.0 \wedge 0.9)/(7', 50\text{ Kg})\}$$

$$= \{(0.5)/(5', 45\text{ Kg}), (0.5)/(5', 50\text{ Kg}), (0.7)/(6', 45\text{ Kg}), \\ (0.8)/(6', 50\text{ Kg}), (0.7)/(7', 45\text{ Kg}), (0.9)/(7', 50\text{ Kg})\}.$$

*Second phase: Computation of the implication by Lukasiewicz function.*

Results of the computation is presented below in matrix  $\mu_R(\text{height}, \text{weight}; \text{speed})$ . For convenience of the readers, we illustrate computation of one matrix element only corresponding to the row index  $5', 45\text{ Kg}$  and column index  $6\text{ m/s}$ .

Here,  $\mu_R(5', 45 \text{ Kg}, 6 \text{ m/s}) = \text{Min}[1, (1 - 0.5 + 0.6) = 1.0]$ . Remaining elements of  $\mu_R$  have been computed similarly.

		6 m/s	8 m/s	9 m/s
		1.0	1.0	1.0
$\mu_R(\text{height, weight; speed}) =$	$5', 45 \text{ Kg}$	1.0	1.0	1.0
	$5', 50 \text{ Kg}$	1.0	1.0	1.0
	$6', 45 \text{ Kg}$	0.9	1.0	0.8
	$6', 50 \text{ Kg}$	0.8	1.0	0.7
	$7', 45 \text{ Kg}$	0.9	1.0	0.8
	$7', 50 \text{ Kg}$	0.7	0.9	0.6

The last part of the problem is to determine the membership distribution of the fuzzy inference: speed is HIGH'. This can be done by composing the t-norm  $t(\mu_{\text{TALL}}'(\text{height}), \mu_{\text{MODERATE}}'(\text{weight}))$  with  $\mu_R(\text{height, weight; speed})$ . Symbolically,

$$\begin{aligned} & \mu_{\text{HIGH}}'(\text{speed}) \\ &= t(\mu_{\text{TALL}}'(\text{height}), \mu_{\text{MODERATE}}'(\text{weight})) \circ \mu_R(\text{height, weight; speed}). \end{aligned}$$

Taking min as the t-norm, we can easily compute the above max-min composition to determine  $\mu_{\text{HIGH}}'(\text{speed})$ . This is left as an exercise for the young readers.

### 3.9 Approximate Reasoning with Multiple Rules

In this section we present a general scheme for approximate reasoning by GMP with multiple rules each having one antecedent clause. Let  $x$  and  $y$  be two linguistic variables in the universes  $X$  and  $Y$  respectively. Also let  $A_1, A_2, \dots, A_n$  be fuzzy sets under the universe  $X$  and  $B_1, B_2, \dots, B_n$  be fuzzy sets under the universe  $Y$ . We consider the following fuzzy production rules (PR):

PR1: IF  $x$  is  $A_1$  THEN  $y$  is  $B_1$ .

PR2: IF  $x$  is  $A_2$  THEN  $y$  is  $B_2$ .

PR3: IF  $x$  is  $A_3$  THEN  $y$  is  $B_3$ .

..... .....

..... .....

PRn: IF  $x$  is  $A_n$  THEN  $y$  is  $B_n$ .

Let  $R_i(x, y)$  be the fuzzy relation between the antecedent and the consequent part of production rule  $PR_i$ . We can then easily define the membership distribution of  $R_i$  by

$$\mu_{R_i}(x, y) = f_i(\mu_{A_i}, \mu_{B_i}) \quad (3.21)$$

where  $f_i$  is a implication function. For example, if  $f_i$  is of Mamdani type then we can write  $f_i(\mu_{A_i}, \mu_{B_i})$  as  $\min(\mu_{A_i}, \mu_{B_i})$ .

Suppose the observed distribution is  $x$  is  $A'$ , where  $A' \approx A_1, A' \approx A_2, A' \approx A_3, \dots$ , and  $A' \approx A_n$ . Then we can determine the membership distribution of  $y$  is  $B'_i$  using the  $i$ -th rule only by

$$\mu_{B'_i}(y) = \max_{x \in X} \{ \min(\mu_{A'_i}(x), \mu_{R_i}(x, y)) \} \quad (3.22)$$

Thus for  $n$  rules we determine  $\mu_{B'_i}(y)$  for  $i = 1$  to  $n$ . Now, for computation of  $\mu_B'(y)$ , we use the maximum of  $\mu_{B'_i}(y)$  for  $i = 1$  to  $n$ . Symbolically,

$$\begin{aligned} \mu_B'(y) &= \max_{i=1}^n [\mu_{B'_i}(y)] \\ &= \max_{i=1}^n [\max_{x \in X} \{ \min(\mu_{A'_i}(x), \mu_{R_i}(x, y)) \}] \end{aligned} \quad (3.23)$$

For discrete membership distributions we can evaluate  $\mu_B'(y)$  by taking the maximum of the max-min composition operation of  $\mu_{A'_i}(x)$  and  $\mu_{R_i}(x, y)$ . Thus,

$$\mu_B'(y) = \max_{i=1}^n [\mu_{A'_i}(x) \circ \mu_{R_i}(x, y)] \quad (3.24)$$

**Example 3.7:** This example illustrates the computation of  $\mu_B'(y)$  using the following 2 production rules:

*PR1: IF height is TALL THEN speed is HIGH.*

*PR2: IF height is MEDIUM THEN speed is MODERATE.*

Suppose the membership distribution of  $\mu_{TALL}(\text{height})$ ,  $\mu_{HIGH}(\text{speed})$ ,  $\mu_{MEDIUM}(\text{height})$  and  $\mu_{MODERATE}(\text{speed})$  are given from which we can construct the fuzzy relational matrices  $R_1$  and  $R_2$  for rule1 and rule2 respectively. For

brevity, we instead of computing the matrices, directly provide the matrices as follows:

$$R_1 = \begin{matrix} \text{height} \rightarrow \\ \begin{pmatrix} 0.4 & 0.5 & 0.6 \\ 0.5 & 0.7 & 0.8 \\ 0.6 & 0.7 & 0.9 \end{pmatrix} \end{matrix} \quad \text{and} \quad R_2 = \begin{matrix} \text{height} \downarrow \\ \begin{pmatrix} 0.6 & 0.9 & 0.7 \\ 0.5 & 0.8 & 0.6 \\ 0.8 & 0.7 & 0.5 \end{pmatrix} \end{matrix}$$

Let the membership distribution of 'height' is ABOVE-AVERAGE' is given, and we want to determine the membership distribution of 'speed is ABOVE-NORMAL'. Suppose,

$$\mu_{\text{ABOVE-AVERAGE}}(\text{height}) = \{0.5/5', 0.9/6', 0.8/7'\}$$

and the entries the relational matrices  $R_1$  and  $R_2$  correspond to the heights 5', 6' and 7' respectively in order. Then we can determine  $\mu_{\text{ABOVE-NORMAL}}(\text{speed})$  by the following formula:

$$\begin{aligned} & \mu_{\text{ABOVE-NORMAL}}(\text{speed}) \\ &= \max[\mu_{\text{ABOVE-AVERAGE}}(\text{height}) \circ R_1, \mu_{\text{ABOVE-AVERAGE}}(\text{height}) \circ R_2] \\ &= \mu_{\text{ABOVE-AVERAGE}}(\text{height}) \circ \max[R_1, R_2] \quad (\text{by rules given in section 2.12.}) \end{aligned}$$

$$= [0.5 \ 0.9 \ 0.8] \circ \begin{pmatrix} 0.6 & 0.9 & 0.7 \\ 0.5 & 0.8 & 0.8 \\ 0.8 & 0.7 & 0.9 \end{pmatrix}$$

$$= [0.8 \ 0.8 \ 0.8]$$

In the above computation,  $\max [R_1, R_2]$  was evaluated by taking the position-wise maximum of the matrices  $R_1$  and  $R_2$  like matrix addition, with the replacement of addition operation by max operation.

If the speeds referenced in  $R_1$  and  $R_2$  are 5m/s, 7m/s and 9m/s respectively in order then the membership distribution of  $\mu_{\text{ABOVE-NORMAL}}(\text{speed})$  should be described as follows:

$$\mu_{\text{ABOVE-NORMAL}}(\text{speed}) = \{0.8/ 5\text{m/s}, 0.8/ 7\text{m/s}, 0.8/ 9\text{m/s}\}.$$

### 3.10 Scope of Parallelism in Approximate Reasoning Using Mamdani Implication Function

A special case of approximate reasoning with Mamdani implication function that leads to massive parallelism and pipelining in the process of generating fuzzy inferences is introduced in this section. Continuing from the last section, we can thus determine the membership distribution of  $y$  is  $B'$  as follows.

$$\mu_{B'}(y) = \max_{i=1}^n [\max_{x \in X} \{\min(\mu_{Ai}'(x), \mu_{Ri}(x, y))\}] \quad (3.25)$$

Now, using min type Mamdani implication function, we can express  $\mu_{Ri}(x, y)$  by

$$\mu_{Ri}(x, y) = \min(\mu_{Ai}(x), \mu_{Bi}(y)) \quad (3.26)$$

Substituting the above in (3.25) we find

$$\mu_{B'}(y) = \max_{i=1}^n [\max_{x \in X} \{\min(\mu_{Ai}'(x), \min(\mu_{Ai}(x), \mu_{Bi}(y)))\}] \quad (3.27)$$

$$= \max_{i=1}^n [\max_{x \in X} \{\min(\min(\mu_{Ai}'(x), \mu_{Ai}(x)), \mu_{Bi}(y))\}] \quad (3.28)$$

$$= \max_{i=1}^n [\max_{x \in X} \{\min(\alpha_i, \mu_{Bi}(y))\}] \quad (3.29)$$

where  $\alpha_i = \min(\mu_{Ai}'(x), \mu_{Ai}(x))$ .

Expression (3.29) is due to Togai and Watanabe [9] who realized it into a fuzzy VLSI inference engine. This was the first VLSI engine capable of generating fuzzy inferences on a single chip. Togai and Watanabe have shown that this VLSI chip on being mounted on a microcomputer board can generate as many as 68,000 fuzzy logical inferences per second.

### 3.11 Realization of Fuzzy Inference Engine on VLSI Architecture

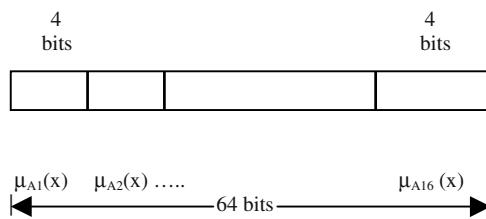
Togai and Watanabe [9] considered a set of fuzzy production rules of the following form.

Rule: IF  $x$ -is  $A_i$  THEN  $y$ -is  $B_i$

where  $A_i$  and  $B_i$  each can assume 16 possible subsets, i.e.  $1 \leq i \leq 16$ . Moreover, the membership distributions  $\mu_{Ai}(x)$  and  $\mu_{Bi}(y)$  can assume membership values from the following set:

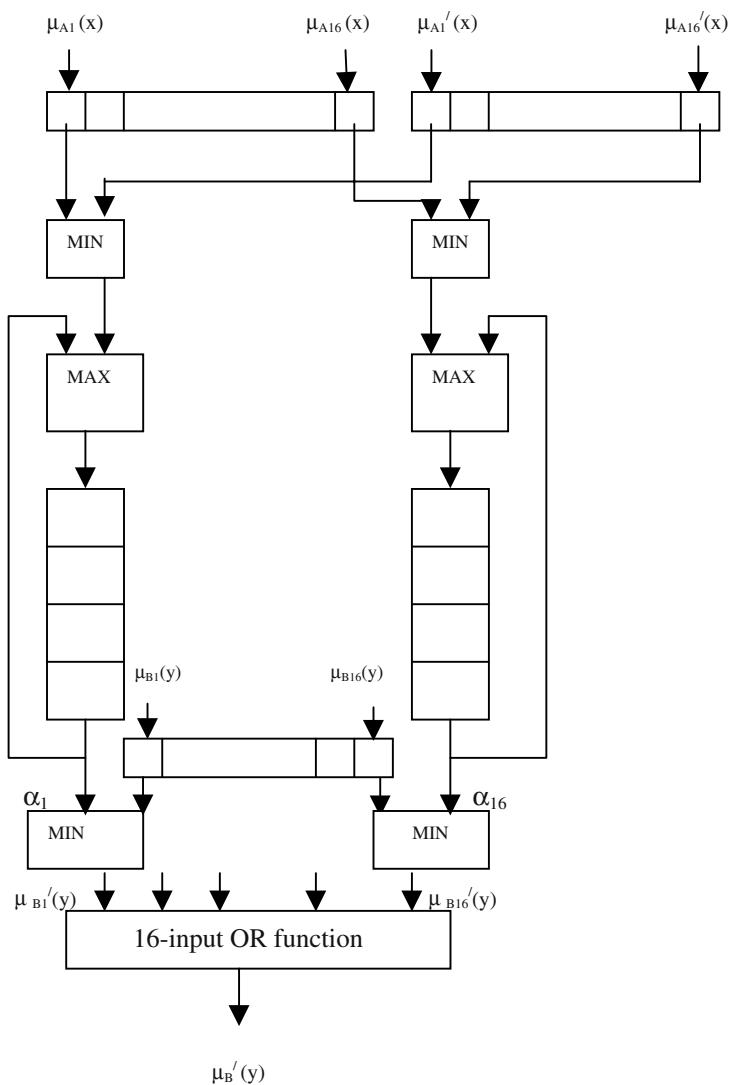
$$\text{Membership set} = \{0, 1/15, 2/15, 3/15, \dots, 14/15, 1\},$$

the elements of which, if multiplied by a scale factor of 15, can be converted to 4-bit binary numbers. Thus to represent  $[\mu_{A1}(x) \mu_{A2}(x) \dots \mu_{A16}(x)]$  they used 64 bit registers, 4-bit each for one field depicting  $\mu_{Ai}(x)$ , as presented in Fig. 3.1.

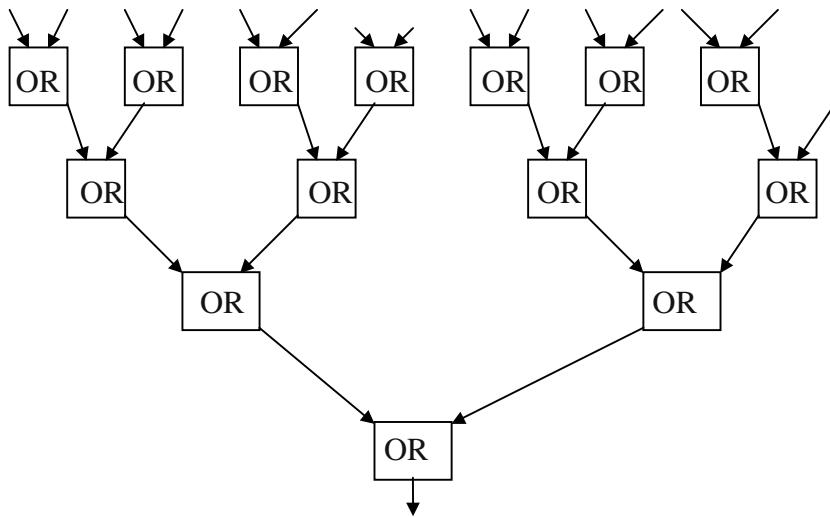


**Fig. 3.1:** The 64-bit register to hold  $\mu_{Ai}(x)$ .

For  $\mu_{Bi}(y)$ s,  $\mu_A'(x)$  and  $\mu_B'(y)$ s, we also use 3 more registers, each of 64 bits. For computing  $\alpha_i$ 's, Togai and Watanabe used the computational scheme (Fig. 3.2). The fuzzy MIN boxes in Fig. 3.2 determine the minimum of the two input signals applied to them. The 4-bit shift register holds the cumulative maximum of the two successive outputs of the MIN box. The  $\alpha_i$ 's thus produced are ANDed with  $\mu_{Bi}(y)$ s to yield  $\mu_{Bi}'(y)$ s, which finally are ORed to produce the  $\mu_B'(y)$  distribution. The 16-input OR function is computed with the help of an OR tree, shown in Fig. 3.3. The system implemented by Togai and Watanabe [9] has an execution speed of 68,000 fuzzy logical inferences per second, when a 20.8 M-Hz crystal is used as a basic timing unit of the VLSI chip



**Fig. 3.2:** The logic architecture of the fuzzy inference engine.



**Fig. 3.3:** The OR-tree realizing the 16-input OR function in the logic architecture of Fig. 3.2. Fourteen (instead of sixteen) inputs are shown in the input layer for lack of space.

### 3.12 Approximate Reasoning with Multiple Rules Each with Multiple Antecedent Clauses

To start with let us consider  $n$  rules each with 2 antecedent clauses only as presented below:

*PR1: IF  $x$  is  $A_1$  and  $y$  is  $B_1$  THEN  $z$  is  $C_1$ .*

*PR2: IF  $x$  is  $A_2$  and  $y$  is  $B_2$  THEN  $z$  is  $C_2$ .*

*PR3: IF  $x$  is  $A_3$  and  $y$  is  $B_3$  THEN  $z$  is  $C_3$ .*

..... .....

..... .....

*PRn: IF  $x$  is  $A_n$  and  $y$  is  $B_n$  THEN  $z$  is  $C_n$ .*

Now, for the  $i$ -th rule membership distribution of the fuzzy implication relation is given by

$$\mu_{R_i}(x, y; z) = f_i(t(\mu_{A_i}, \mu_{B_i}), \mu_{C_i}) \quad (3.30)$$

where  $f_i$  is a implication function. If the  $t$ -norm, for example, is interpreted as the min operator, and the  $f_i$  as the Mamdani type min operator, then  $\mu_{R_i} = \min(\min(\mu_{A_i}, \mu_{B_i}), \mu_{C_i})$ .

Given an observed distribution of  $x$  is  $A'$  and  $y$  is  $B'$ , we can easily evaluate the membership distribution of  $z$  is  $C'_i$  by the  $i$ -th rule as follows:

$$\mu_{C_i}' = t(\mu_A', \mu_B') \circ \mu_{R_i}(x, y; z) \quad (3.31)$$

Consequently, taking the contribution of all  $n$  rules together, we can determine the membership distribution of  $z$  is  $C'$  by taking the max of  $\mu_{C_i}'$  for  $i=1$  to  $n$ . Symbolically,

$$\begin{aligned} \mu_C' &= \max_{i=1}^n [\mu_{C_i}'] \\ &= \max_{i=1}^n [t(\mu_A', \mu_B') \circ \mu_{R_i}(x, y; z)] \quad [\text{by (3.31)}] \\ &= \max_{i=1}^n [t(\mu_A', \mu_B') \circ f_i(t(\mu_{A_i}, \mu_{B_i}), \mu_{C_i})] \quad [\text{by (3.30)}] \end{aligned} \quad (3.32)$$

### 3.13 Fuzzy Abductive Reasoning

The reasoning methodology introduced in the last three sections was based on GMP. There are, however, situations when we need to determine the membership distribution of the antecedent clause of a rule for an observed distribution of its consequence. This is called GMT or *abductive reasoning* [8]. The adverb fuzzy has been included in the heading to distinguish it from the crisp abduction. The most important application of abductive reasoning is in diagnostic problems. In diagnostic applications, the defective item in a system needs to be identified from the measurement data. Since abnormality in measurements is mainly caused by the abnormal behavior of the defective components, determination of the defective item is feasible by reasoning with GMT.

In this section we discuss the principle of abductive reasoning by 2 chain rules listed below:

*PR1: IF  $x$  is  $A$  THEN  $y$  is  $B$ .*

*PR2: IF  $y$  is  $B$  THEN  $z$  is  $C$ .*

Suppose, the observed membership distribution of  $z$  is  $C'$  is available, and we want to determine the membership distribution of  $x$  is  $A'$ . To do so, we have to apply GMT twice, once to determine the membership of  $y$  is  $B'$  and next to determine the membership of  $x$  is  $A'$ . The whole scheme is stepwise outlined below:

- Determine the membership distribution  $\mu_{R1}(x, y)$  and  $\mu_{R2}(y, z)$  by the following formula:

$$\mu_{R1}(x, y) = f(\mu_A(x), \mu_B(y)) \quad (3.32)$$

$$\mu_{R2}(y, z) = f(\mu_B(y), \mu_C(z)) \quad (3.33)$$

where  $f$  is any typical implication function.

- Compute  $\mu_B'$  first and then  $\mu_A'$  by executing the following two steps in order:

$$\mu_B'(y) = \mu_c'(z) \circ [\mu_{R2}(y, z)]^T \quad (3.34)$$

$$\mu_A'(x) = \mu_B'(y) \circ [\mu_{R1}(x, y)]^T \quad (3.35)$$

where  $T$  denotes the transposition operation over the specified relations.

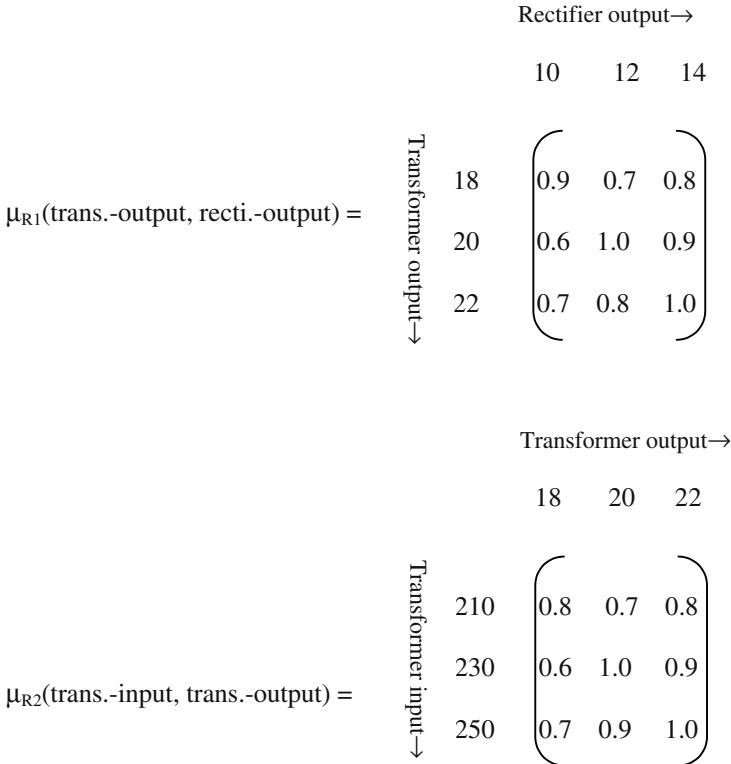
If number of chain rules increase we can easily evaluate the membership distribution of the cause from the membership distribution of its direct or chained effect by extending the above principle.

**Example 3.8:** Given below a fragment of the diagnostic knowledge base of an audio player system:

*PRI: IF transformer secondary output is CLOSE -TO 20 V  
THEN rectifier output is CLOSE-TO 12 V.*

*PR2: IF the transformer primary input is CLOSE-TO 230 V  
THEN transformer secondary output is CLOSE-TO 20 V.*

and an observation that the rectifier output is CLOSE-TO 10 V. Also suppose that the following membership distributions are available, we need to determine the membership distribution of the transformer primary input to be CLOSE-TO approximately 230V. Let the relational matrices for PR1 and PR2 be R1 and R2 respectively. Then suppose:



and  $\mu_{\text{CLOSE-TO}}(\text{rectifier-output}) = \{0.9/10V, 0.6/12V, 0.2/14V\}$ .

It is to be noted that to represent that rectifier output is CLOSE-TO 10V, we assigned a high membership value to 10V and relatively smaller membership values to 12V and 14V.

Now, by step 2 of the abductive reasoning principle, we find

$$\begin{aligned}
 & \mu_{\text{CLOSE-TO}}(\text{transformer-output}) \\
 &= \mu_{\text{CLOSE-TO}}(\text{rectifier-output}) \circ [\mu_{R1}(\text{trans.-output}, \text{recti.-output})]^T \\
 &= \{0.9/18V, 0.6/20V, 0.7/22V\}.
 \end{aligned}$$

and  $\mu_{\text{CLOSE-TO}}(\text{transformer-input})$

$$\begin{aligned}
 &= \mu_{\text{CLOSE-TO}}(\text{transformer-output}) \circ [\mu_{R2}(\text{trans.-input}, \text{trans.-output})]^T \\
 &= \{0.8/210V, 0.7/230V, 0.7/250V\}
 \end{aligned}$$

### 3.14 Conclusions

Classical reasoning models, such as modus ponens, modus tollens and syllogisms, have limited applications for their inability in rule firing with partial matching of data clauses. The chapter extended the classical models using the logic of fuzzy sets to reason with a knowledge base in presence of partially instantiable data clauses.

Among the different fuzzy reasoning schemes introduced in this chapter, multiple rules each with multiple antecedent clauses using Mamdani's implication relation is of special interest for its utility in many practical problems. The Togai-Watanabe's VLSI engine has also received much attention in the fields of fuzzy control. There is a vast literature on fuzzy reasoning, some of which employed structured models like fuzzy Petri nets [6]. Most of the applications of fuzzy reasoning in control, communication, signal processing and robotics until date employ the classical fuzzy models outlined in the chapter.

### Exercise

- Given the following rule:

*IF resistance is EXCESSIVE  
THEN current-flow is INSIGNIFICANT.*

Also given the relational matrix

$$R(\text{resistance, current}) = \begin{matrix} & \text{current} \rightarrow \\ & \quad 150\text{mA} \quad 50\text{mA} \quad 10\text{mA} \\ \text{resistance} \rightarrow \\ \begin{matrix} 10\text{k} \\ 50\text{k} \\ 100\text{k} \end{matrix} & \left( \begin{array}{ccc} 0.8 & 0.7 & 0.9 \\ 0.6 & 0.4 & 0.3 \\ 0.2 & 0.5 & 0.1 \end{array} \right) \end{matrix}$$

Suppose the membership distribution of very excessive resistance is given by

$$\mu_{\text{VERY-EXCESSIVE}}(\text{resistance}) = \{ 0.7/ 10k, 0.8/ 50k, 0.9/ 100k \}.$$

Determine the membership distribution of very insignificant current flow through the device under test.

[**Answer:**  $\mu_{\text{VERY-INSIGNIFICANT}}(\text{current}) = \{ 0.7/ 100mA, 0.7/ 50mA, 0.7/ 10mA \}$ .]

2. Suppose, the membership of “current flow through a device is very insignificant” is given by

$$\mu_{\text{VERY-INSIGNIFICANT}}(\text{current}) = \{ 0.7/100mA, 0.7/50mA, 0.7/10mA \}.$$

Also given the rule and the relational matrix introduced in Exercise 1. Determine by abductive reasoning the membership of “resistance of the device under test is very excessive”.

[**Answer:**  $\mu_{\text{VERY-EXCESSIVE}}(\text{resistance}) = \{ 0.7/ 10k, 0.6/50k, 0.5/100k \}$ .]

3. Verify the results obtained for Exercises 1 and 2 by cylindrical extension, intersection and projection operations.
4. Construct following Togai and Watanabe a parallel logic engine for fuzzy modus tollens.

$$[\text{Hints: } \mu_{A_i}'(x) = \max_{i=1}^n [\max_{y \in Y} \{ \min(\mu_{B_i}'(y), \min(\mu_{A_i}(x), \mu_{B_i}(y))) \}]$$

$$= \max_{i=1}^n [\max \{ \min(\alpha_i, \mu_{A_i}(x)) \}]$$

$$\text{where } \alpha_i = \min_{y \in Y} (\mu_{B_i}'(y), \mu_{B_i}(y)).$$

The rest is obvious.]

## References

- [1] Dubois, D. and Prade, H., *Fuzzy Sets and Systems: Theory and Applications*, Academic Press, NY, 1980.
- [2] Jackson, P., *Introduction to Expert Systems*, Addison-Wesley, Reading, MA, 1986.
- [3] Jang, J.-S. R., Sun, C.-T., Mizutani, E., *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and machine Intelligence*, Prentice-Hall, NJ, 1997.
- [4] Klir, G. J. and Yuan, B., *Fuzzy Sets and Fuzzy Logic: Theory and Applications*, Prentice-Hall, NJ, 1995.
- [5] Konar, A. and Jain, L. C., “An introduction to computational intelligence paradigms,” In *Practical Applications of Computational Intelligence Techniques*, Jain, L. and Widle, P. D. (Eds.), Kluwer Academic Press, Dordrecht, pp. 1-9, 2001.
- [6] Konar, A. and Mandal, A. K., “Uncertainty management in expert systems using fuzzy Petri nets,” *IEEE Trans. on Knowledge and Data Engineering*, vol. 8, no. 1, pp. 96-105, 1996.
- [7] Ross, T. J., *Fuzzy Logic with Engineering Applications*, McGraw-Hill, NY, 1991.
- [8] Saha, P. and Konar, A., “A heuristic algorithm for computing the max-min inverse fuzzy relation, *Int. J. of Approximate Reasoning*, Elsevier, North Holland, vol. 30, pp. 131-147, 2002.
- [9] Togai, M. and Watanabe, H., “Expert system on a chip: An engine for real time approximate reasoning,” *IEEE Expert*, pp. 55-62, Fall 1986.
- [10] Wang, L.-X., *A Course in Fuzzy Systems and Control*, Prentice-Hall, NJ, 1997.
- [11] Zadeh, L. A., “The role of fuzzy logic in the management of uncertainty in expert systems,” *Fuzzy Sets and Systems*, Elsevier, North Holland, vol. 11, pp. 199-227, 1983.
- [12] Zimmerman, H. J., *Fuzzy Set Theory and its Applications*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1991.

# 4

# Fuzzy Logic in Process Control

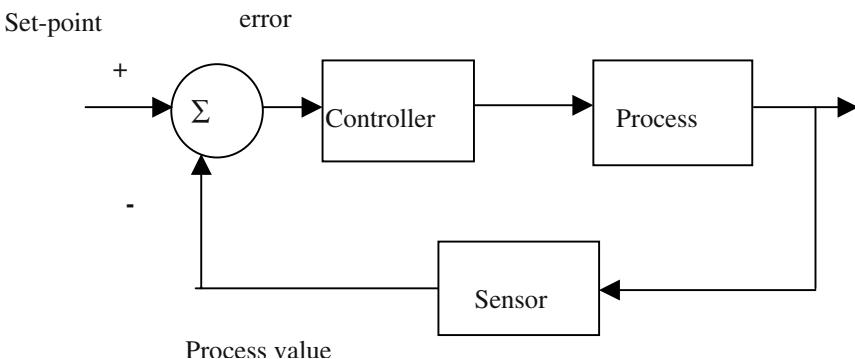
The logic of fuzzy sets and its application in approximate reasoning have already been introduced in the last two chapters. This chapter further extends the scope of approximate reasoning of fuzzy logic in industrial process control systems. Two distinct models of fuzzy control namely Mamdani's model and Takagi-Sugeno's model have been discussed in this chapter with numerical illustrations. One important aspect of controller design for smart processes is to ensure stability of the closed loop control system. The chapter provides an introduction to stability analysis for the Takagi-Sugeno model, as it nowadays is widely being used in the design of industrial fuzzy controllers. The principle of defuzzification is introduced with an example. Lastly the chapter ends with a discussion on a case study of fuzzy control of a nuclear reactor.

## 4.1 Process Control

An industrial plant/process is a system capable of changing its output following a given input. Controlling a process thus means adjustment in its input, so as to cause a necessary change in its output parameter. A thermal plant, for example, is an industrial process whose output parameter temperature is controlled around a desired level by adjustment of its heat influx. Similarly, steam pressure inside

a boiler drum is controlled around a given set-point by adjusting temperature or water level inside the drum. Alternatively, the rate of flow of a liquid through a tube is controlled by adjustment of the level of the liquid in the source buffer. A controller is a device used for automatic adjustment of the input parameter of the process so as to control its output parameter around a desired level.

A process control loop works through a negative feedback arrangement. A sensor is employed in the loop to measure the output parameter of the process in numerical quantity. A set-point for the process is fixed to describe the desired level of the process output parameter. The difference between the set-point and the measured process output parameter is called the error. The controller is designed in a manner so as to adjust the input parameter of the process based on the error signal. A simple process control loop that works following the above principle is presented in Fig. 4.1.



**Fig. 4.1:** A typical process control loop.

Processes can be of 2 basic types: linear or non-linear. A linear process can be described by a linear homogenous differential equation, which for simplicity in algebraic manipulation is expressed by transfer function. A transfer function is the ratio of the output to the input of a process in a transformed domain, such as S-domain, with a setting of all initial conditions to zero. Processes having non-linear dynamics are expressed by non-linear differential equations or describing functions. A describing function is the ratio of the Fourier transform of the fundamental frequency component of the output to the Fourier transform of the input of the process with all initial conditions in the transforms to be zero. For designing a controller, some specific control criteria such as *gain margin*, *phase margin* [5] or allowable *percentage overshoot* [9] etc. need to be specified along with the process dynamics. Thus when the process dynamics is not clearly known, controller design by conventional approaches is not feasible.

## 4.2 Advantages of Fuzzy Control

In absence of the dynamics of the plant, we need to rely on the rule-based model of the controller. A rule-based controller includes a set of IF-THEN production rules that cause a change in the input of the process based on the error signal. The error in the present context is defined as the difference of set-point and process value (vide Fig. 4.1). One typical production rule (PR) for a temperature control system, for example, is given below:

*PR1: IF error is 20° Celcius THEN keep the heater on for 10 seconds.*

Since the error can attain many possible values, thousands of rules are needed to describe the complete span of error. Thus for only one input variable, the number of rules counts to 1000 say. When there are 2 variables, say the error and rate of change of error, the number of rules may be as high as  $10^6$ . A large number of rules calls for a large matching time of the error status with the antecedent parts of the rules, until one among a set of rules ready for firing is identified. One alternative way to handle this problem is to employ a minimal number of fuzzy production rules whose antecedent clauses can partially match with the available error status (or rate of error status) of the control loop. As an example, the representative control laws may take the following form:

*PR1: IF error is POSITIVE-HIGH  
THEN keep heater on for LONGER duration.*

*PR2: IF error is POSITIVE-LOW  
THEN keep heater on for SMALLER duration.*

*PR3: IF error is NEGATIVE-HIGH  
THEN keep the heater off for LONGER duration.*

*PR4: IF the error is NEGATIVE-LOW  
THEN keep the heater off for SMALLER duration.*

In the above rules, error is a linguistic variable and POSITIVE-HIGH, POSITIVE-LOW, NEGATIVE-HIGH and NEGATIVE-LOW are subsets of the universe ERROR. Duration is another linguistic variable, and SMALLER and LONGER are two fuzzy sets under the universe DURATION. It is indeed clear that such set of fuzzy production rules drastically reduce the necessity of a large number of crisp rules specifically in the application of controller design.

## 4.3 Typical Fuzzy Control Systems

Two typical fuzzy control systems are prevalent in the current literature on fuzzy logic. These are popularly known as Mamdani type and Takagi-Sugeno (T-S)

type. Mamdani type fuzzy systems employ fuzzy sets in the consequent part of the rules. The generic format of the production rules in the Mamdani system is illustrated below:

### *Rule Format in Mamdani System*

*PRj: IF  $x_1$  is  $A1^j$  and  $x_2$  is  $A2^j$  and .....and  $x_n$  is  $An^j$   
THEN  $y$  is  $Bj$ ,  $j = 1, 2, \dots, M$*

where

$x_i$  for  $i=1, 2, \dots, n$  are linguistic input variables;

$Ai^j$  for  $i=1, 2, \dots, n$  are input fuzzy sets;

$y$  is the linguistic output variable;

$Bj$  is the output fuzzy set;

$M$  is the number of fuzzy rules.

Takagi-Sugeno fuzzy systems, however, employ function of the input fuzzy linguistic variables as the consequent of the rules. The j-th production rule PRj in T-S system is presented below.

### *Rule Format in T-S System*

*PRj: IF  $x_1$  is  $A1^j$  and  $x_2$  is  $A2^j$  and .....and  $x_n$  is  $An^j$   
THEN  $y_j = f_j(x_1, x_2, \dots, x_n)$ , for  $j = 1, 2, \dots, M$*

where  $f_j(\cdot)$  is a linear or non-linear function of input variables. The linear function of input variables is generally chosen, i.e.,

$$\begin{aligned} & PRj: \text{IF } x_1 \text{ is } A1^j \text{ and } x_2 \text{ is } A2^j \text{ and .....and } x_n \text{ is } An^j \\ & \text{THEN } y_j = \sum_{i=0}^n p_{ji} x_i, \text{ for } j = 1, 2, \dots, M \end{aligned} \quad (4.1)$$

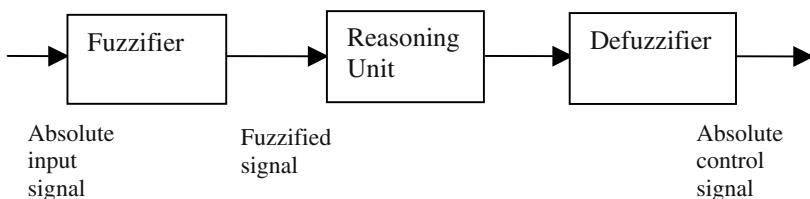
where  $x_0 \equiv 1$  and  $p_{ji}$  are the weights of the respective inputs  $x_i$ .

The T-S system defined by (4.1) is called linear T-S fuzzy system.

One point should be addressed before closing this section. This is why we add one superscript  $j$  in the fuzzy sets for the  $j$ -th rule. This in fact is needed to represent variations in the fuzzy sets. For example, suppose  $A1^j = \text{HIGH}$ , whereas  $A1^k = \text{VERY-HIGH}$ , where both correspond to the same linguistic variable  $x_1$ , say.

## 4.4 Architecture of Typical Fuzzy Control Systems

A typical fuzzy control system embodies a fuzzifier, a reasoning module and a defuzzifier. The fuzzifier transforms the absolute signals into fuzzy memberships by using intuitively designed membership functions. The reasoning module generates fuzzy inferences by GMP, and the defuzzifier module converts the fuzzy signals into appropriate signal levels suitable for the domain specific application. The defuzzified output is treated as the control signal and transferred to the process for controlling its output parameter. Fig. 4.2 describes the basic steps involved in the generation of control signal by a fuzzy controller.



**Fig. 4.2:** Basic steps involved in the generation fuzzy control signal.

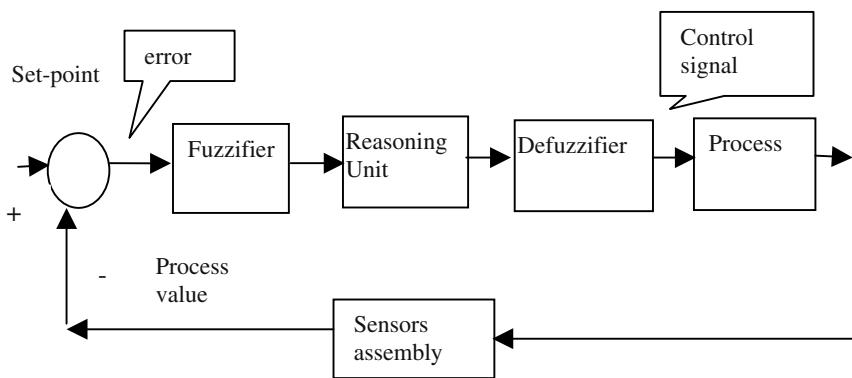
The absolute input signal in a closed loop fuzzy control system (vide Fig. 4.3(a)) could be error or function of error in general. There are, however, occasions when instead of error some parameters of interest are determined from the plant response for subsequent generation of control signals from those measurements. Consider for example power control in a nuclear power plant. Here, 2 parameters namely *power fraction* and *period* are extracted from the response of the plant. These parameters are used subsequently for instantiation of the linguistic variables in the antecedent parts of fuzzy production rules. The consequent part of the rules includes signals for power control. A schematic diagram of the fuzzy control system of the latter type is presented in Fig. 4.3(b).

## 4.5 Reasoning in Mamdani Type Fuzzy Control Systems

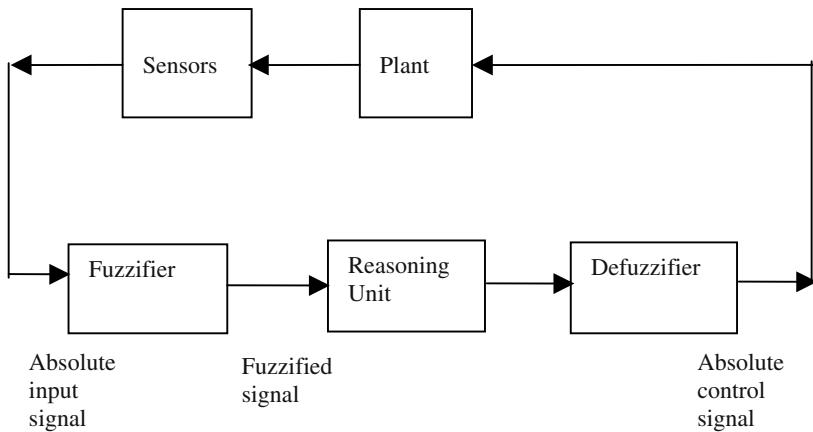
It has been discussed earlier that the production rules employed in Mamdani type reasoning system have fuzzy sets in both the antecedent and the consequent parts. For convenience we reproduce the j-th rule PR<sub>j</sub> below:

$$\begin{aligned} PR_j: & \text{ IF } x_1 \text{ is } A_1^j \text{ and } x_2 \text{ is } A_2^j \text{ and } \dots \dots \text{ and } x_n \text{ is } A_n^j \\ & \text{ THEN } y \text{ is } B_j, \quad j = 1, 2, \dots, M \end{aligned}$$

where the notations have been explained in section 4.3 earlier.



**Fig. 4.3 (a):** Generation of control signal from error.



**Fig. 4.3(b):** Generation of control signal from the measurements of the plant response.

Let

$\mu_{A_i^j}(x_i)$  be the membership of  $x_i$  to belong to the fuzzy set  $A_i^j$ , for  $i=1,2,\dots,n$

and  $\mu_{B_j}(y)$  be the membership of  $y$  to belong to fuzzy set  $B_j$ .

Then the fuzzy implication relation between the antecedent and the consequent part of the  $j$ -th rule is given by

$$R_j(x_1, x_2, \dots, x_n) = \{\mu_{R_j}(x_1, x_2, \dots, x_n; y) / (x_1, x_2, \dots, x_n, y)\} \quad (4.2)$$

where

$$\mu_{R_j}(x_1, x_2, \dots, x_n; y) = f_j(\mu_{\text{Antecedent}}(x_1, x_2, \dots, x_n), \mu_{B_j}(y)), \quad (4.3)$$

$f_j$  is any implication function,

$$\mu_{\text{Antecedent}}(x_1, x_2, \dots, x_n) = \mu_{A_1}^j(x_1) \wedge \mu_{A_2}^j(x_2) \wedge \dots \wedge \mu_{A_n}^j(x_n), \quad (4.4)$$

and  $\wedge$  in the last expression denotes the  $\wedge$  norm.

When the observed membership distribution of the antecedent clauses  $\mu_{A_1}/(x_1)$ ,  $\mu_{A_2}/(x_2)$ ,  $\dots$ ,  $\mu_{A_n}/(x_n)$ , are available, we can easily evaluate the membership distribution of the consequent clause  $\mu_{B_j}'(y)$  by

$$\begin{aligned} & \mu_{B_j}'(y) \\ &= \max_{x_1 \in X_1, x_2 \in X_2, \dots, x_n \in X_n} [\min \{(\mu_{A_1}'(x_1) \wedge \mu_{A_2}'(x_2) \wedge \dots \wedge \mu_{A_n}'(x_n)), \mu_{R_j}(x_1, x_2, \dots, x_n; y)\}] \end{aligned} \quad (4.5)$$

where  $X_1, X_2, \dots, X_n$  are  $n$  fuzzy universes.

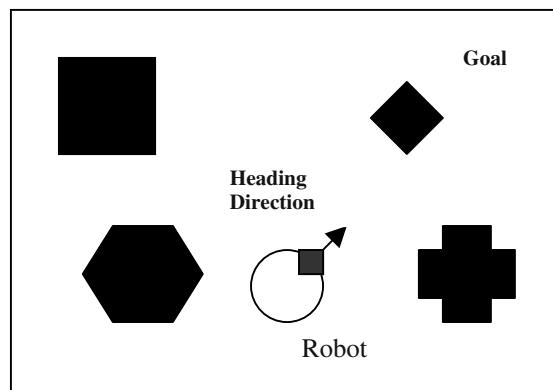
Finally, taking aggregation of  $\mu_{B_j}'(y)$  for  $j=1$  to  $M$ , we finally find

$$\mu_B'(y) = \max_{j=1}^M \{\mu_{B_j}'(y)\}. \quad (4.6)$$

**Example 4.1:** Consider the path-planning problem of a mobile robot amidst obstacles. The initial and the goal position of the robot are prescribed. The robot moves incrementally, and plans and executes its next step of motion in its current position until the robot reaches the goal position. In path-planning problem the *spacing* between the robot and the obstacles is an important parameter. In fact *spacing* and *heading direction* of the robot are governing issues in determining the *speed* of the robot. A simple fuzzy rule that relates robot's *speed* setting with *spacing* and *heading direction* is presented below:

*Rule 1: IF (spacing with obstacles is ACCEPTABLE), and  
(heading-direction is TOWARDS-GOAL)  
THEN (speed is HIGH).*

There are other rules as well, but we do not mention them here, as rule 1 is adequate to illustrate the desired concept.



**Fig. 4.4:** A circular shaped mobile robot moves through its workspace to the Goal position amidst several obstacles (denoted by dark bodies).

Here, spacing, heading-direction and speed are fuzzy linguistic variables in 3 fuzzy universes SPACING, HEADING-DIRECTION and SPEED respectively, and ACCEPTABLE, TOWARDS-GOAL and HIGH are fuzzy subsets of the appropriate universes. Suppose, membership functions for spacing and heading-directions are given (Fig. 4.5(a) and (b)), and the robot measures its heading direction and determines the membership of the heading direction towards the goal. It also measures the distance of the nearest obstacle (spacing) to avoid collision with the obstacle during its next step of movement.

Now, suppose that the sensory readings of spacing and heading-direction are given as 10 cm and 20° respectively and we are interested to know the speed of the robot in the next step of its movement. For this computation we use expression (4.5). Here,

$$\begin{aligned} & \mu_{\text{HIGH}}^{\prime}(\text{speed}) \\ &= \text{Max} [ \text{Min} \{ (\mu_{\text{ACCEPTABLE}}^{\prime}(\text{sp}) \text{ t } \mu_{\text{TOWARDS-GOAL}}^{\prime}(\text{hd})) , \mu_{Rj}(\text{sp}, \text{hd}; \text{speed}) \} ] \\ & \text{sp} \in \text{SPACING}, \text{hd} \in \text{HEADING-DIRECTION} \quad [\text{by (4.5)}] \end{aligned} \quad (4.7)$$

where sp denotes spacing and hd denotes heading-direction.

From the given membership curves 4.5(a) and (b), we now find:

$$[\mu_{\text{ACCEPTABLE}}^{\prime}(\text{sp})]_{\text{sp}=10 \text{ cm}} = 1.0, \text{ and}$$

$$[\mu_{\text{TOWARDS-GOAL}}^{\prime}(\text{hd})]_{\text{hd}=20^\circ} = 0.6.$$

Further,

$$\mu_{Rj}(sp, hd; speed)$$

$$= \text{Min}[\text{Min}\{\mu_{\text{ACCEPTABLE}}(sp), \mu_{\text{TOWARDS-GOAL}}(hd)\}, \mu_{\text{HIGH}}(\text{speed})] \quad (4.8)$$

Assuming 3-points membership distributions, suppose we have:

$$\mu_{\text{ACCEPTABLE}}(sp) = \{0.6/5 \text{ cm}, 1.0/10 \text{ cm}, 0.7/15 \text{ cm}\} \text{ and}$$

$$\mu_{\text{TOWARDS-GOAL}}(hd) = \{0.5/10^\circ, 0.6/20^\circ, 1.0/30^\circ\}.$$

Further, suppose that we are also given the membership distribution profile of speed as follows:

$$\mu_{\text{HIGH}}(\text{speed}) = \{0.7/10 \text{ m/s}, 0.8/20 \text{ m/s}, 0.9/30 \text{ m/s}\}.$$

Consequently,  $\mu_{Rj}(sp, hd; speed)$ , which should have 27 terms, has the largest term equal to  $1.0/(10\text{cm}, 30^\circ, 30 \text{ m/s})$ . Taking Min as the t-norm, we finally have

$$\mu_{\text{HIGH}}'(speed)$$

$$= \text{Max}[\text{Min}\{(\mu_{\text{ACCEPTABLE}}'(sp) \text{ Min } \mu_{\text{TOWARDS-GOAL}}'(hd)), \mu_{Rj}(sp, hd; speed)\}]$$

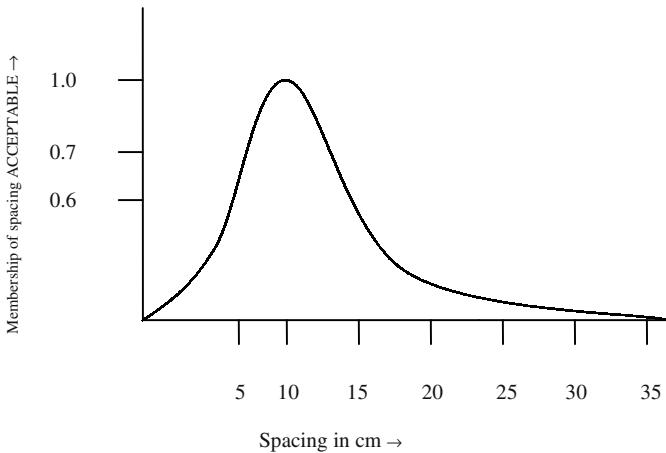
$$= \text{Max}[\text{Min}\{(\text{Min}(1.0, 0.6), \mu_{Rj}(sp, hd; speed))\}]$$

$$= \text{Max}[\text{Min}\{0.6, \mu_{Rj}(sp, hd; speed)\}]$$

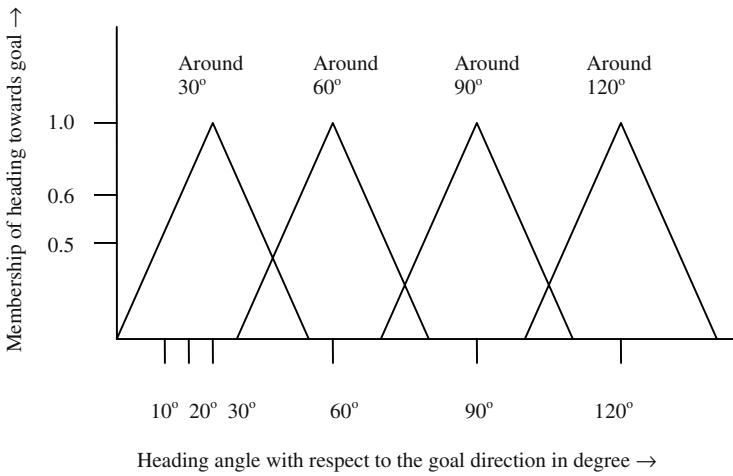
$$= \text{Min}[0.6, \text{Max}\{\mu_{Rj}(sp, hd; speed)\}]$$

$$= \text{Min}[0.6, 1.0] = 0.6.$$

Since the result corresponds to the term  $1.0/(10\text{cm}, 30^\circ, 30 \text{ m/s})$  of  $\mu_{Rj}(sp, hd; speed)$ ,  $\mu_{\text{HIGH}}'(speed) = 0.6$  denotes the membership of speed=30 m/s to be 0.6.



**Fig. 4.5 (a):** Membership curve  $\mu_{\text{ACCEPTABLE}}(\text{spacing})$ .



**Fig. 4.5(b):** Membership curve:  $\mu_{\text{TOWARDS-GOAL}}$  (heading-direction).

## 4.6 Reasoning in T-S Fuzzy Control System

The model described by equation (4.1) is a simple form of Takagi-Sugeno system model. A more general form of T-S model composed of N production rules, is given by

$$\begin{aligned} PR_i: & \text{ IF } x_1(k) \text{ is } M_1^i \text{ and } x_2(k) \text{ is } M_2^i \text{ and....and } x_n(k) \text{ is } M_n^i \\ & \text{ THEN } x_i(k+1) = A_i x(k) + B_i u(k), \quad i = 1, 2, \dots, N, \end{aligned} \quad (4.9)$$

where

$k$ = discrete time index;

$x_j$ =j-th linguistic variable or state;

$M_j^i$  = a fuzzy term of  $M_j$  selected for rule  $i$ ;

$M_j$  = a fuzzy term set of  $x_j$ ;

$x(k) = [x_1(k) \ x_2(k) \dots \ x_n(k)]^T \in R^n$  is a n-dimensional state vector in the real space;

$u(k) = [u_1(k) \ u_2(k) \dots \ u_n(k)]^T \in R^n$  is the input vector;

$A_i \in R^{n \times n}$  and  $B_i \in R^{n \times m}$ .

Given a current state vector  $x(k)$  and a input vector  $u(k)$ , the T-S model infers  $x(k+1)$  as follows:

$$x(k+1) = \sum_{i=1}^N h_i(k) [A_i x(k) + B_i u(k)] \quad (4.10)$$

where

$$h_i(k) = w_i(k) / \sum_{j=1}^N w_j(k) \quad (4.11)$$

and  $w_i(k) = M_1^i(x_1(k)) \cdot M_2^i(x_2(k)) \dots \cdot M_n^i(x_n(k))$ .

$$= \prod_{j=1}^n M_j^i(x_j(k)). \quad (4.12)$$

For an autonomous system having  $u(k)=0$ , (4.10) can be written as

$$x(k+1) = \sum_{i=1}^N h_i(k) A_i x(k) \quad (4.13)$$

**Example 4.2:** Let us consider the following 2 production rules (PR) of T-S format.

*PR1: IF  $x_1$  is LARGE and  $x_2$  is MODERATE  
THEN  $x_1(k+1) = A_1 x(k)$ .*

*PR2: IF  $x_1$  is SMALL and  $x_2$  is LARGE  
THEN  $x_2(k+1) = A_2 x(k)$*

where  $x(k) = [x_1(k) \ x_2(k)]^T$ .

$$\text{Also given, } M^1_{\text{LARGE}}(x_1) = 0.3, \quad M^1_{\text{MODERATE}}(x_2) = 0.4,$$

$$M^2_{\text{SMALL}}(x_1) = 0.7, \quad \text{and} \quad M^2_{\text{LARGE}}(x_2) = 0.8.$$

Suppose, the measured values of  $x_1$  and  $x_2$  at  $k=1$  are

$$x_1(1) = 40 \text{ units and } x_2(1) = 20 \text{ units,}$$

$$A_1 = \begin{pmatrix} 0.5 & 0.4 \\ 2.0 & 1.0 \end{pmatrix}$$

$$\text{and } A_2 = \begin{pmatrix} 0.6 & 0.8 \\ 1.0 & 2.0 \end{pmatrix}.$$

we want to compute  $x(k+1)$  at  $k=1$ .

$$\text{Now, } x(k+1) = \sum_{i=1}^2 h_i(k) A_i x(k)$$

where

$$w_1(1) = M^1_{\text{LARGE}}(x_1(1)) \cdot M^1_{\text{MODERATE}}(x_2(1))$$

$$= 0.3 \times 0.4$$

$$= 0.12,$$

$$w_2(1) = M^2_{\text{SMALL}}(x_1(1)) \cdot M^2_{\text{LARGE}}(x_2(1))$$

$$= 0.7 \times 0.8$$

$$= 0.56,$$

$$h_1(1) = w_1(1) / [w_1(1) + w_2(1)]$$

$$= 0.12 / [0.12 + 0.56]$$

$$= 12/68,$$

$$h_2(1) = w_2(1) / [w_1(1) + w_2(1)]$$

$$= 0.56 / [0.12 + 0.56]$$

$$= 56/68.$$

Therefore,

$$x(2) = \sum_{i=1}^2 h_i(1) A_i x(1)$$

$$= h_1(1) A_1 x(1) + h_2(1) A_2 x(1)$$

$$= (12/68) \begin{pmatrix} 0.5 & 0.4 \\ 2.0 & 1.0 \end{pmatrix} \begin{pmatrix} 40 \\ 20 \end{pmatrix} + (56/68) \begin{pmatrix} 0.6 & 0.8 \\ 1.0 & 2.0 \end{pmatrix} \begin{pmatrix} 40 \\ 20 \end{pmatrix}$$

$$= [37.88 \quad 83.52]^T$$

## 4.7 Stability Analysis of Dynamic Systems Using Lyapunov Energy Functions

There are many approaches to stability analysis of dynamic systems. A list of the classical techniques of stability analysis is available in any standard literature on systems theory and control [5]. In this section, stability analysis of dynamic systems using Lyapunov energy functions will be undertaken. A Lyapunov function is  $V(x_1, x_2, \dots, x_n)$  of  $n$  variables is called an energy function that satisfies the following conditions:

1.  $V(0, 0, \dots, 0) = 0$ ,
2.  $V(x_1, x_2, \dots, x_n) > 0$  for  $x_i > 0$  for all  $i$ ,
3. As  $x_i$  approaches infinity,  $V$  should approach infinity.

For stability analysis of a dynamic system, we should select the Lyapunov energy function  $V$  in a manner, so that the dynamics can be mapped on the energy surface, and asymptotic stability of the system can be ascertained by noting a negative change in  $\Delta V$  for a state transition of the system from  $X^T(k)$  to  $x^T(k+1)$ . Suppose, given a discrete system

$$x(k+1) = A x(k), \quad (4.14)$$

and we want to analyze the stability of the system.

Let

$$V(x_1, x_2, \dots, x_n) = x^T(k) P x(k), \quad (4.15)$$

where  $P$  is a positive definite matrix. To illustrate how the above  $V$  satisfy the characteristics of the Lyapunov energy function, suppose we take  $P = I$ , the identity matrix. Then

$$V = x^2(k) = x_1^2(k) + x_2^2(k) + \dots + x_n^2(k), \quad (4.16)$$

which satisfies all the three characteristics of a Lyapunov function.

$$\text{Now, } \Delta V = \Delta[x^T(k) P x(k)]$$

$$\begin{aligned} &= x^T(k+1) P x(k+1) - x^T(k) P x(k) \\ &= x^T(k) A^T P A x(k) - x^T(k) P x(k) \quad [\text{by (4.14)}] \\ &= x^T(k)[A^T P A - P] x(k) \\ &= -x^T(k) Q x(k), \text{ say} \end{aligned} \quad (4.17)$$

where

$$A^T P A - P = -Q \quad (4.18)$$

The system  $x(k+1) = Ax(k)$  is said to be *asymptotically stable if  $Q$  is positive definite*.

## 4.8 Stability Analysis of T-S Fuzzy Systems

Tanaka and Sugeno [13] suggested an important criterion for the analysis of stability of the T-S fuzzy system.

**Property 1:** The equilibrium state  $x=0$  of the fuzzy system (4.13) is globally asymptotically stable if there exists a common positive definite matrix  $P$  such that

$$A_i^T P A_i - P < 0, \text{ for all } i=1,2,\dots,N.$$

First, let us consider  $N=2$  for simplicity. Then for stability analysis of the system (4.13) we break it into 2 sub-systems.

$$x(k+1) = x_1(k+1) + x_2(k+1) \quad (4.19)$$

$$\text{where } x_1(k+1) = h_1(k) A_1 x(k) \quad (4.20)$$

$$= A_1 x_1(k), \text{say}$$

$$\text{and } x_2(k+1) = h_2(k) A_2 x(k) \quad (4.21)$$

$$= A_2 x_2(k), \text{say}$$

In general,

$$x_i(k+1) = A_i x_i(k) \quad \text{for } i=1,2,\dots,N \quad (4.22)$$

Narendra and Balakrishnan [8] suggested a systematic way of finding the common  $P$  matrix for  $N$ -simultaneous continuous-time linear system satisfying pair-wise commutative characteristics. The results are extended below for discrete-time systems following [4].

**Property 2:** Suppose matrices  $A_1$  and  $A_2$  are Schur and commutative, i.e.,

$$A_1 A_2 = A_2 A_1 \quad (4.23)$$

Now, for each discrete system given by (4.20) and (4.21) we employ the following Lyapunov criteria for stability analysis.

$$A_1^T P_1 A_1 - P_1 = -Q \quad (4.24)$$

$$A_2^T P_2 A_2 - P_2 = -P_1 \quad (4.25)$$

where  $Q > 0$  and  $P_1$  and  $P_2$  are the unique positive definite solutions of the above equations respectively. Then we always have

$$A_i^T P_2 A_i - P_2 < 0 \text{ for } i=1,2,\dots$$

**Proof:** Details of the proof is available in [4]. We here give an outline, so that interested readers can prove it themselves with the given outline.

- ◆ Substitute the value of  $P_1$  from (4.25) in (4.24).
- ◆ Replace  $(A_2 A_1)^T = A_1^T A_2^T$  and  $(A_1 A_2)^T = A_2^T A_1^T$  in the resulting expression, so that the result can finally be expressed as

$$-Q = A_2^T \psi_1 A_2 - \psi_1 \quad (4.26)$$

$$\text{where } \psi_1 = -A_1^T P_2 A_1 + P_2 \quad (4.27)$$

Since  $Q > 0$  and  $A_2$  is Schur, the solution  $\psi_1 > 0$  is unique.

- ◆ Assuming the Lyapunov function

$$V(x(k)) = x^T(k) P_2 x(k)$$

for both systems, we now compute  $\Delta V$  for  $A_2$  system as follows:

$$\begin{aligned} \Delta V &= x^T(k) (A_2^T P_2 A_2 - P_2) x(k) \\ &= -x^T(k) P_1 x(k) < 0 \text{ for all } x(k) \end{aligned} \quad (4.28)$$

since  $P_1 > 0$  in (4.25)

- ◆ Further for the  $A_1$  system,

$$\begin{aligned} \Delta V &= x^T(k) (A_1^T P_2 A_1 - P_2) x(k) \\ &= -x^T(k) \psi_1(k) x(k) < 0 \text{ for } x(k) \neq 0 \end{aligned} \quad (4.29)$$

since  $\psi_1 > 0$

- ◆ From expressions (4.28) and (4.29) we find that

$$A_i^T P_2 A_i - P_2 < 0 \text{ for } i=1, 2$$

and  $P_2$  thus can be used as common  $P$  matrix of the T-S fuzzy model.

For  $N$ -plant rules, we can easily extend the above concept as summarized in property 4.3.

**Property 4.3:** Suppose  $A_i$  for  $i=1,2,\dots,N$  is Schur, i.e.,

$$A_j A_{j+1} = A_{j+1} A_j \text{ for } j=1,2,\dots,N-1$$

Considering  $N$  Lyapunov equations:

$$\left. \begin{array}{l} A_1^T P_1 A_1 - P_1 = -Q \\ A_2 P_2 A_2 - P_2 = -P_1 \\ \vdots \\ \vdots \\ A_N^T P_N A_N - P_N = -P_{N-1} \end{array} \right\} \quad (4.30)$$

where  $Q > 0$  and  $P_i$  for  $i=1,2,\dots,N$  is the unique positive definite symmetric solution for each equation. Then we always have

$$A_i^T P_N A_i - P_N < 0 \quad \text{for } i=1,2,\dots,N \quad (4.31)$$

Proof of the above property is similar to previous proof.

**Example 4.3:** For 3 plant rules and 2 state variables, let the corresponding  $A_i$ 's be

$$A_1 = \begin{pmatrix} 1.0 & 0.2 \\ -0.3 & 0.5 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} 0.5 & 0.12 \\ -0.18 & 0.2 \end{pmatrix} \quad \text{and}$$

$$A_3 = \begin{pmatrix} 0.8 & 0.16 \\ -0.24 & 0.4 \end{pmatrix}$$

It can easily be verified that

$$A_1 A_2 = A_2 A_1 = \begin{pmatrix} 0.464 & 0.16 \\ -0.24 & 0.064 \end{pmatrix}$$

and

$$A_2 A_3 = A_3 A_2 = \begin{pmatrix} 0.3712 & 0.128 \\ -0.192 & 0.051 \end{pmatrix}$$

With  $Q=I$ , we can show that

$$A_1^T P_1 A_1 - P_1 = -Q \text{ yields } P_1 = \begin{pmatrix} 7.30 & 2.00 \\ 2.00 & 2.25 \end{pmatrix}$$

$$A_2^T P_2 A_2 - P_2 = -P_1 \text{ yields } P_2 = \begin{pmatrix} 9.21 & 2.67 \\ 2.67 & 2.62 \end{pmatrix}$$

$$A_3^T P_3 A_3 - P_3 = -P_2 \text{ yields } P_3 = \begin{pmatrix} 19.374 & 6.543 \\ 6.543 & 4.711 \end{pmatrix}$$

Thus  $P_3$  which is symmetric and positive definite is the common matrix that should satisfy

$$A_i^T P_3 A_i . P_3 < 0 \text{ for } i=1, 2, 3.$$

In fact a computation of  $A_i^T P_3 A_i - P_3$  for  $i=1$  to  $3$  can be shown to satisfy the inequality (4.31). Thus a common  $P$  matrix which guarantees the stability of T-S fuzzy model with  $N$  plant rules can be found systematically by using property 4.3.

## 4.9 Application in Power Control of a Nuclear Reactor

In late 1980's a group of researchers led by John. A. Bernerd in MIT Nuclear Reactor laboratory developed an autonomous scheme for power control in a nuclear reactor. In this section, we briefly outline their scheme below.

**Process:** The power output of a nuclear reactor greatly depends on the reactivity, which in turn may be considered as fractional change in the neutron population. A reactivity of zero implies that the reactor power is constant. Adjustment in power output is accomplished by positioning a neutron absorbing

control rod inside the thermonuclear pile. To increase the power output, the operator has to withdraw the control rod from the thermonuclear pile.

Under this circumstance, the reactivity is increased. On the other hand, when the power level approaches the desired value, the control rod is inserted gradually into the thermonuclear pile in order to reduce the reactivity to zero. The process is a complicated one because the reactor dynamics is non-linear and there are power-dependent feedback effects.

### Plant I/O parameters

The parameters used in designing the rules for power control are presented below:

**Reactor Period/Period:** It is defined as power level divided by the rate of change of power. Thus a period of infinity denotes steady state, while a period of small positive number represents a rapid change in power.

The period is one *universe of discourse* in the present context. The subsets of the universe are denoted by ‘too short’, ‘short’, ‘long’ etc.

**Rod Height:** This denotes height of the rod outside the thermonuclear pile. The more is the rod height or rod withdrawal, the higher is the reactivity and consequently higher is the power output.

**Power Fraction:** It denotes the fraction of the generated power delivered to the load. The power fraction is a measure of the generated power by the nuclear reactor.

**Degree of fulfillment of a clause:** Since the measurement/data used in firing a rule may not always conform to the clauses in the if-part of the rules, there is a need for partial matching. The degree to which a measurement/data conforms to the clauses is called the *degree of fulfillment (DOF)*.

For instance, suppose a measurement of period =80 second may be called *short* with a  $DOF=2-0.02\times80 = 0.04$ . In fact, DOF is same as membership functions.

**Signal:** This is the output parameter of the fuzzy controller employed in the nuclear power plant referred to above. The parameter signal ranges in [-1,+1]. A positive unity signal indicates that the rod has to be withdrawn from the thermonuclear pile at its maximum speed. A negative unity signal on the other hand represents insertion of the rod into the pile at maximum speed. A fractional value of +0.57, for instance, denotes that the rod has to be withdrawn from the thermonuclear pile at 57% of its maximum speed.

**Table 4.1:** Fuzzy subsets and DOFs of linguistic variables

<b>Subset</b>	<b>Range</b>	<b>DOF</b>
<b>Period(T) in sec.</b>		
Too Short	$0 < T < 50$ $50 < T < 100$	1.0 2.0 - 0.02T
Short	$50 < T < 100$ $100 < T < 120$	1.0 6.0-0.05T
Positive	$100 < T < 120$ $120 < T < 500$	1.0 1.814-0.002627
Long	$500 < T < 1000$	2.0 - 0.002T
Too Long	$500 < T < \infty$ $-1000 < T < -\infty$ $-500 < T < -1000$	1.0 0.8 -0.8-0.0016T
Negative	$0 < T < -\infty$	1.0
<b>Rod Withdrawal (<math>\Delta H</math>) in inch</b>		
Excessive	$4.0 < \Delta H$ $2.0 < \Delta H < 4.0$	1.0 0.5( $\Delta H$ )-1.0
<b>Power Fraction (PF)</b>		
Very Low	$0.00 < PF < 0.40$	1.0
Low	$0.40 < PF < 0.80$ $0.00 < PF < 0.40$	1.0 2.5×PF
High	$1.00 < PF$ $0.98 < PF < 1.00$	1.0 50×PF-49
<b>Rod Height (H) in inch</b>		
High	$10.0 < H$ $6.0 < H < 10.0$	1.0 0.25 × H - 1.5

**Linguistic Variables and DOF:** Table 4.1 presented above lists the fuzzy subsets and DOFs of the linguistic variables in these subsets. It is an illustrative list of fuzzy subsets and the DOFs. A complete list of the fuzzy subsets used for nuclear plant is available in [1-2]. A set of rules constructed to control the power output by regulating the rod position in the thermonuclear pile is also outlined below.

### Fuzzy Production Rules for Power Control

*Rule 1: IF period is SHORT and PF is VERY LOW  
THEN signal = -1*

*Rule 2: IF period is SHORT and PF is LOW  
THEN signal = -0.5*

*Rule 3: IF period is POSITIVE and PF is VERY LOW  
THEN signal = 0.0*

*Rule 4: IF period is LONG and PF is VERY LOW  
THEN signal = 1.0*

*Rule 5: IF period is TOO LONG and PF is VER LOW  
THEN signal = 1.0*

*Rule 6: IF PERIOD is NEGATIVE and PF is HIGH  
THEN signal = -1.0*

*Rule 7: IF (rod position is HIGH OR rod-withdrawal is EXCESSIVE)  
and PF is LOW  
THEN signal is -1.0*

The above rules are not exhaustive. A complete list of 20 rules used in designing the MIT nuclear power plant is given in [1].

Unlike pure Mamdani or T-S systems, the control law for power control in the proposed nuclear plant is given by

$$\text{Control} = \frac{\sum_i (\text{action})_i \times (\text{DOF})_i}{\sum_i (\text{DOF})_i} \quad (4.32)$$

As an example, suppose the following measurements are available, and we are interested to compute the control signal based on these measurements.

Measurements supplied

$$\begin{aligned} \text{PF} &= 0.25 \\ \text{T} &= 600 \text{ sec.} \\ \text{H} &= 8.25 \text{ inch} \end{aligned}$$

$$\Delta H = 2.88 \text{ inch}$$

A look at Table 4.1 reveals that here PF falls in the category LOW or VERY LOW; period (T) falls in category LONG or TOO LONG; rod height (H) and rod withdrawal ( $\Delta H$ ) fall in the fuzzy sets HIGH and EXCESSIVE respectively. A summary of the active fuzzy sets for the given measurements is presented below in Table 4.2.

**Table 4.2:** Active fuzzy sets for the given measurements

Parameter	Active Fuzzy Set
PF	LOW/ VERY LOW
T	LONG/ TOO LONG
H	HIGH
$\Delta H$	EXCESSIVE

The measurements given above provide an instantiation space of the fuzzy sets present at the antecedent part of rules 4, 5 and 7, thereby enabling the said rules to fire. Consequently, the control signal for rod insertion/ withdrawal is generated for the given rule. DOF computations for the above three rules are needed for evaluation of the control signal. As an example, the DOF computation for rule 4 is presented below.

$$\begin{aligned}
 \text{DOF} |_{\text{Rule4}} &= \text{Min} \{ \text{DOF}_{\text{LONG}} (\text{T}), \text{Max} \{ \text{DOF}_{\text{VERY LOW}} (\text{PF}), \text{DOF}_{\text{LOW}} (\text{PF}) \} \\
 &= \text{Min} \{ 2 - 0.002 \times 600, \text{Max} \{ 1, 2.5 \times 0.25 \} \} \\
 &= \text{Min} \{ 0.80, \text{Max} \{ 1, 0.625 \} \} \\
 &= \text{Min} \{ 0.80, 1 \} \\
 &= 0.80.
 \end{aligned}$$

Similarly we compute  $\text{DOF} |_{\text{Rule 5}} = 1.00$  and  $\text{DOF} |_{\text{Rule 7}} = 0.56$ .

Now, by (4.32), we have

$$\begin{aligned}
 \text{Control} &= \frac{\text{Signal}_4 \times \text{DOF} |_{\text{Rule 4}} + \text{Signal}_5 \times \text{DOF} |_{\text{Rule 5}} + \text{Signal}_7 \times \text{DOF} |_{\text{Rule 7}}}{\text{DOF} |_{\text{Rule4}} + \text{DOF} |_{\text{Rule5}} + \text{DOF} |_{\text{Rule7}}} \\
 &= \frac{1.00 \times 0.80 + 1.0 \times 1.00 + (-1.00) \times (0.56)}{0.80 + 1.00 + 0.56} \\
 &= 0.53,
 \end{aligned}$$

which means that the control rod will be withdrawn at 53% of its full speed.

## 4.10 Defuzzification Techniques

Suppose, a fuzzy control system employs the following control rule:

*Rule 1: IF the duty-cycle of the current is LARGE  
THEN the temperature of the heater is HIGH.*

Here duty-cycle (dc) and temperature (temp) are fuzzy variables, and LARGE and HIGH are fuzzy sets of the universes: DUTYCYCLE and TEMPERATURE respectively. Given a relation R (dc, temp) and the membership function  $\mu_{\text{LARGE}}(\text{dc})$ , we can easily compute the membership function  $\mu_{\text{HIGH}}(\text{temp})$  by using

$$\mu_{\text{HIGH}}'(\text{temp}) = \mu_{\text{LARGE}}'(\text{dc}) \circ R(\text{dc}, \text{temp}) \quad (4.33)$$

where  $\text{HIGH}' \approx \text{HIGH}$  and  $\text{LARGE}' \approx \text{LARGE}$

Just for the sake of convenience, let us assume that

$$\mu_{\text{HIGH}}'(\text{temp}) = \{0.6/80^\circ\text{C}, 0.8/90^\circ\text{C}, 0.7/100^\circ\text{C}\}.$$

Then how can we determine the temperature of the heating coil? Here lies the importance of defuzzification. Different types of defuzzification techniques are available in the current literature on fuzzy sets [3]. The most popular among them is the center of mass (also called gravity) approach. By this approach, we can compute the temperature of the coil as follows.

Temperature of the coil

$$\begin{aligned} &= \frac{0.6 \times 80^\circ\text{C} + 0.8 \times 90^\circ\text{C} + 0.7 \times 100^\circ\text{C}}{0.6 + 0.8 + 0.7} \\ &= \frac{48^\circ + 72^\circ + 70^\circ}{2.1} \\ &= \frac{190^\circ}{2.1} \approx 95^\circ\text{C} \end{aligned}$$

Formally, if a fuzzy control signal is given by  $\{\mu_A(u) | u\}$  then the defuzzified control signal may be evaluated by

$$\text{Control} = \frac{\sum_{u \in U} \{\mu_A(u) \times u\}}{\sum_{u \in U} \{\mu_A(u)\}} \quad (4.34)$$

where  $u$  is an element of the universe of discourse  $U$  and  $A$  is a fuzzy set under the universe  $U$ .

## 4.11 Conclusions

The chapter presented two different schemes of fuzzy control for complex plant dynamics, the mathematical form of which is not explicitly known. The former scheme proposed by Mamdani [7] is classical [6], [14] and is prevalent in the domain of fuzzy sets for around two decades [15]. The latter scheme proposed by Takagi and Sugeno [10-12] is novel and is more flexible in implementing the control law. The most important advantage of the latter scheme is that its foundation rests on the well-known state-space model of control system design. Consequently, the standard methods of stability analysis are equally applicable in the latter scheme. The chapter demonstrated applications of fuzzy control in robot motion planning and nuclear power plants.

## Exercise

- Given the domain of universe of a fuzzy set  $U = \{1, 2, \dots, 8\}$ , and

$$\mu_A(x) = \{0/1, 0.2/2, 0.5/5, 0.8/4, 1/5, 0.5/6, 0.2/7, 0/8\}$$

where  $A \subset U$ , determine the signal  $u$  by the center of gravity defuzzification principle.

[Answer:  $u = 4.53$  units]

- Given the measurements of a process to be

$$\begin{aligned}x_1(k) &= 4 \text{ units, and} \\x_2(k) &= 60 \text{ units.}\end{aligned}$$

The fuzzy production rules employed in the system are given below.

*Rule1: IF  $x_1$  is BIG and  $x_2$  is MED*

$$\text{THEN } u_1(k+1) = x_1(k) - 3x_2(k)$$

*Rule2: IF  $x_1$  is SMALL and  $x_2$  is BIG*

$$\text{THEN } u_2(k) = 4x_1(k) = 2x_2(k).$$

Here,  $u_1$  and  $u_2$  are fuzzy control signals, which are defuzzified and the final control signal  $u$  is applied to the process.

Given:  $\mu_{\text{BIG } 1}(x_1) = 0.3$ ,  $\mu_{\text{MED } 1}(x_2) = 0.7$ ,

$$\mu_{\text{SMALL } 2}(x_1) = 0.7, \mu_{\text{BIG } 2}(x_2) = 0.35$$

determine  $u(k+1)$  by T-S method.

$$[\text{Hints: } w_1(k) = \mu_{\text{BIG } 1}(x_1) \cdot \mu_{\text{MED } 1}(x_2)$$

$$= 0.3 \times 0.7 = 0.21$$

$$\begin{aligned} w_2(k) &= \mu_{\text{SMALL } 2}(x_1) \cdot \mu_{\text{BIG } 2}(x_2) \\ &= 0.7 \times 0.35 \\ &= 0.245 \end{aligned}$$

$$\begin{aligned} h_1(k) &= w_1 / (w_1 + w_2) \\ &= 0.21 / (0.21 + 0.245) \\ &= 0.46 \end{aligned}$$

$$\begin{aligned} h_2(k) &= w_2 / (w_1 + w_2) \\ &= 0.245 / (0.21 + 0.245) \\ &= 0.96 \end{aligned}$$

Therefore,  $u(k+1)$

$$= h_1(k) A_1 x(k) + h_2(k) A_2 x(k)$$

where

$$A_1 = [1 \quad -3],$$

$$A_2 = [4 \quad 2],$$

$$x(k) = \begin{pmatrix} x_1(k) \\ x_2(k) \end{pmatrix} = \begin{pmatrix} 4 \\ 60 \end{pmatrix}$$

Consequently,  $u(k+1) = 0.46(-176) + 0.96(136)$

$$= -80.96 + 130.56$$

$$= 49.6$$

Thus,  $u(k+1) = 49.6$  units.]

3. Show with an example that the nuclear power control discussed earlier is a special case of T-S model.

[**Hints:** Here, the i-th rules can be expressed as

IF  $x_1(k)$  is  $M_1^i$  and  $x_2(k)$  is  $M_2^i$  and .....and  $x_n(k)$  is  $M_n^i$

THEN  $x_i(k+1) = B_i u(k)$ ,

where  $B_i = [+1]$  or  $[-1]$  and  $u(k) = 1$ .

Thus, T-S model is applicable.]

4. Show that the center of gravity defuzzification technique is inclusive of T-S model for control signal generation.

[**Hint:** By definition,

$$u(k+1) = h_1(k) A_1 x(k) + h_2(k) A_2 x(k)$$

where

$$h_1(k) = w_1 / (w_1 + w_2)$$

$$h_2(k) = w_2 / (w_1 + w_2)$$

and  $w_i = \prod_{j=1}^n M_j^i(x_j(k))$  for  $i \in \{1, 2\}$ .

Thus,  $u(k+1)$  can be represented in the form:

$$\frac{\text{membership1} \times \text{signal 1} + \text{membership 2} \times \text{Signal 2}}{\text{membership1} + \text{membership 2}},$$

and hence T-S model includes center of gravity defuzzification.]

## References

- [1] Bernard, J.A., "Use of a Rule Based System for Process Control," *IEEE Control System Magazine*, October 1988.

- 
- [2] Bernard, J. A., Kwok, K. S., Ornedo, R. S., Lanning , D. D. and Hopps, J. H., "The Application of Digital Technology to the Control of Reactor Experiments," *Proc. of 6th Power plant Dynamics, Control and Testing Symp.*, Knoxville, TN, Apr 1986.
  - [3] Drainkov, D., Hellendoorn, H. and Reinfrank, M., *An Introduction to Fuzzy Control*, Springer –Verlag, Berlin, Germany, pp. 132-140, 1993.
  - [4] Joh, J., Chen, Y. - H. and Langari, R., " On the Stability Issues of linear Takagi- Sugeno Fuzzy Models," *IEEE Trans. on Fuzzy Systems*, Vol. 6, no.3, August 1998.
  - [5] Kuo, B. C., *Automatic Control Systems*, Prentice-Hall, NJ, 1975.
  - [6] Lee, C.C., "Fuzzy logic in Control systems: fuzzy controllers- part I and part II," *IEEE Trans. on Systems, Man and Cybernetics*, Vol. SMC-2092, pp. 404-435, 1990.
  - [7] Mamdani, E.H. and Assilian, S., " An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller," *Int. J. Man – Machine Studies*, vol.7, pp. 1-13, 1975.
  - [8] Narendra, K. S. and Balakrishnan, J., "A common Lyapunov function for stable LIT systems with commuting A-matrices," *IEEE Trans. on Automatic Control*, vol. 39, pp. 2569- 2571, Dec.1994.
  - [9] Ogata, K., *Modern Control Engineering*, Prentice-Hall, NJ, 1990.
  - [10] Ross, T. J., *Fuzzy Logic with Engineering Applications*, McGraw-Hill, NY, ch. 13, pp. 469-514, 1995.
  - [11] Takagi, T. and Sugeno, M., "Fuzzy identification of systems and its application to modeling and control," *IEEE Trans. on Systems, Man and Cybernetics*, vol. SMC-15, pp. 116-132, Jan.1985.
  - [12] Takagi, T. and Sugeno, M, "Stability analysis and design of fuzzy control systems," *Fuzzy Sets and Systems*, vol.45, pp.135-156, 1992.
  - [13] Tanaka, K., "Stability and stabilizability of fuzzy-neural-linear control systems," *IEEE Trans. on Fuzzy Systems*, vol. 3, pp. 438-447, Nov. 1995.
  - [14] Tag, K. L. and Mulholland, R. J., "Computing Fuzzy Logic with Classical Controller Design," *IEEE Trans. on Systems, Man and Cybernetics*, vol. SMC-17, no.6, pp. 1085-1087, 1987.
  - [15] Wang, L.-X., *A Course in Fuzzy Systems and Control*, Prentice-Hall International, Inc., NJ, pp. 270 –275, 1997.

# 5

# Fuzzy Pattern Recognition

*Classical models of pattern recognition partition a set of patterns into classes depending on the similarity in features of the patterns. When the distinctive features of the patterns are correctly identified, the classes can easily be distinguished in the feature space. Unfortunately, features in most pattern recognition problems are selected on an ad hoc basis, consequently causing the pattern classes to overlap, thereby leading to an ambiguity in object recognition. This chapter presents a well-known technique for fuzzy pattern recognition, capable of partitioning the patterns by soft boundaries. Thus a pattern may be classified into one or more classes with a certain degree of membership to belong to each class. The algorithm for fuzzy pattern recognition is numerically illustrated, and its application in object recognition from real time video frames is also presented.*

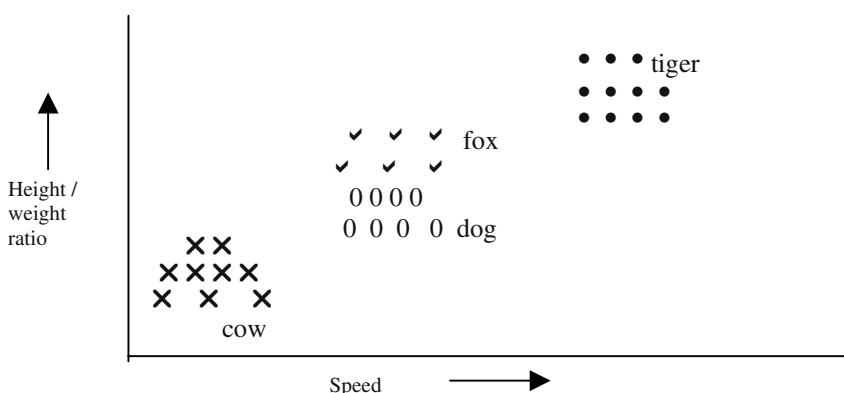
## 5.1 Introduction

A *pattern* is a physical or abstract structure of objects. It is distinguished from others by a collective set of attributes called *features*, which together represent the pattern. A triangular pattern, for instance, can be recognized by i) the length of its three sides, or ii) length of its any two sides and the angle between them, or iii) any two angles and one adjacent side common to the two angles. Thus, a

set of three features is required to minimally describe a triangular pattern. Bezdek [2-5] defined *pattern recognition* as a search for structure in data. Thus recognition of a pattern requires determining the nearest match of an unknown structure with a set of known structure of data.

Zimmermann considers pattern recognition as a problem of three main steps: i) pattern acquisition by transducers, ii) feature extraction by dimensionality reduction and iii) decision algorithms for mapping extracted features to pattern classes. The first step transforms physical world data to measured variables. The second step transforms a set of m measured variables to a set of n features, where  $n \leq m$ . This is popularly known as *dimensionality reduction* problem. The third step then maps the features to one of more classes.

Mapping an unknown pattern to one or more known pattern classes is referred to as **unsupervised classification/ learning**. In the last few chapters we discussed the scope of fuzzy logic in approximate reasoning. Another important aspect of fuzzy logic, we will discuss here, is its capability of **unsupervised classification** [1]. A question then obviously arises: why fuzzy logic is useful for such classification? To answer this let us take an example. Suppose, in an animal kingdom, we want to classify the animals by their speed and height to weight ratio. We can represent the classification process here by two dimensions. Let the X- and the Y-axes denote speed and height/ weight of the animals respectively. Now, if we plot the co-ordinates for each animal in the kingdom we will find that the animals of the same species (say, dogs) will occupy very close co-ordinates and thus form clusters (Fig. 5.1). Consequently animals of different species will form different clusters.



**Fig. 5.1:** Learning animals through classification.

But how do these clusters help us? They help us in classifying an unknown animal from its **feature space**, such as its speed and height to weight ratio, once the clusters are known. In other words, if one knows the measurements of speed and height to weight ratio of an animal, he can determine the class to which the animal should belong. So, classification has some justifications. In the next paragraph we will discuss the need for fuzzy classification.

The clustering process we discussed so far does not consider overlapping of classes. But overlapping of classes occur in most practical classification problems. For instance, the cluster of foxes and dogs should overlap in our last example as their speed and height to weight ratio are very close. So, from the measurements of these two features only, one is unable to say whether the unknown animal, which we want to classify is a fox or a dog. Such problems can be handled using fuzzy clustering. Here a given animal can belong to different classes with a membership value in the interval [0..1], but sum of its memberships of belonging to different classes must be 1. One such classical clustering algorithm proposed by Prof. Bezdek, the father of fuzzy pattern recognition, is popularly known as **Fuzzy c-means clustering (FCM)** [2-3]. The next section will outline this algorithm.

## 5.2 The Fuzzy C-means Clustering Algorithm

The objective of the fuzzy c-means clustering algorithm is to classify a given set of  $p$  dimensional data points  $X = [x_1 \ x_2 \ x_3 \ \dots x_n]$  into a set of  $c$  fuzzy classes or partitions  $A_i$  [6], represented by clusters, such that the sum of the memberships of any component of  $X$ , say  $x_k$ , in all the  $c$  classes is 1. Mathematically, we can represent this by

$$\sum_{i=1}^c \mu_{A_i}(x_k) = 1, \text{ for all } k = 1 \text{ to } n. \quad (5.1)$$

Further, all elements of  $X$  should not belong to the same class with membership 1. This is so because otherwise there is no need of the other classes. Thus mathematically, we state this by

$$0 < \sum_{k=1}^n \mu_{A_i}(x_k) < n. \quad (5.2)$$

For example, if we have 2 fuzzy partitions  $A_1$  and  $A_2$ , then for a given  $X = [x_1 \ x_2 \ x_3]$  say, we can take

$$\mu_{Ai} = [0.6/x_1 \ 0.8/x_2 \ 0/x_3] \text{ and}$$

$$\mu_{A2} = [0.4/x_1 \ 0.2/x_2 \ 1/x_3].$$

It is to be noted that the conditions described by expressions (5.1) and (5.2) are valid in the present classification.

Given  $c$  classes  $A_1, A_2, \dots, A_c$  we can determine their cluster centers  $V_i$  for  $i = 1$  to  $c$  by using the following expression.

$$V_i = \left[ \sum_{k=1}^n [\mu_{Ai}(x_k)]^m x_k \right] / \sum_{k=1}^n [\mu_{Ai}(x_k)]^m \quad (5.3)$$

Here,  $m (> 1)$  is any real number that influences the membership grade. It is to be noted from expression (5.3) that the cluster center  $V_i$  basically is the weighted average of the memberships  $\mu_{Ai}(x_k)$ . A common question now naturally arises: can we design the fuzzy clusters  $A_1, A_2, \dots, A_c$  in a manner so that the data point (feature) vector  $x_k$  for any  $k$  is close to one or more cluster centers  $V_i$ . This can be formulated by a performance criterion given by

Minimize  $J_m$  over  $V_i$  (for fixed partitions  $U$ ) and  $\mu_{Ai}$  (for fixed  $V_i$ )

$$J_m(U, V_i) = \sum_{k=1}^n \sum_{i=1}^c [\mu_{Ai}(x_k)]^m \|x_k - V_i\|^2 \quad (5.4)$$

$$\text{subject to } \sum_{i=1}^c \mu_{Ai}(x_k) = 1 \quad (5.5)$$

where  $\| . \|$  is an inner product induced norm in  $p$  dimension.

First let us consider the above constrained optimization problem for fixed  $V_i$ . Then we need to satisfy (5.4) and (5.5) together. Using Lagrange's multiplier method, we obtain that the problem is equivalent to minimizing

$$L(U, \lambda) = \sum_{k=1}^n \sum_{i=1}^c (\mu_{Ai}(x_k))^m \|x_k - V_i\|^2 - \sum_{k=1}^n \lambda_k (\sum_{i=1}^c \mu_{Ai}(x_k) - 1) \quad (5.6)$$

without constraints. The necessary condition for this problem is

$$\frac{\partial L}{\partial \mu_{Ai}} = m (\mu_{Ai}(\mathbf{x}_k))^{m-1} \| \mathbf{x}_k - \mathbf{V}_i \|^2 - \lambda_k = 0 \quad (5.7)$$

$$\text{and } \frac{\partial L}{\partial \lambda_k} = \sum_{i=1}^c \mu_{Ai}(\mathbf{x}_k) - 1 = 0. \quad (5.8)$$

From (5.7) we have

$$\mu_{Ai}(\mathbf{x}_k) = \left( \frac{\lambda_k}{m \| \mathbf{x}_k - \mathbf{V}_i \|^2} \right)^{1/(m-1)} \quad (5.9)$$

Substituting (5.9) in (5.8) we find:

$$\left( \frac{\lambda_k}{m} \right)^{1/(m-1)} = \frac{1}{\sum_{i=1}^c (1/\| \mathbf{x}_k - \mathbf{V}_i \|^2)^{1/(m-1)}}, \quad (5.10)$$

Substituting (5.10) in (5.9) we have

$$\mu_{Ai}(\mathbf{x}_k) = \left[ \sum_{j=1}^c \left( \frac{\| \mathbf{x}_k - \mathbf{V}_i \|^2 / \| \mathbf{x}_k - \mathbf{V}_j \|^2}{m} \right)^{1/(m-1)} \right]^{-1} \quad (5.11)$$

for  $1 \leq i \leq c$  and  $1 \leq k \leq n$ .

Now, suppose  $\mu_{Ai}(\mathbf{x}_k)$ 's are fixed. Then this is an unconstrained minimization problem, and the necessary condition is

$$\frac{\partial J_m}{\partial \mathbf{V}_i} = - \sum_{k=1}^n 2 (\mu_{Ai}(\mathbf{x}_k))^m (\mathbf{x}_k - \mathbf{V}_i) = 0 \quad (5.12)$$

which yields (5.3).

This is a great development which led to the foundation of the fuzzy c-means clustering algorithm. The algorithm is formally presented below.

### **Procedure Fuzzy c-means clustering**

**Input:** Initial pseudo-partitions  $\mu_{Ai}(x_k)$  for  $i=1$  to  $c$  and  $k=1$  to  $n$

**Output:** Final cluster centers

**Begin**

**Repeat**

**For**  $i:=1$  to  $c$

    Evaluate  $V_i$  by expression (5.3);

**End For;**

**For**  $k:=1$  to  $n$

**For**  $i:=1$  to  $c$

    Call  $\mu_{Ai}(x_k)$  OLD\_ $\mu_{Ai}(x_k)$ ;

**If**  $||x_k - V_i||^2 > 0$

        Then evaluate  $\mu_{Ai}(x_k)$  by (5.11) and call it CURRENT\_ $\mu_{Ai}(x_k)$

**Else do**

**Begin**

                set  $\mu_{Ai}(x_k):=1$  and call it (them) CURRENT\_ $\mu_{Ai}(x_k)$

                and set  $\mu_{Aj}(x_k):=0$  for  $j \neq i$  and call it (them) CURRENT\_ $\mu_{Aj}(x_k)$ ;

**End If;**

**End For;**

**End For;**

**Until**  $|CURRENT_{\mu_{Ai}}(x_k) - OLD_{\mu_{Ai}}(x_k)| \leq \epsilon = 0.01$ (say)

**End.**

The procedure fuzzy c-means clustering inputs the  $x_k$ 's and the initial values of  $\mu_{Ai}(x_k)$ 's supplied by the user for  $i=1$  to  $c$  and  $k=1$  to  $n$ . It then computes  $V_i$  for  $i=1$  to  $c$ . If  $||x_k - V_i||^2 > 0$  then the procedure evaluates  $\mu_{Aj}(x_k)$  by (5.11) and defines  $\mu_{Aj}(x_k)$  as its current value. If  $||x_k - V_i||^2 = 0$  then the point  $x_k$  has been correctly located at the clusters and thus its current value of  $\mu_{Ai}(x_k)$ 's is set to 1, and consequently the current membership values of  $\mu_{Aj}(x_k)$  for  $j \neq i$  are set to 0. The outer repeat-until loop keeps track of the terminating condition ensuring that the algorithm will terminate only when  $|CURRENT_{\mu_{Ai}}(x_k) - OLD_{\mu_{Ai}}(x_k)| \leq \epsilon = 0.01$ (say). On execution of the procedure, it returns the location of the cluster centers.

**Example 5.1:** Consider a set of 15 2-dimensional data points  $x_1$  to  $x_{15}$  (Table 5.1), we want to classify them into 2 pseudo-partitions  $A_1$  and  $A_2$ , say. Let the components of each point  $x_k$  be  $x_{k1}$  and  $x_{k2}$ . Thus we can plot these 15 points with respect to  $x_{k1}$  and  $x_{k2}$  axes.

**Table 5.1:** A set of 15 2-dimensional data points

k	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$x_{k1}$	0	0	0	1	1	1	2	3	4	5	5	5	6	6	6
$x_{k2}$	0	2	4	1	2	3	2	2	2	2	1	2	3	0	4

Let the initial pseudo-partitions be  $A_1$  and  $A_2$ , such that

$$\mu_{A1} = \{0.854/x_1 \quad 0.854/x_2 \quad \dots \dots \quad 1/x_{15}\} \text{ and}$$

$$\mu_{A2} = \{0.146/x_1 \quad 0.146/x_2 \quad \dots \dots \quad 0/x_{15}\}.$$

We now follow the procedure evaluate the cluster centers for the 2-partitions. Here, the cluster center for partition  $A_1$  will have 2-coordinates one for  $x_{k1}$  and the other for  $x_{k2}$ . Let the  $x_{k1}$  coordinate for partition  $A_1$  be  $V_{11}$  and the  $x_{k2}$  coordinate for the same partition be  $V_{12}$ . Then with  $m=1.25$  we find:

$$V_{11} = \frac{\{(0.854)^{1.25} (0+0+0+1+1+1+2+3+4+5+5+5+6+6) + (1)^{1.25} (6)\}}{\{14 (0.854)^{1.25} + (1)^{1.25}\}}$$

$$= 3.042, \text{ and}$$

$$V_{12} = \frac{\{(0.854)^{1.25} (0+2+4+1+2+3+2+2+2+2+1+2+3+0) + (1)^{1.25} (4)\}}{\{14 (0.854)^{1.25} + (1)^{1.25}\}}$$

$$= 2.028.$$

The cluster centers  $V_{21}$  and  $V_{22}$  for the partition  $A_2$  can also easily be evaluated by replacing 0.854 by 0.146 and 1.0 by 0 in the last 2 expressions.

The computation of the new membership of each point  $x_k$  can now also be evaluated with the computed values of  $V_{11}$ ,  $V_{12}$ ,  $V_{21}$  and  $V_{22}$  and last coordinate of the points. After 6 iterations of execution of the algorithm the cluster centers are found as

$$V_1 = (V_{11}, V_{12}) = (0.88, 2.0) \text{ and}$$

$$V_2 = (V_{21}, V_{22}) = (5.14, 2.0).$$

The membership values of the points to belong to partition  $A_1$  and  $A_2$  after 6 iterations are found as

$$\mu_{A1} = \{0.99/x_1 \ 1/x_2 \ 0.99/x_3 \ 1/x_4 \ 1/x_5 \ 1/x_6 \ 0.99/x_7 \ 0.47/x_8 \ 0.01/x_9$$

$0/x_{10} \ 0/x_{11} \ 0/x_{12} \ 0.01/x_{13} \ 0/x_{14} \ 0.01/x_{15}$  } and

$$\mu_{A2} = \{0.01/x_1 \ 0/x_2 \ 0.01/x_3 \ 0/x_4 \ 0/x_5 \ 0/x_6 \ 0.01/x_7 \ 0.53/x_8 \ 0.99/x_9$$

$1/x_{10} \ 1/x_{11} \ 1/x_{12} \ 0.99/x_{13} \ 1/x_{14} \ 0.99/x_{15}\}.$

It may indeed be noted that sum of the memberships of any point  $x_k$  to belong to the 2 partitions  $A_1$  and  $A_2$  to be 1 is always maintained irrespective of the iterations.

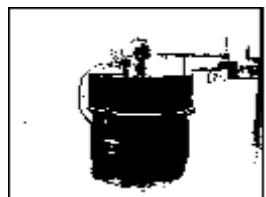
### 5.3 Image Segmentation Using Fuzzy C-Means Clustering Algorithm

A monochrome digital image usually is a two-dimensional array of gray pixels. On occasions, the components of the image are needed to be isolated. The process of isolating important regions of an image into components/ modules is generally referred to as image segmentation. A number of well-known algorithms of image segmentation is available in a textbook of image processing [12]. In this section, we following Raghukrishnapuram [7] present a study of image segmentation [12] using fuzzy c-means algorithm.

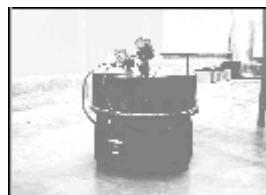
The fuzzy c-means clustering algorithm for image segmentation has 2 parameters, namely, the number of clusters, denoted by ‘c’, and the exponential weighting factor ‘m’ over the membership functions. The experiments are performed by gradually varying these two parameters and their effects on clustering are noted. The number of clusters needed is usually determined by the problem in hand. For segregating a dark object from a light background or vice versa, we should select  $c=2$  and thus obtain 2 clusters, one corresponding to the dark region and the other to the lighter region. Fig 5.2 shows the results of clustering with  $c=2$ . Further, the value of exponential weighting factor  $m$  has been increased in steps from  $m$  slightly greater than 1, followed by  $m=1.2$  and  $m=2.5$ . The variation of  $m$  clearly indicates the difference between a hard cluster and a soft cluster.



(a)



(b)

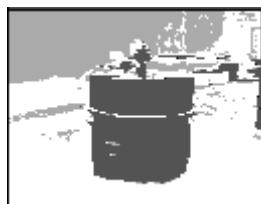


(c)



(d)

**Fig. 5.2:** Fuzzy C -means clustering algorithm applied to segment the given image (a) into 2 clusters with  $m=1.01$  in (b),  $m=1.2$  in (c), and  $m=2.5$  in (d).



(a)



(b)



(c)

**Fig. 5.3:** Fuzzy C-means clustering algorithm applied to segment the image in 2(a) into 3 Clusters with  $m=1.01$  in (a),  $m=1.2$  in (b) and  $m=2.5$  in (c).

For the purpose of illustration, we have constructed the figures using the membership value of each pixel mapped to gray value levels. The contrasting shades in Fig. 5.2(b) indicate that each pixel belongs to either of the classes with large membership value and with a very small membership value for the other class. This is typical of a hard cluster where the pixels are assigned to either of the two classes. In the subsequent figures 5.2(c) and 5.2(d), the 2 shades become less contrasting. As the shades are representing the membership values, it means that each pixel now have intermediate membership values of belonging to the 2 classes. This is fuzzy clustering where each pixel has finite memberships of belonging to the 2 classes. Thus, we have greater latitude of deciding which pixels to select based on their membership values.

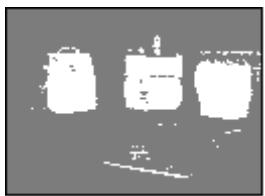
In Fig. 5.3, the number of clusters is 3 and they are represented by 3 shades: dark, intermediate gray and light. Here, too, we observe that how the value of  $m$  alters the final membership values of the pixels. As explained for Fig. 2, a value of  $m$  close to 1 ( $m=1.01$ ) generates crisp clusters while  $m=1.2$  or larger makes the clusters fuzzy.



(a)



(b)



(c)

**Fig. 5.4:** Fuzzy C-means clustering algorithm applied to isolate very black objects from the image in (a) with grayscale as the feature vector in (b), logarithm of the grayscale as the feature vector in (c).

In all the above experiments, we used gray level of the pixels as the feature of interest and accordingly constructed the feature vector. In Fig 5.4(b) it is

observed that although the clustering is done, we do not get the cluster as expected intuitively. This may be attributed to the fact that the general illumination of the image is so low that the black objects cannot be distinguished from the surrounding floor based on gray level alone. We propose that in such circumstances, some transformation of the gray level values be taken to construct the feature vector. The choice of this transformation is based on some prior knowledge of the object of interest.

To illustrate the above point, let us consider the case where we are interested in segregating some dark object from a poorly illuminated workspace. The suitable transformation in such a circumstance is to take the logarithm of the gray level values of the pixels. As we know, logarithmic transform expands and emphasizes the lower values and cramps the larger value in a smaller range, thus de-emphasizing them. Fig 5.4(c) illustrates the process of clustering with gray levels as the elements of the feature vector. It is now observed that this transformation has successfully clustered the ‘very’ dark objects from the dark environment. Thus, we have a third controlling factor, namely, the transformation on the gray level whereby we may classify the image according to the requirements of the problem in hand.

## 5.4 Conclusions

The chapter presented an algorithm for classification of n-dimensional patterns into c number of fuzzy classes. The importance of the algorithm lies in its capability to adjust the location of the cluster centers in the n-dimensional space. One interesting point to note about the algorithm is that the steady-state location of the cluster centers in the algorithm is insensitive to small variations in the initial choice of the memberships. The proposed algorithm has also successfully been applied in image segmentation problems. The experiment on image segmentation reveals an interesting observation. The darker regions in an image can be segregated from the dark region by applying the c-means algorithm on the logarithm of the gray scale intensity patterns of the image. This is useful in many applications especially when the darker object needs to be segmented from its shadow.

There exists plenty of interesting literature on fuzzy pattern classifier design. A few works that deserve special mention are available in [8-11].

## Exercise

1. Verify that after 6 iterations the cluster centers  $\mathbf{V}_1$  and  $\mathbf{V}_2$  for the problem considered in example 5.1 are found to be as follows.  $\mathbf{V}_1 = (0.88, 2.0)$  and

$V_2 = (5.14, 2.0)$ . Assume the initial pseudo partitions and the data points as given in Example 5.1.

2. Replace the initial memberships 0.854 by 0.754 and 0.146 by 0.246 in the above problem. Check whether you observe any change in the cluster center positions at steady state.
3. Represent an  $(n \times n)$  image into a  $(1 \times n^2)$  vector, by placing the consecutive rows of the image in cascade on the  $(1 \times n^2)$  array. Write a program on fuzzy c-means clustering algorithm and run it on the image vector. Check the effects of varying m and c in the program.

## References

- [1] Basak, J., De, R. K. and Pal, S. K., "Unsupervised feature selection using neuro-fuzzy approach," *Pattern Recognition Letters*, vol. 19, pp. 997-1006, 1998.
- [2] Bezdek, J. C., *Fuzzy Mathematics in Pattern Classification*, Ph D thesis, Applied Math. Center, Cornell University, Ithaca, 1973.
- [3] Bezdek, J. C., *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum Press, NY, 1981.
- [4] Bezdek, J. C. and Pal, S. K. (Eds.), *Fuzzy Models for Pattern Recognition: Methods that Search for Patterns in Data*, IEEE Press, NY, 1992.
- [5] Bezdek, J. C., Keller, J. M., Krishnapuram, R. and Pal, N. R., *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*, Kluwer Academic, MA, 1999.
- [6] Hathaway, R. J., Bezdek, J. C. and Hu, Y., "Generalized fuzzy c-means clustering strategies using  $L_p$  norm distances," *IEEE Trans. on Fuzzy Systems*, vol. 8, no. 5, October 2000.
- [7] Krishnapuram, R., Segmentation, In *Handbook of Fuzzy Computation*, Ruspini, E. H., Bonissone, P. P. and Pedrycz, W. (Eds.), IOP Pub. Ltd., 1998.
- [8] Nath, A. K. and Lee, T. T., "On the design of a classifier with linguistic variables as input," *Fuzzy Sets and Systems*, vol. 11, pp. 265-286, 1983.

- 
- [9] Nath, A. K., Liu, S. W. and Lee, T. T., "On some properties of linguistic classifier," *Fuzzy Sets and Systems*, vol. 17, pp. 297-311, 1985.
  - [10] Pal, S. K. and Majumder, D. K. D., *Fuzzy Mathematical Approach to Pattern Recognition*, John Wiley, NY, 1986.
  - [11] Pal, S. K. and Mitra, S., *Neuro-Fuzzy Pattern Recognition: Methods in Soft Computing*, John Wiley & Sons, NY, 1999.
  - [12] Pratt, W. K., *Digital Image Processing*, Wiley-Interscience, NY, 1978.

# 6

# Fuzzy Databases and Possibilistic Reasoning

*The chapter provides a possibilistic interpretation of fuzzy relational databases containing imprecise and noisy data. It proposes fuzzy equality relations, and represents fuzzy functional dependency using such relations. It also outlines a novel scheme for testing the lossless join decomposition of fuzzy relational databases. The chapter finally employs the above two concepts in the design of fuzzy relational databases. The concepts outlined in the chapter have been illustrated with many examples.*

## 6.1 Introduction to Relational Database Systems

Database management became increasingly important in scientific and commercial applications, including space research, rocket launching, electronic commerce, digital library and banking transactions. There exist quite a large

variety of data models such as hierarchical, network, relational, object oriented and logic program based models. Among these data models, the relational model is widely used in most commercial database packages like INGRESS, ORACLE and SYBASE. In this chapter, we present a fuzzy extension of the relational paradigm. Before introducing the fuzzy extension, we briefly outline the relational model and fundamentals of database design using this model.

### 6.1.1 The Relational Model

Let  $U = \{A_1, A_2, \dots, A_n\}$  be a universal set of attributes of a relation  $r$ . We call  $R(A_1, A_2, \dots, A_n)$  as the relational schema. Each attribute  $A_i$  has a domain  $\text{dom}(A_i)$  associated with it. The domain  $\text{dom}(A_i)$  denotes the possible valuation space for the attribute  $A_i$ . Usually lower case letters such as  $\{a_{i1}, a_{i2}, \dots, a_{in}\}$  are used to describe the domain of attribute  $A_i$ .

A relation  $r$  having attributes  $\{A_1, A_2, \dots, A_n\}$  is a subset of the Cartesian product of domains of the attributes, i.e.,

$$r \subseteq \text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_n) \quad (6.1)$$

Thus a relation may be considered to be an instance of a relational schema  $R(A_1, A_2, \dots, A_n)$ .

An element of a relation is called a *tuple* or *row*. For example, for a relational schema  $R(A, B, C)$ , let  $\text{dom}(A) = \{a_1, a_2\}$ ,  $\text{dom}(B) = \{b_1, b_2\}$  and  $\text{dom}(C) = \{c_1\}$ . Then we have 4 tuples  $a_1 b_1 c_1$ ,  $a_1 b_2 c_1$ ,  $a_2 b_1 c_1$  and  $a_2 b_2 c_1$ .

If  $t$  is a tuple in a relational schema  $R(A_1, A_2, \dots, A_n)$ , then  $t[A_1]$  denotes the  $A_1$  component of the tuple  $t$ . For example, let  $\{a_{j1}, a_{j2}, \dots, a_{jn}\}$  be a tuple of  $r$ , then  $t[A_1] = a_{j1}$ , and  $t[A_1 A_2] = a_{j1} a_{j2}$ . The notion of  $t$  thus denotes a restriction on one or more attributes. Finally if  $U = \{A_1, A_2, \dots, A_n\}$  and  $X \subseteq U$ , then  $t[X]$  is called a restriction of  $t$  to  $X$ .

**Example 6.1:** Consider a Students relational schema  $S$  (name, roll no, % of marks), given in Table 6.1.

**Table 6.1:** Student's database

	name	roll no.	% of marks
$S =$	A. Das	12	82
	B. Banik	13	86
	C. Kumar	14	92

where,

$$\text{dom(name)} = \{\text{A. Das, B. Banik, C. Kumar}\}$$

$$\text{dom(roll no.)} = \{12, 13, 14\},$$

$$\text{dom } (\% \text{ of marks}) = \{82, 86, 92\} .$$

The relational schema includes 3 tuples.

$$\text{A.Das} \quad 12 \quad 82$$

$$\text{B.Banik} \quad 13 \quad 86$$

$$\text{and C. Kumar} \quad 14 \quad 92$$

Further, the name and roll no. attributes (fields) of the tuple  $t = \text{A.Das } 12 \ 82$  is given by  $t[\text{name roll no.}] = \text{A.Das } 12$ .

For answering queries in relational model, we often require the following 3 relational algebraic operations: natural join, projection and selection. Let  $r_1$  and  $r_2$  be two relations over  $(A_1 \ A_2 \ A_3)$  and  $(A_1 \ A_4)$ . The natural join of  $r_1$  and  $r_2$  denoted by  $r_1 \blacktriangleright \blacktriangleleft r_2$  is a relation  $r$  over  $(A_1, A_2, A_3, A_4)$  and defined by

$$r = r_1 \blacktriangleright \blacktriangleleft r_2$$

$$= \{ t \mid t [ A_1 A_2 A_3 ] \in r_1 \text{ and } t [ A_1 A_4 ] \in r_2 \} \quad (6.2)$$

The projection of the attributes  $A_i \ A_j \ A_k$  from a given relational scheme  $R$   $(A_1, A_2, \dots, A_i, \dots, A_j, \dots, A_k, \dots, A_n)$ , denoted by  $\Pi_{A_i, A_j, A_k}(r)$  and is defined below:

$$\Pi_{A_i, A_j, A_k}(r) = \{ t [ A_i, A_j, A_k ] \mid t \in r \}. \quad (6.3)$$

The selection of tuples that satisfy a given condition in a relational Algebra is formalized as follows. Let  $C$  be a condition, then selection of tuples  $t$ , that satisfy the condition  $C$  in a relational model is denoted by  $\sigma_c(r)$ . The selection operation is formally defined by

$$\sigma_c(r) = \{ t \mid \text{where condition } c \text{ is satisfied} \}$$

Here  $c$  is a Boolean condition, applied on the attributes of  $R$  ( $A_1, A_2, \dots, A_n$ ). Typical conditions, for instance, include  $A_i > A_j$ ,  $A_i \geq A_j$ ,  $A_i = A_j$ ,  $A_i \neq A_j$ , and  $A_i >$  a given constant etc.

**Example 6.2:** This example illustrates the three elementary relational operators with respect to the following relational schema of grade card of a particular class of students. Let  $R_1$  ( Name, Roll, Phy, Che, Math, Bio) and  $R_2$  ( Name, Roll, Eng) be two such relational schema, where Phy, Che, Math, Bio, Eng. denote marks obtained by students of a class in Physics, Chemistry, Mathematics, Biology and English literature respectively.

**Table 6.2:** A partial grade card information

	name	Roll no.	Phy	Chemistry	Maths	Biology
$R_1 =$	A. Roy	10	80	76	82	90
	B. Basak	11	76	82	95	80
	C. Kumar	12	48	62	80	75

**Table 6.3:** Marks of English Literature

	Name	Roll.no.	English
$R_2 =$	A. Roy	10	68
	B . Basak	11	72
	C. Kumar	12	59

$r_1 \bowtie r_2$  is presented in table 6.4.

**Table 6.4:** The natural join of  $r_1$  and  $r_2$

Name	Roll. No.	Physics	Chemistry	Maths	Biology	English
A. Roy	10	80	76	82	90	68
B. Basak	11	76	82	95	80	72
C. Kumar	12	48	62	80	75	59

Projection of marks in Physics, Chemistry and Maths from the tuple

---

$t = A.$  Roy 10 80 76 82 90

of relation  $r_1$  is given by

$$\Pi_{\text{Phy}, \text{Che}, \text{Math}}(r_1) = (80 \quad 76 \quad 82).$$

Now, to illustrate the selection operation, suppose we need to identify the students whose marks in Chemistry is greater than their respective marks obtained in Physics. We can describe this selection by

**Table 6.5:** Selection of tuples satisfying Che>Phy

$\sigma_{\text{Che} > \text{Phy}}(r_1) =$	Name	Roll.no	Phy	Che	Math	Bio
	B.Basak	11	76	82	95	80
	C.Kumar	12	48	62	80	75

## 6.2 Issues in Relational Database Design

Generally, a relational schema  $R(A_1, A_2, \dots, A_n)$  for commercial databases includes many redundant entries. For example, consider a relational schema  $R(S, SA, I, P)$  where  $S$ ,  $SA$ ,  $I$  and  $P$  denote the supplier name, suppliers' address, items and parts supplied by different venders. Here a supplier may supply more than one item and parts. Consequently his address will be occurring more than once in the schema. Further some suppliers might have stopped supplying items and parts, so their corresponding tuples contain supplier-name and supplier address repeatedly.

The objective of relational database design is to ensure that the relational schemas should not include repeated entries. One way to handle this problem is to represent the relational schema  $R$  into row or sub – schemas such that the latter together equivalently describe the former relational schema  $R$ . To ensure the last point, we need to define two important constraints in describing equivalence of a relational schema  $R$  by sub- schemas  $R_1, R_2$  and others . The constraints referred to above are popularly known as a) lossless join decomposition and b) functional dependency preservation.

### 6.2.1 Lossless Join Decomposition

Let  $r$  be a relation, which is decomposed into  $r_1, r_2, \dots, r_s$  in a manner, such that

$$r_1 = \prod_{R_1}(r) \quad (6.4)$$

$$r_2 = \prod_{R_2}(r) \quad (6.5)$$

$$\text{and } r_s = \prod_{R_s}(r) \quad (6.6)$$

where  $r_1, r_2, \dots, r_s$  are the relational obtained by projecting the appropriate attributes of  $R_1, R_2, \dots, R_s$  from  $R$ .

The condition for lossless join requires that the natural join of  $r_1, r_2, \dots, r_s$  yields the original relation  $r$ .

By notation,

$$r = r_1 \bowtie r_2 \bowtie r_3 \dots r_{s-1} \bowtie r_s \quad (6.7)$$

As an example, if the relational schema  $R = (S, SA, I, P)$  is broken into two relational schema  $R_1 = (S, SA)$  and  $R_2 = (I, P)$ , then it can be shown that

$$r = r_1 \bowtie r_2$$

is maintained. Thus the given decomposition  $R_1$  and  $R_2$  satisfy the lossless join conditions.

### 6.2.2 Functional Dependency Preservation

Let  $R (A_1, A_2, \dots, A_n)$  be a relational schema. Let  $\text{dom}(A_i)$  denote the domain of  $A_i$ . Let  $t_1$  and  $t_2$  be two tuples of  $r$ . If  $t_1 [A_i] = t_2 [A_i]$  then  $t_1 [A_j] = t_2 [A_j]$  hold, for any arbitrary tuples of  $r$ , then we say

$$A_i \rightarrow A_j \quad (6.8)$$

is a functional dependency in  $r$ . In case, we find for  $t_1 [A_i] = t_2 [A_i]$ ,  $t_1 [A_j A_k] = t_2 [A_j A_k]$  also holds, we say

$$A_i \rightarrow A_j A_k \quad (6.9)$$

is a functional dependency in  $r$ .

In general, if equality of one or more attributes ensures equality of some attributes of any two tuples  $t_1$  and  $t_2$  of  $r$ , then a functional dependency exists from the former to the latter set of attributes.

Armstrong (vide [5]) defined a set of inference axioms on functional dependencies. These axioms are needed to determine the exhaustive set of functional dependencies, called the closure of functional dependencies. The list of axioms is presented below.

### Armstrong's axioms

Let  $X$  and  $Y$  be two sets of attributes in a relation  $r$ .

1. If  $Y \subseteq X$  then  $X \rightarrow Y$  is a functional dependency in  $r$  (reflexivity).
2. If  $X \rightarrow Y$  is a functional dependency in  $r$ , then  $XZ \rightarrow YZ$  is also a valid functional dependency in  $r$  (augmentation). It is to be noted that  $XZ$  stands for  $X \cup Z$ , similarly  $YZ$  stands for  $Y \cup Z$ .
3. If  $X \rightarrow Y$  and  $Y \rightarrow Z$  hold in  $r$ , then  $X \rightarrow Z$  also holds in  $r$  (transitivity)

**Closure of functional dependencies:** Given a set  $F$  of functional dependencies. The closure of  $F$ , denoted by  $F^+$ , represents the exhaustive set of functional dependencies derived from  $F$  using Armstrong's axioms.

**Example 6.3:** Given a relational schema  $R$  ( $A, B, C, D$ ) and let  $F = \{A \rightarrow B, B \rightarrow C, AC \rightarrow D\}$

By Armstrong's axiom 3,  $A \rightarrow C$  holds (as  $A \rightarrow B$  and  $B \rightarrow C$  hold) so,  $F^+ = \{A \rightarrow B, A \rightarrow C, B \rightarrow C, AC \rightarrow D\}$  is the closure of  $F$ .

A formal technique for computation of closure of  $F$  is available in any text book on database systems [5].

**Functional dependency Preservation:** When a given relation  $r$  is decomposed into  $r_1, r_2, \dots, r_n$ , it should be checked that the  $F^+$  of  $r$  is preserved in the  $F_1^+ \cup F_2^+ \cup \dots \cup F_n^+$  where  $F_i^+$  for  $1 \leq i \leq n$  denotes the closure of functional dependencies of  $r_i$ .

**Example 6.4:** The relational schema

$R(SN, SA, I, P)$  has functional dependencies

$$F = \{SN \rightarrow SA, SN, SA \rightarrow I, P\}.$$

Here,  $F^+ = F$ . If  $R$  is decomposed into  $R_1$  (SN, SA) and  $R_2$  (SN, I, P) then by decomposition, the second functional dependency  $SN, SA \rightarrow I, P$  is not supported by  $r_1$  or  $r_2$ . Thus, functional dependency preservation fails in the decomposition process.

**Criteria for a database design:** The following two basic criteria should be satisfied in a relational database design.

1. *Lossless join condition should be maintained in decomposing a relation.*
2. *Functional dependency preservation criteria should be satisfied.*

### 6.3 Possibility Distribution and Fuzzy Sets

In this section, we briefly outline the notion of possibility distribution with an ultimate aim to apply the concept in fuzzy databases. To introduce the topic, let us consider a fuzzy set HIGH-SALARY, as defined below.

$$\text{HIGH-SALARY} = \{0.1/ 20k, 0.3 / 30k, 0.5 / 40k, 0.7 / 50k, 0.9/70k, \\ 0.95/80k, 1.0/ 90k \} \quad (6.10)$$

where the terms like  $a/b$  denote a particular salary  $b$  and the membership of  $b$  to be a HIGH-SALARY, i.e.  $a = \mu_{\text{HIGH-SALARY}}(b)$ .

Suppose, it is given that John has a High-Salary. Thus according to the possibilistic interpretation, the possibility of John having a salary of 40k is 0.5.

According to Zadeh [7], a fuzzy proposition “ $X$  is  $F$ ”, where  $F$  is a fuzzy subset of  $U$  and  $X$  is a fuzzy variable that takes values from the universe  $U$ . Here  $U$  includes a possibility distribution  $\Pi_x$  that is equal to  $F$ , formally,

$$\Pi_x = F. \quad (6.11)$$

Thus, the possibility of  $X = u$  is denoted by  $\text{Poss}(X = u) = \mu_F(u)$  for all  $u \in U$ . One may also define a function  $\Pi_x: U \rightarrow [0, 1]$  which is equal to  $\mu_F$  and associates with each  $u \in U$  the possibility that  $X$  could take  $u$  as its value, i.e.,

$$\Pi_x(u) = \text{Poss}(X = u) = \mu_F(u) \text{ for all } u \in U \quad (6.12)$$

The function  $\Pi_x$  is popularly known as possibility distribution function of  $X$ . The possibilistic distribution  $\Pi_x$  may also be employed to define a fuzzy measure  $\Pi$  on  $U$ , for any  $A \subseteq U$ .

$$\Pi(A) = \text{Poss}(X \in A) = \sup_{u \in A} \Pi_X(u) \quad (6.13)$$

Possibility distribution of compound fuzzy propositions will be computed in a subsequent section from the knowledge of possibility distributions of its constituent propositions.

## 6.4 Fuzziness in Relational Models

Let  $r$  be a fuzzy relation on a relational schema  $R(A_1, A_2, \dots, A_n)$ . Let  $\text{dom}(A_i)$  be the domain of  $A_i$  for  $1 \leq i \leq n$ . Then fuzzy relation  $r$  is a subset of  $\text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_n)$ .

Depending on the complexity of domain  $(A_i)$ , for  $i = 1, \dots, n$ , fuzzy relations are classified into two categories, called type - 1 and type-2 fuzzy relations. In type-1 fuzzy relations,  $\text{dom}(A_i)$  can only be a classical/ fuzzy sets. The type-2 fuzzy relation has the advantage to represent a wider type of imprecision in data values. It allows the  $\text{dom}(A_i)$  to be even a set of fuzzy sets.

It should be mentioned here that organization of a fuzzy relation is similar with typical non-fuzzy relations, but a fuzzy relation had an additional column for  $\mu_r(t)$ , which denotes the membership of the tuple  $t$  in  $r$ . It needs also mention that only those tuples  $t$  having  $\mu_r(t) > 0$  are included in the relation.

### 6.4.1 Type-1 Fuzzy Relational Data Model

As stated in section 6.4,  $\text{dom}(A_i)$  in a type-1 fuzzy relation may be a classical fuzzy subset of  $U_i$ . Let  $u_i$  be a fuzzy variable in the universe  $U_i$  for  $1 \leq i \leq n$ , then  $\mu_r(t)$  of a tuple  $t$  in relation  $r$  is defined as

$$\mu_r(t) = \mu_r(u_1, u_2, \dots, u_n) \leq \min [\mu_{A1}(u_1), \mu_{A2}(u_2), \dots, \mu_{An}(u_n)] \quad (6.14)$$

According to probabilistic interpretation of fuzzy sets,  $\mu_r$  can be treated as a possibility distribution function in  $U$ , where

$$U = U_1 \times U_2 \times \dots \times U_n. \quad (6.15)$$

Consequently,  $\mu_r(u_1, u_2, \dots, u_n)$  represents the possibility measure that a tuple  $t \in U$  has  $t[A_i] = u_i$  for  $1 \leq i \leq n$ .

Examples 6.5 and 6.6 below respectively illustrate fuzzy relations, having classical and fuzzy domains for attributes  $A_i$ , for  $i=1,2,\dots,n$ .

**Example 6.5:** Let us consider a relational schema Likes(Student, Course), where domains:  $\text{dom}(\text{student})$  and  $\text{dom}(\text{course})$  are classical sets. In other words, the domains here are crisp.  $\mu_r(t)$  in this particular context denotes the possibility measure of a particular student liking a specific course. For instance the possibility measure of A. Konar liking an Algorithms Course is 0.90 (Table 6.6)

**Table 6.6:** Likes (Students, Course) relation

Name	Course	$\mu_r$
A. Konar	Algorithms	0.90
A. K. Nath	Communication	0.70
A. R. Saha	VLSI	0.80

**Example 6.6:** To illustrate fuzzy relations with attributes defined on a fuzzy domain, let us consider the following relational schema:  $R(N, J, X, S)$ , where N denotes name of the employee, J stands for his/her job, X represents his/ her experience and S denotes his/her salary. Here,  $\text{dom}(N)$  and  $\text{dom}(J)$  are classical sets, and  $\text{dom}(X)$  and  $\text{dom}(S)$  are fuzzy sets. HIGH – EXPERIENCE and HIGH- SALARY are subsets of the universes  $U_x$  and  $U_s$  respectively, where

$$X \in U_x = [0..30] \text{ and}$$

$$S \in U_s = [10k..100k].$$

Membership function  $\mu_{HX}(X)$  and  $\mu_{HS}(S)$  of the fuzzy sets HIGH- EXPERIENCE and HIGH- SALARY are presented below.

$$\left. \begin{aligned} \mu_{HX}(X) &= [1 + |X - 10| / 4]^{-1}, X \leq 10 \\ &= 1, \text{ otherwise} \end{aligned} \right\} \quad (6.16)$$

$$\left. \begin{aligned} \mu_{HS}(S) &= [1 + |S - 60,000| / 20,000]^{-1}, \text{ for } x \leq 60,000 \\ &= 1, \text{ otherwise} \end{aligned} \right\} \quad (6.17)$$

The membership  $\mu_r(t)$  for a tuple  $t$  in  $r$ , thus can be interpreted as the truth value of the fuzzy proposition that “N has HIGH- EXPERIENCE and HIGH- SALARY.” with reference to Table 6.7, the membership of “L. Jain had high salary and high experience “ is found to be

$$\text{Min} [\mu_{HX}(8), \mu_{HS}(60,000)]$$

$$= \text{Min}[0.67, 1]$$

$$= 0.67.$$

**Table 6.7:** University employee relation

Name	Job	Experience	Salary	$\mu_r$
L. Jain	Professor	8	60,000	0.67
A. Nath	Professor	9	70,000	0.80
A. Saha	Reader	8	40,000	0.50

#### 6.4.2 Type-2 Fuzzy Relational Data Model

In a type-1 relational schema Employee(Name, Salary), one is not permitted to specify salary of A. Konar to be in the range of \$ 40, 000 – 50, 000 and that of A. K. Nath to be a fuzzy set “Low”. To accommodate such data ambiguities, a further generalization of the fuzzy relational model is considered next. One way to generalize fuzzy relational model is to construct fuzzy domains such as  $\text{dom}(A_i)$  for the data attributes  $A_i$ , for  $i = 1$  to  $n$ . Consequently, a tuple  $t = (a_1, a_2, \dots, a_n)$  in domain

$D = \text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_n)$  becomes a fuzzy subset.

$$U = U_1 \times U_2 \times \dots \times U_n$$

with

$$\mu_t(u_1, u_2, \dots, u_n) = \text{Min} [\mu_{a1}(u_1), \mu_{a2}(u_2), \dots, \mu_{an}(u_n)] \quad (6.18)$$

A type- 2 fuzzy relation  $r$  thus is a fuzzy subset of  $D$ , where

$$\mu_r : D \rightarrow [0,1]$$

and for any tuple  $t = (a_1, a_2, \dots, a_n) \in D$

$$\mu_r(t) \leq \max [\min \{\mu_{a1}(u_1), \mu_{a2}(u_2), \dots, \mu_{an}(u_n)\}] \quad (6.19)$$

$$(u_1, u_2, \dots, u_n) \in U$$

In this case of type-2 fuzzy relations may be interpreted either as a probabilistic measure of associations among the data values or as a truth value

of a fuzzy predicate associated with  $r$ . With respect to the relation: Employer (Name, Salary), consider an instance of a tuple ( A. K. Nath , S ),

where

$$S = \{0.3 / 10k, 0.6 / 20k, 0.8 / 30k\}.$$

Thus  $S$  represents a possibility distribution for the salary of A. K. Nath. For instance,

$$\text{Poss}(\text{Salary of A. K. Nath}, 20k) = 0.6.$$

The possibilistic interpretation for a tuple  $t$  of  $r$ , may be computed by the following expression:

$$\begin{aligned} \text{Poss}(t[A_1] = u_1, t[A_2] = u_2, \dots, t[A_n] = u_n) \\ = \text{Min}[\mu_r(t), \mu_t(u_1, u_2, \dots, u_n)]. \end{aligned} \quad (6.20)$$

where  $\mu_t$  is given by (6.18).

**Example 6.6:** Given an EMPLOYEE(N, D, J, X, S, I) relational schema, where N = Name of the Employee, D= Department, J= Job, X= Experience, S= Salary , and I= Income tax (I/T). Here, domains:  $\text{dom}(N)$ ,  $\text{dom}(D)$  and  $\text{dom}(J)$  are ordinary sets, whereas  $\text{dom}(X)$ ,  $\text{dom}(S)$  and  $\text{dom}(I)$  are sets of fuzzy sets in universes  $U_x$  ,  $U_s$  and  $U_i$  respectively. Thus X,S, and I are fuzzy linguistic variables, where

$$X \in U_x = [0..30]$$

$$S \in U_s = [10k..100k]$$

and  $I \in U_I = [0..10k]$  respectively.

Fuzzy sets HIGH, LITTLE, MODERATE etc. may be constructed to describe the membership of X, S and I in the above universes. These membership functions are constructed intuitively. For instance, the membership of experience X to be MODERATE and LITTLE are given by

$$\begin{aligned} \mu_{\text{MODERATE}}(x) &= (1 + |x - 8|)^{-1}, \text{ for } x \geq 8 \\ &= 0, \text{ otherwise.} \end{aligned}$$

$$\mu_{LITTLE}(x) = (1+12x)^{-1}, \text{ for } x > 0$$

= 0, otherwise.

Similarly, the membership of salary to be high be described by the following membership function:

$$\begin{aligned}\mu_{HIGH}(S) &= (1+a|S-C|)^{-1}, \text{ for } S \leq C \\ &= 1, \quad \quad \quad \text{for } S > C.\end{aligned}$$

where  $a = 1/20,000$  and  $c = 60,000$  are presumed for the relational database of Table 6.8.

**Table 6.8:** The Employee Relation

Name	Dept	Job	Experience	Salary	I/T	$\mu_r$
A.B.Roy	Maths	Professor	15-20	HIGH	5k	0.90
A.K.Nath	ETCE	Lecturer	LITTLE	Low	Low	0.60
A.Konar	ETCE	Reader	MODERATE	40k-60k	4k-6k	0.80

Let us now consider the first tuple of Table 6.8. Suppose, we are interested to evaluate the possibility distribution of Mr. A. B. Roy to have a 18 years moderate experience and around Rs. 50,000 salary, which is given by

$$\text{Min} [\mu_{MODERATE}(18), \mu_{HIGH}(50,000)]$$

$$= \text{Min}(1/11, 2/3)$$

$$= 1/11$$

$$= 0.09.$$

Now, since  $\mu_t$  for the first tuple = 0.90,

$$\text{Poss}([t | X=18 \text{ years}] = MODERATE) \& \text{ Poss}([t | S=50,000] = HIGH)$$

$$= \text{Min} [\mu_t, \text{Min} [\mu_{MODERATE}(18), \mu_{HIGH}(50,000)]]$$

$$= \text{Min} [0.70, 0.09]$$

$$= 0.09.$$

## 6.5 Fuzzy Relational Operators

We define two fuzzy relational operators: projection and join to introduce the notion of lossless join of fuzzy relations.

### 6.5.1 Projection of Fuzzy Relations.

Let  $r$  be an instance of fuzzy relations of a relational schema  $R$  ( $A_1, A_2, \dots, A_n$ ). We consider a subset  $R_i$  ( $A_{i1}, A_{i2}, \dots, A_{ik}$ ) of  $R$ . Now, for a given tuple  $t$  of  $r$  (i.e.,  $\mu_r(t) > 0$ ,  $t[R_i]$  denotes the restriction of  $t$  on the attributes of  $r$ . For example, if  $t = (a_1, a_2, \dots, a_n)$ ,  $t[R_i] = (a_{i1}, a_{i2}, \dots, a_{ik})$ . According to Professor Zadeh [7], projection  $r_i = \prod_{R_i}(r)$  is a  $k$ -ary fuzzy relation in  $\text{dom}(A_{i1}) \times \text{dom}(A_{i2}) \times \dots \times \text{dom}(A_{ik})$ . The membership function  $\mu_{ri}$  is given by

$$\mu_{ri}(t) = \max_{t_r} (\mu_r(t_r) \mid t_r[R_i] = t[R_i]) \quad (6.21)$$

where  $t_r$  is a tuple of  $r$ .

**Example 6.8:** Consider a fuzzy relation  $r$ , given in Table 6.9. The projection of job and salary from relation  $r$  is shown in Table 6.10.

**Table 6.9:** The relational schema  $R$  (Name, Job, Salary)

Name	Job	Salary	$\mu_r$
AA	Engineer	10k	0.6
BB	Engineer	10k	0.8
CC	Manager	60k	0.9
DD	Manager	60k	0.8
EE	Secretary	40k	0.6

**Table 6.10:** The projection of job and salary from Table 6.9

Name	Salary	$\mu_r$
Engineer	10k	0.8
Manager	60k	0.9
Secretary	40k	0.6

It is clear from Table 6.9 and 6.10 that for common entries in job and salary, the tuple having the largest  $\mu$  in table 6.9 is included in Table 6.10.

### 6.5.2 Cylindrical Extension of Fuzzy Relations

Cylindrical extension of a projection relation  $r_i = \prod_{R_i}(r)$  to the original relation  $r$  is an important operation in fuzzy relational database. To formalize the definition of cylindrical projection, let  $r_i$  be an instance of a relational schema  $R_i = (A_{i1}, A_{i2}, \dots, A_{ik})$ . Let us also consider a relational schema  $R = (A_1, A_2, \dots, A_n)$ , such that  $R_i \subseteq R$ . Then cylindrical extension of  $r_i$  on  $R$ , denoted by  $r_i^c = C_R(r_i)$ , is thus an n-ary fuzzy relation in  $\text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_n)$ . The membership  $\mu_{r_i^c}$  of  $r_i^c$  is given by

$$\mu_{r_i^c}(t) = \mu_{r_i}(t [ R_i ]), \quad (6.22)$$

for  $t$  in  $D = \text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_n)$ .

It is clear from definition (6.22) that for any instance  $r$  of a relational schema  $R$  and  $R_i \subseteq R$ ,

$$\mu_{r_i^c}(t) \geq \mu_r(t) \quad (6.23)$$

where  $r_i^c = C_R(\prod_{R_i}(r))$  and  $t$  is a tuple in  $D$ . Thus,

$$r \subseteq C_R(\prod_{R_i}(r)). \quad (6.24)$$

**Example 6.8:** Consider the relational schema  $R$  (Name, Job, salary) presented in table 6.9, and the projected relational schema  $R_1$  of Table 6.20.

The cylindrical extension of  $R_1$  on  $R$  is given in Table 6.11

**Table 6.11:** Cylindrical extension of  $R_1$  on  $R$

Name	Job	Salary	$\mu$
AA	Engineer	10k	0.8
BB	Engineer	10k	0.8
CC	Engineer	10k	0.8
DD	Engineer	10k	0.8
EE	Engineer	10k	0.8
AA	Manager	60k	0.9
BB	Manager	60k	0.9
CC	Manager	60k	0.9
DD	Manager	60k	0.9
EE	Manager	60k	0.9
AA	Secretary	40k	0.6
BB	Secretary	40k	0.6
CC	Secretary	40k	0.6
DD	Secretary	40k	0.6
EE	Secretary	40k	0.6

It can thus be easily shown that for any instance  $r_i$  of  $R_i$  and  $R \supseteq R_i$ ,

$$r_i = \prod_{R_i} (C_R ( r_i )) \quad (6.25)$$

### 6.5.3 Natural Join of Fuzzy Relations

Let  $\rho = | R_1, R_2, \dots, R_s |$  be a set of relational schema and  $R ( A_1, A_2, \dots, A_n ) = R_1 R_2 \dots R_s$ . Consider  $r_i, 1 \leq i \leq s$  be an instance of  $R_i$ . The natural join of  $r_1, r_2, \dots, r_s$ , denoted by

$$r = r_1 \blacktriangleright \blacktriangleleft r_2 \blacktriangleright \blacktriangleleft r_3 \blacktriangleright \blacktriangleleft \dots \blacktriangleright \blacktriangleleft r_s \quad (6.26)$$

is a fuzzy relation of the relational schema  $R$ . The membership function of  $r$  is given by

$$\mu_r ( a_1, a_2, \dots, a_n )$$

$$= \text{Min} [ \mu_{r_1}^c ( a_1, a_2, \dots, a_n ), \mu_{r_2}^c ( a_1, a_2, \dots, a_n ), \dots, \mu_{r_s}^c ( a_1, a_2, \dots, a_n ) ] \quad (6.27)$$

where  $a_i \in \text{dom}(A_i)$  and  $\mu_{r_j}^c$  is the membership of the cylindrical extension of  $r_j$  on  $R$  for  $i = 1, 2, \dots, n$  and  $j = 1, 2, \dots, s$ .

**Example 6.9:** Consider the join of relations  $r = \text{LIKES}$  (Student, Course) and  $r_1 = \text{TEACHING}$  (Teacher, Course). The membership function  $\mu_{r_1}$  may be interpreted as a measure of the teacher's ability to teach a course. Table 6.6(re-written) and 6.12 below are joined to get Table 6.13.

**Table 6.6: Likes (Students, Course)**

Name	Course	$\mu_r$
A. Konar	Algorithms	0.9
A. K. Nath	Communication	0.7
A. R. Saha	VLSI	0.8

**Table 6.12:** Teaching (Teacher, Course)

Teacher	Course	$\mu_r$
P. K. Das	Algorithms	0.75
A. Basak	Algorithms	0.85
A. Ghosh	Communication	0.95
B. Sen	VLSI	0.65

**Table 6.13:** Join of Likes and Teaching Relations

Students	Teacher	Course	$\mu_r$
A. Konar	P. K. Das	Algorithms	(0.75 $\wedge$ 0.9)
A. Konar	A. Basak	Algorithms	(0.8 $\wedge$ 0.9)
A. K. Nath	B. Ghosh	Communication	(0.95 $\wedge$ 0.7)
A. R. Saha	B. Sen	VLSI	(0.65 $\wedge$ 0.8)

#### 6.5.4 Lossy Fuzzy Joins

Let  $r_i$ , for  $i = 1, 2, \dots, s$  are obtained by taking projections of  $r$  on  $R_i$ ,  $i=1, 2, \dots, s$ . From (6.24) and (6.27) it is evident that

$$r \subseteq \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) \bowtie \dots \bowtie \Pi_{R_s}(r). \quad (6.28)$$

The above condition equally holds for classical relations and leads to the lossless join problem.

### 6.6 Fuzzy Integrity Constraints

Integrity constraints employed in relational database systems can broadly be classified into two common categories.

(i) *Domain dependency*: It restricts the domain of attributes. For example, in an employee relation the “age of an employee  $\leq 65$  years” or “height of a person  $\leq 10$  feet” are typical integrity constraints that restricts the domain of attributes age and height respectively.

(ii) *Data dependency*: It ensures that if some tuples in the databases fulfill certain equalities, then either some other tuples must exist in the database, or

some values of the given tuples are equal. Functional dependency, for example, is one such type of data dependency constraints.

To illustrate the scope of fuzziness in the domain of integrity constraints, we consider the following examples. In a PLAYERS (Name, Age, Height, Sport, Income) relational schema, we can introduce domain dependency integrity constraints like “many tennis players have high income” or “most basketball players are tall”. Similarly, in an EMPLOYEE (Name, Department, Job, Experience, Salary) relational schema, we may employ data dependency integrity constraints like “Employees having similar jobs and experience in any department have almost equal salary”.

In order to construct fuzzy integrity constraints, we should be equipped with appropriate tools. Zadeh [7] has introduced the concept of particularization of fuzzy relations. To formalize this issue, we discuss conjunction of fuzzy propositions.

### 6.6.1 Conjunction of Fuzzy Propositions

Let  $X = A_1 A_2 \dots A_m$  and  $Y = A_{i_1} A_{i_2} \dots A_{i_k}$  and  $Y \subseteq X$ .  $F$  is a fuzzy subset of domain  $D = \text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_m)$ , and  $G$  is a fuzzy subset of  $\text{dom}(A_{i_1}) \times \text{dom}(A_{i_2}) \times \dots \times \text{dom}(A_{i_k})$ . The conjunction of fuzzy propositions  $X$  is  $F$  and  $Y$  is  $G$  now is defined using the possibilistic distributions  $\Pi_x = F$  and  $\Pi_y = G$ . The particularization of  $\Pi_x$  by  $\Pi_y$  (or  $F$  by  $G$ ) is denoted by  $\Pi_x [\Pi_y = G]$  and is defined to be the intersection of  $F$  with cylindrical extension of  $G$ . Formally,

$$\Pi_x [\Pi_y = G] = F \cap G_c \quad (6.29)$$

where  $G_c$  is the cylindrical extension of  $G$  in  $\text{dom } D$ .

The particularization of a fuzzy relation  $r$  of  $R(A_1, A_2, \dots, A_m)$  due to a fuzzy proposition  $Y$  is  $G$ , where  $Y = A_{i_1} A_{i_2} \dots A_{i_k}$  produces a relation  $r_1$ , where

$$\mu_{r1}(a_1, a_2, \dots, a_m) = \min [\mu_r(a_1, a_2, \dots, a_m), \mu_{Gc}(a_1, a_2, \dots, a_m)] \quad (6.30)$$

where  $\mu_{Gc}(a_1, a_2, \dots, a_m)$  is the membership function of the cylindrical extension of  $G$ .

**Example 6.10:** Consider an fuzzy instance of students represented by the relational schema  $R$  (Name, Age, Grade). Suppose, we require that the relation

r must satisfy the domain dependency “ Students are Young” where YOUNG is the fuzzy set as defined below.

$$\text{YOUNG} = \{0.4 / 18 \text{ years}, 0.6 / 20 \text{ years}, 0.9 / 22 \text{ years}, 0.4 / 30 \text{ years}\}$$

The particularization of the fuzzy relation in Table 6.14 due to the above fuzzy proposition as an instance of YOUNG Students is presented in Table 6.15.

**Table 6.14:** The relational schema R (Name, Age, Grade)

Name	Age	Grade	$\mu$
AA	18	A	0.9
BB	20	B	0.8
CC	22	C	0.7

**Table 6.15:** Particularization of R by YOUNG

Name	Age	Grade	$\mu$
AA	18	A	(0.4 $\wedge$ 0.9)
BB	20	B	(0.6 $\wedge$ 0.8)
CC	22	C	(0.9 $\wedge$ 0.7)

## 6.6.2 The Modifier Rule

Consider a modified proposition X is  $\sigma$  F, where  $\sigma$  is a modifier, such as “not”, “very”, “more or less”. These modifiers are described by a mapping function

$$f\sigma : [0, 1] \rightarrow [0, 1].$$

$$\text{The possibility distribution } \Pi_x^\sigma \equiv F^\sigma \quad (6.31)$$

where  $F^\sigma$  is a fuzzy subset of U with membership function

$$\mu_{F^\sigma}(u) = f_\sigma(\mu_F(u)) \text{ for } u \in U \quad (6.32)$$

$$\text{For } \sigma = \text{not}, f_\sigma(X) = 1 - X \quad (6.33)$$

$$\sigma = \text{very}, f_\sigma(X) = X^2 \quad (6.34)$$

$$\text{and } \sigma = \text{more -or - less}, f_\sigma(x) = \sqrt{x}. \quad (6.35)$$

### 6.6.3 Possibility Distribution for Conditional Fuzzy Propositions

Let  $F$  and  $G$  be fuzzy sets of  $U$  and  $V$  respectively. The Possibility distribution  $\Pi(X \rightarrow Y)$  associated with the conditional fuzzy rule:

If  $X$  is  $f$  then  $Y$  is  $G$  given by

$$\Pi(X \rightarrow Y) = R \quad (6.36)$$

where  $R$  is a fuzzy subset of  $U \times V$  with membership

$$\begin{aligned} \mu_R(u, v) &= 1, \text{ if } \mu_F(u) \leq \mu_G(v) \\ &= 0, \text{ otherwise.} \end{aligned} \quad (6.37)$$

The rules discussed above can be applied to determine the possibility distribution of compound integrity constraints used in relational databases.

## 6.7 Fuzzy Functional Dependency

We remember that in a conventional relational schema  $R(A_1, A_2, \dots, A_n)$ , a functional dependency  $f: x \rightarrow y$  holds if for each pair of tuples  $t_1$  and  $t_2$  of  $r$ , when  $t_1[x] = t_2[x]$ , we have  $t_1[y] = t_2[y]$ . In a fuzzy relational model, equality of domain values defines a fuzzy proposition, and thus may be expressed as “more-or-less equal”, “appropriately equal” etc. To describe equality of domain values in fuzzy relational systems, we should briefly outline equality as a fuzzy relation.

### 6.7.1 Equality as a Fuzzy Relation

A fuzzy EQUAL (EQ) relation over a universe of discourse  $U$  is a fuzzy subset of  $U \times U$ , where  $\mu_{EQ}(a, b)$  for  $a, b \in U$  denotes the membership of equality of  $a$  and  $b$ .  $\mu_{EQ}(a, b)$  holds the reflexive and symmetric closure, i.e.,

$$\mu_{EQ}(a, a) = 1 \text{ (reflexive)}$$

$$\text{and } \mu_{EQ}(a, b) = \mu_{EQ}(b, a). \text{ (symmetric)}$$

It needs mention here that unlike classical equality, fuzzy equality relation is not transitive. So, fuzzy equality need not be a similarity relation.

The definition of fuzzy equality can be extended over composite domains as follows. Let R (A<sub>1</sub>, A<sub>2</sub>, ..., A<sub>n</sub>) be a relational schema having a domain D and tuples t<sub>1</sub> and t<sub>2</sub> where

$$D = \text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_n).$$

Then fuzzy equality relation extended over D defines a fuzzy subset D×D with membership function

$$\begin{aligned} \mu_{EQ}(t_1, t_2) = \text{Min} \{ & \mu_{EQ}^1(t_1[A_1], t_2[A_1]), \mu_{EQ}^2(t_1[A_2], t_2[A_2]), \\ & \dots, \mu_{EQ}^n(t_1[A_n], t_2[A_n]) \}, \end{aligned} \quad (6.38)$$

where j in  $\mu_{EQ}^j(t_1[A_j], t_2[A_j])$  denotes a specific parameter set for the fuzzy membership function  $\mu_{EQ}^j$ .

**Example 6.11:** Suppose we need to compare the equality of the tuples in an EMPLOYEE relation having attributes Name(N), Department(D), Job(J) Experience(X) and Salary(S).

Here, the fuzzy equality function will be of different types depending on the attributes. For example, for attributes like N, D and J we use the following equality relation

$$\mu_{EQ}(a, b) = 0 \text{ for } a \neq b, \text{ and}$$

$$\mu_{EQ}^1(a, a) = 1$$

where  $a, b \in \text{dom}(A)$  and  $A \in \{N, D, J\}$ .

But to compare the attributes X or S we may employ the following membership function.  $\mu_{EQ}(a, b) = 1/(1 + \beta |a - b|)$

where  $\beta = 1$ , for  $a, b \in \text{dom}(X)$

$$= 1/2000, \text{ for } a, b \in \text{dom}(S).$$

$$= 1/5000, \text{ for } a, b \in \text{dom}(I).$$

On occasion a may be crisp, while b may be fuzzy subset, both defined on the same domain. Under this case  $\mu_{EQ}(a, b) = \mu_b(a)$ .

Further, when a and b are both fuzzy subsets, then

$$\mu_{EQ}(a, b) = \text{Max} \{ c/\text{card}(a), c/\text{card}(b) \}$$

where “card” denotes cardinality of fuzzy sets and  $c = \text{card}(a \cap b)$ .

Fuzzy equality relations with modifiers can be described as follows.

$$\mu_{\sigma EQ}(a, b) = f_{\sigma}(\mu_{EQ}(a, b)) \quad (6.39)$$

As an example, suppose  $\sigma = \text{More-or-less}$

$$\mu_{MORE-OR-LESS EQUAL}(a, b) = (\mu_{\sigma EQ}(a, b)).$$

### 6.7.2 Representing Functional Dependency Using Fuzzy Equality Relation

Let  $X$  and  $Y$  be sets of attributes, such that  $X, Y \subseteq R$ , where  $R$  is a fuzzy relational schema. Let  $r$  be an instance of  $R$ . Then for any two tuples  $t_1$  and  $t_2$  in  $r$ , there exists a fuzzy functional dependency (ffd)  $X \rightarrow Y$  if

$$\mu_{EQ}(t_1[X], t_2[X]) \leq \mu_{EQ}(t_1[Y], t_2[Y]) \quad (6.40)$$

The above condition logically means that if the attribute set  $X$  of tuples  $t_1$  and  $t_2$  satisfy the fuzzy equality criterion, then the attributes set  $Y$  for  $t_1$  and  $t_2$  also satisfy the equality relation.

To illustrate the notion of  $\sigma$  in the construction of fuzzy integrity constraints, we consider the following example. Suppose, we need to denote “ $X$  more or less determines  $Y$ ” as an integrity constraint in a fuzzy relational database. The following membership function represents the above rule.

$$\mu_{EQ}(t_1[X], t_2[X]) \leq \mu_{EQ}(t_1[Y], t_2[Y]) \quad (6.41)$$

**Example 6.12:** Suppose that a SUPPLY(Item, Order-date, Delivery-date) relational schema satisfies an integrity constraint “order-date for any item more-or-less determines delivery- date”. Since items are crisp, we can use the following membership functions to test the equality item.

$$\mu_{EQ}(a, b) = 0, \text{ for } a \neq b; a, b \in \text{dom}(Item)$$

$$\mu_{EQ}(a, a) = 1, a \in \text{dom}(Item).$$

For order-date and delivery dates of item we may use the following membership distribution.

$$\mu_{EQ}(a, b) = 1 / (1 + |a - b|),$$

for  $a, b \in \text{dom}(\text{Order- date})$  or  $\text{dom}(\text{Delivery- date})$ .

Thus more-or-less equal delivery date can be described by  $\sqrt{(1 + |a - b|)}$ , where  $a$  and  $b$  are two delivery days. Consequently, the ffd that for equality in items, order- date determines the delivery date should be defined as follows.

If  $t_1[\text{Item}] = t_2[\text{Item}]$  then

$$1 / (1 + |t_1[\text{order- date}] - t_2[\text{order- date}]|)$$

$$\leq \sqrt{(1 / (1 + |t_1[\text{delivery- date}] - t_2[\text{delivery- date}]|))}.$$

## 6.8 Fuzzy Lossless Join

To answer a user's query, it is necessary to join two or more fuzzy relations. It is also clear from extension (6.28) that natural joins of fuzzy relations may not always recover the original relation. However, lossless join is an essential issue in relational database design. In this section, we examine the lossless join decomposition of fuzzy relations in presence of fuzzy functional dependencies.

Let  $R$  be a relational schema, and  $\rho = \{R_1, R_2, \dots, R_n\}$  be a decomposition of  $R$  with  $R = R_1 R_2 \dots R_n$ . The decomposition is a lossless join with respect to a set of fuzzy functional dependencies (ffds)  $F$ , if for every  $r$  of  $R$  that satisfies these ffds, the following condition holds.

$$r = \prod_{R1}(r) \bowtie \prod_{R2}(r) \bowtie \dots \bowtie \prod_{Rn}(r) \quad (6.42)$$

According to Raju and Majumder [4], a decomposition  $\rho$  of  $R$  is lossless join for a given set of integrity constraints, iff for any attributes set  $(a_1, a_2, \dots, a_n) \in \text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_n)$  there exist

$$r_i^c = C_R(\prod_{Ri}(r)) \text{ for some } i \in \{1, 2, \dots, n\} \text{ satisfying}$$

$$\mu_{ri}^c(a_1, a_2, \dots, a_n) = \mu_r(a_1, a_2, \dots, a_n) \quad (6.43)$$

**Example 6.13:** Let  $R(ABC)$  be a relational schema with  $\{A \rightarrow B, C \rightarrow B\}$  as the set of functional dependencies. Also let  $\rho = \{AB, BC\}$  be a decomposition of  $R$ . It is known that the above decomposition does not support lossless join. Suppose,  $r$  has two tuples  $t_1 = (a_1 b c_1)$  and  $t_2 = (a_2 b c_2)$ . Thus for  $t = (a_1 b c_2)$   $\mu_r(t) = 0$ , as it is not present in  $r$ . But  $r_1^c = C_R(\prod_{R1}(r))$  or  $r_2^c = C_R(\prod_{R2}(r))$

includes  $t = a_1bc_2$ , and thus  $\mu_{r^c}(t) = 1$  and  $\mu_{r^c}(t) = 1$  for  $t = a_1bc_2$ . So,  $\mu_{r^c}(a_1bc_2) \neq \mu_r(a_1bc_2)$ . Thus, the condition for lossless join (6.43) has a logical basis.

## 6.9 Design of Fuzzy Relational Databases

Like conventional relational database, fuzzy relation database too needs to satisfy the criterion of (fuzzy) lossless join and (fuzzy) functional dependency preservation. Raju and Majumder presented one technique [4] for testing the lossless join condition of a fuzzy relational system. Thus given a relational schema, we first construct EQUALITY relations to determine the fuzzy functional dependencies by satisfying criterion (6.40). Now, we construct a decomposition  $\rho = \{R_1, R_2, \dots, R_n\}$  of  $R$  such that functional dependencies are preserved. Next by Raju and Majumder's technique [4], we verify the lossless join decomposition criterion.

**Example 6.14:** Consider the relation ORDERS (S, I, Q, P, T) where S = Supplier, I = Item, Q= Quantity, P = Price and T= Sales Tax. To determine the list of functional dependencies, we design the following EQUALITY relations.

$$1. \mu_{EQ}(a, b) = 0, \text{ for } a \neq b \text{ and } a, b \in \text{dom}(S) \text{ or } \text{dom}(I).$$

$$2. \mu_{EQ}(a, b) = 1 / (1 + \beta |a-b|)$$

$$\begin{aligned} \text{where } \beta &= 1/100 \text{ for } a, b \in \text{dom}(Q) \\ &= 1/1000 \text{ for } a, b \in \text{dom}(P), \text{ and} \\ &= 1/50 \text{ for } a, b \in \text{dom}(T). \end{aligned}$$

Now, satisfying (6.40) we determine the following functional dependencies.

1. S and I more-or-less determine Q.
2. I and Q more-or-less determine P.
3. P more-or-less determines T.

So,  $\beta = \{R_1(SIQ), R_2(IQP), R_3(PT)\}$  be a decomposition of ORDER. Now, we verify following Raju and Majumder [4] that the above decomposition is lossless.

## 6.10 Conclusions

The chapter presented the fundamentals of fuzzy relational database design. It emphasized the need for EQUALITY relation to determine the fuzzy functional

dependencies and defines a criterion for a decomposition to satisfy the fuzzy lossless join criteria. The above criteria have been applied in designing fuzzy relational databases. The work presented in the chapter is useful to design relational databases suffering from uncertainty of information. It also demonstrates a new approach to construct and verify fuzzy constraints for noisy databases.

## Exercise

- Given the membership functions for moderate experience (MX) X and moderate salary (MS) S as follows:

$$\mu_{MX}(X) = 1 / [1 + |X - 6| / 16]$$

$$\text{and } \mu_{MS}(S) = 1 / [1 + |S - 50,000| / 20,000].$$

Determine the membership of A. Nath to have moderate experience and moderate salary.

[**Hints:** Compute  $\text{Min}[\mu_{MX}(9), \mu_{MS}(70,000)]$ . ]

- With respect to the Employee relation, shown in Table 6.8, determine the membership of A. B. Roy to have a moderate experience X of 9 years and a high salary S of US\$ 42K. Given

$$\mu_{MODERATE}(X) = 1 / [1 + |X - 8|]$$

$$\text{and } \mu_{HIGH}(S) = 1 / [1 + a |S - C|]$$

with  $a = 1/20,000$  and  $c = 60,000$ .

[**Hints:** Compute  $\text{Min} [\mu_t, \text{Min} \{\mu_{MODERATE}(9), \mu_{HIGH}(42,000)\}]$ , where  $\mu_t = 0.9$  from Table 6.8.]

- Given below a relational schema R for students' grade card.

R=	Name	Roll No.	Marks obtained in Physics (P)	Marks obtained in Maths. (M)
	AA	10	85	95
	BB	12	88	96

Using  $\mu_{EQ}(a, b) = 1 / [1 + |a - b|]$  for  $a, b \in \text{dom}(A)$  and  $A \in \{P, M\}$ , determine whether there exist any fuzzy functional dependency  $P \rightarrow M$ .

[**Hints:** Test whether

$$\mu_{EQ}(t_1[P], t_2[P]) \leq \mu_{EQ}(t_1[M], t_2[M]),$$

where  $t_1[P] = 85$ ,  $t_2[P] = 88$ ,  $t_1[M] = 95$  and  $t_2[M] = 96$ .]

4. Given below the relational schema  $R_i(Job, Salary)$  and  $R(Name, Job, Salary)$ , determine the cylindrical extension of  $r_i$  on  $R$ .

	Job	Salary	$\mu$
$R_i =$	Engineer	\$60,000	0.8
	Professor	\$40,000	0.6
	Manager	\$80,000	0.7

	Name	Job	Salary	$\mu$
$R =$	AA	Engineer	\$ 60,000	0.8
	BB	Engineer	\$ 50,000	0.2
	CC	Manager	\$ 80,000	0.7
	DD	Professor	\$ 40,000	0.6

**Answer:**  $C_R(\Pi_{Ri}(r)) =$

Name	Job	Salary	$\mu$
AA	Engineer	\$ 60,000	0.8
BB	Engineer	\$ 60,000	0.8
CC	Engineer	\$ 60,000	0.8
DD	Engineer	\$ 60,000	0.8
AA	Professor	\$ 40,000	0.6
BB	Professor	\$ 40,000	0.6
CC	Professor	\$ 40,000	0.6
DD	Professor	\$ 40,000	0.6
AA	Manager	\$ 80,000	0.7
BB	Manager	\$ 80,000	0.7
CC	Manager	\$ 80,000	0.7
DD	Manager	\$ 80,000	0.7

5. Verify that  $r \subseteq C_R(\Pi_{Job, Salary}(r))$  for the relational schema  $R(Name, Job, Salary)$  presented in Exercise 4.

## References

- [1] Buckles, B. P. and Petry, F. E., *Fuzzy Databases*, Kluwer Academic Press, Boston, 1998.
- [2] Kacprzyk. J. and Ziolkowski, A., "A database queries with fuzzy linguistic quantifiers," *IEEE Trans. on Systems, Man and Cybernetics*, SMC- 16, vol.3, pp. 474-479, May/June 1986.
- [3] Raju, K. V. S.V.N. and Majumdar, A. K., "The Study of joins in fuzzy relational databases," *Fuzzy Sets and Systems*, vol. 21, no.1, pp. 19-34, 1987.
- [4] Raju, K.V.S.V.N. and Majumdar, A. K., "Fuzzy functional dependencies and lossless join decomposition of fuzzy relational database system," *ACM Trans. on Database Systems*, vol.1, no.2, June 1988.
- [5] Ullman, J. D., *Principles of Database Systems*, Computer Science Press, Rockville, 1980.
- [6] Umano, M., Retrieval from fuzzy database by fuzzy relational algebra, In *Fuzzy Information, Knowledge Representation and Decision Analysis*, Sanchez, E. (Ed.), Pergamon press, Oxord, U.K., pp. 1-6, 1984.
- [7] Zadeh, L.A., "Fuzzy sets as a basis for theory of possibility," *Fuzzy Sets and Systems*, vol.1, no.1, pp. 3-28, 1978.

# 7

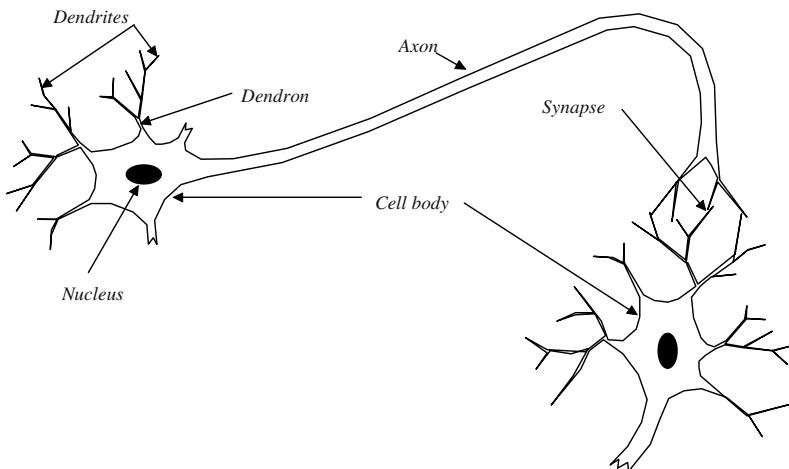
# Introduction to Machine Learning Using Neural Nets

*This chapter provides an introduction to machine learning using artificial neural networks. It reviews biological neural networks, and presents a general framework to construct their mathematical models with a view to study their applications in machine learning. The chapter overviews five different types of machine learning such as supervised learning, unsupervised learning, competitive learning, reinforcement learning and Hebbian learning. Stability and convergence are two fundamental issues in studying machine learning algorithms. The interrelationship between stability of a dynamical learning system and convergence of a learning algorithm is presented in detail in this chapter. Concluding remarks are appended at the end of the chapter.*

## 7.1 Biological Neural Networks

The human nervous system consists of small unicellular structures called neurons. Neurons thus are fundamental units or building blocks of a biological

nervous system. A neuron comprises of 5 elements: dendrite, dendron, cell body, synapse and axon (Fig. 7.1). The dendrites receive signals from other neurons, muscles or sensory organs and carry them to a relatively thick fiber called dendron. The dendrons carry the signals to the cell body, which in turn generates a composite signal based on the strength of the signals received from the dendrites. The composite signal is transmitted to the synapse through the axon (Fig. 7.1).



**Fig. 7.1:** A biological neural net comprising of two neurons, where the dendrites of the second neuron receive signal from the synapse of the first neuron.

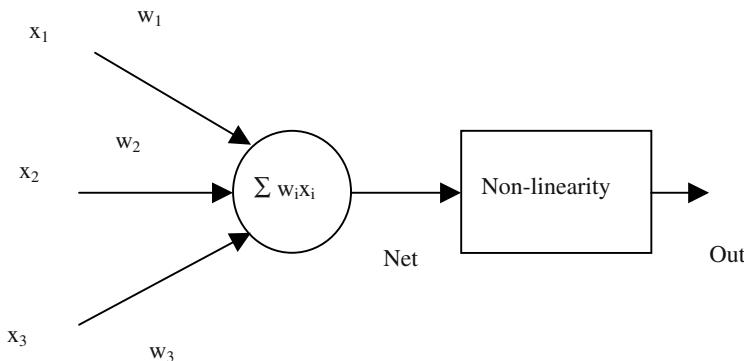
The synapse is like a potential barrier that controls the flow of signal from the axon of one neuron to the dendrites of other neurons. The transmission of signals from one neuron to another at a synapse is a complex chemical process. Generally, a specific chemical called neuro-transmitter is released from the transmitting end of the synapse. The transmitters lower or raise the potential barrier of the synapse. On lowering the potential barrier, the signal from the transmitting end can easily reach the other end of the synapse. On raising the barrier, a signal loss takes place and only a small component of the signal can reach the other end of the synapse. Flow of signal from one neuron to the others thus can be controlled by the synapse. When the influence of the synapse tends to activate the post-synaptic neuron, the synapse is called *excitatory*. On the other hand, when the synapse prohibits the passage of signal flow to the post-synaptic neuron, the synapse is called *inhibitory*. The synaptic ending of an axon thus is of excitatory or inhibitory nature.

How the neurons participate in the learning process of the human beings remained a mystery till this date. However, it is evident that the process of learning has a correspondence with the type and amount of neuro-transmitters

released at the pre-synaptic end of the neurons. Thus for similar sensory/control/motor actions a neuron releases the same type and amount of neuro-transmitters. How exactly the neurons perform the above task is an interesting topic of research for the biologists and medical researchers.

## 7.2 Artificial Neural Networks

Artificial neural networks are electrical analogue of the biological nervous system. A typical artificial neuron is mathematically represented by two modules: i) a linear activation/inhibition module and ii) a non-linearity that limits the signal levels within a finite band. Fig. 7.2 presents the typical organization of an artificial neuron.



**Fig. 7.2:** A typical artificial neuron.

The summer in Fig. 7.2 takes the role of the cell body and the inputs of the summer may be treated like dendrites. The synapse is modeled by a non-linear function and the connection from the summer to the non-linear unit is like the axon. Here, Net is a linear combiner of the inputs  $x_1, x_2, \dots, x_n$ . Mathematically,

$$\text{Net} = \sum_{i=1}^n w_i x_i, \quad (7.1)$$

where some of the inputs are excitatory (positive) and the rest are inhibitory (negative). ‘Out’ in the present context can take different mathematical forms. Some of the common forms are presented below.

$$\text{Out} = u(\text{Net} - \text{th}) \quad (7.2)$$

$$\text{Out} = \text{Sgn}(\text{Net}) \quad (7.3)$$

$$\text{Out} = 1 / [1 + \exp(-\text{Net})] \quad (7.4)$$

$$\text{Out} = \tanh(\text{Net}/2) \quad (7.5)$$

The mathematical form of Out can be smooth functions like sigmoid vide expression (7.4) or tanh vide expression (7.5) and sharp changing functions like step vide expression (7.2) or signum function (vide expression (7.3)).

The unit step function u in expression (7.2) is formally defined as follows:

$$\left. \begin{array}{l} u(\text{net} - \text{th}) \\ =1, \text{ Net} > \text{th} \\ =0, \text{ otherwise.} \end{array} \right\} \quad (7.6)$$

The signum function Sgn in expression (7.3) is formally defined as

$$\left. \begin{array}{l} \text{Sgn} (\text{Net}) \\ = +1, \text{ Net} > 0 \\ = -1, \text{ otherwise.} \end{array} \right\} \quad (7.7)$$

The definitions of sigmoid and tanh are very standard and thus need no further explanation.

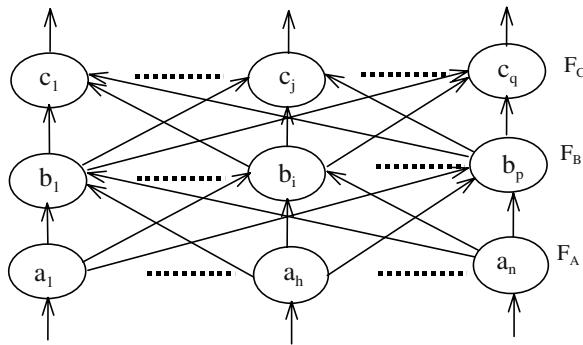
Neurons in an artificial neural net are connected in different topological configurations. Two most common type of configurations are i) feed-forward and ii) feedback topology. Usually, a feed-forward network contains a number of layers, each layer consisting of a number of neurons (Fig. 7.3). Signal propagation in such networks usually take place in the forward direction only, i.e., signals from the  $i$ -th layer can be propagated to any layer following the  $i$ -th layer, for  $i \geq 1$ . In a recurrent neural network, there exists feedback from one or more neurons to others. Fig. 7.4 describes a recurrent network.

The most important aspect of an artificial neural net is its capability of learning. In the next section, we introduce the concepts of learning on artificial neural nets.

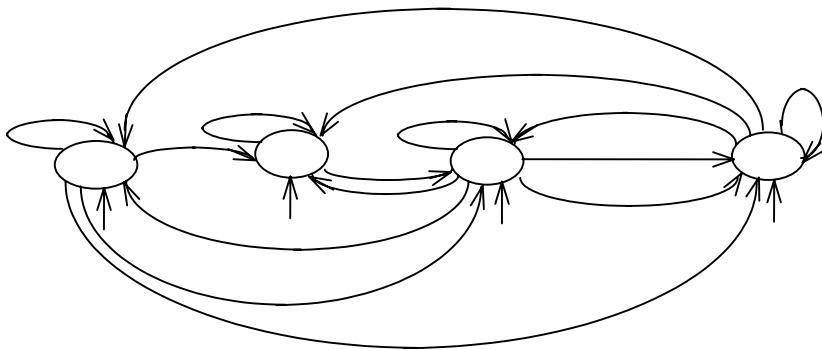
### 7.3 Principles of Learning in a Neural Net

Informally “encoding” or “learning” refers to adaptation of weights in a neural net. Thus until the weights converge to a steady state value, the process of encoding is continued. Adaptation of weights can be accomplished in a neural net by 4 different ways; they are supervised learning, unsupervised learning,

reinforcement learning and competitive learning. A brief outline to the learning schemes is presented below.



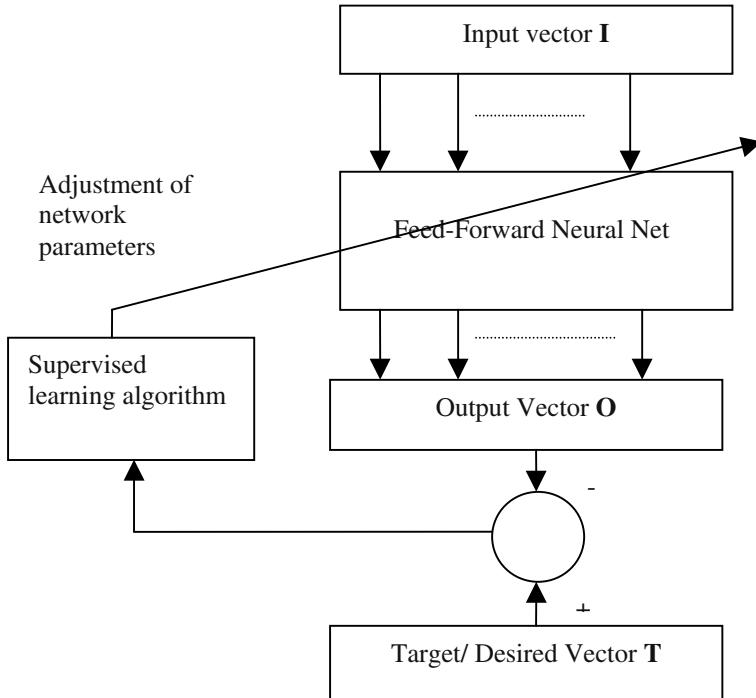
**Fig. 7.3:** A feed-forward neural net of 3 layers.



**Fig. 7.4:** A typical recurrent neural net.

◆ **Supervised Learning:** Supervised learning generally employs a trainer, who provides the input-output training instances of a given neural net. As an example, let us consider a pattern recognition problem, where we need to recognize an object from its feature-space. Here, the set of features such as size of the object and its shape described by its boundary descriptors, for instance, may be considered as input, while the type of the object such as books, pencils etc. may be treated as output of the neural net. Thus for  $n$  distinct objects, we require  $n$ -outputs of the neural net, each corresponding to one object. When one of  $n$ -outputs has the maximum value, the object is regarded to fall within the particular class. Further for a large  $n$ , we can

denote the output class by an encoded number, such as binary string. Thus for a given input feature vector, if a binary string 0011 appears at the output, we consider the object to belong to class 3.



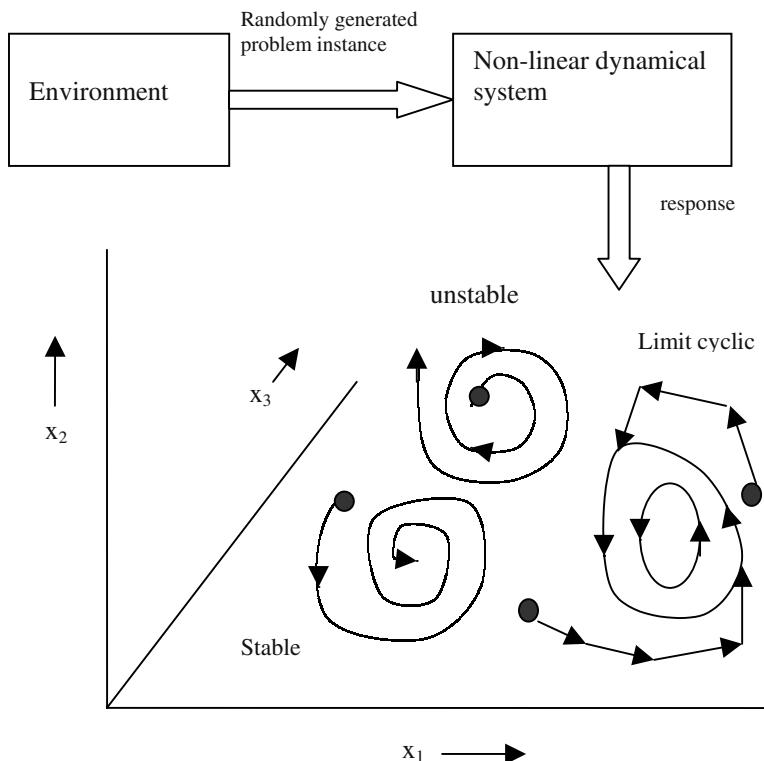
**Fig. 7.5:** A simple supervised learning scheme.

Fig. 7.5 describes a scheme for supervised learning. Here, given a input vector  $\mathbf{I}$  and a target vector  $\mathbf{T}$ , we need to fix the weights in the network, such that  $\mathbf{T}$  is produced at the output of the network when excited with the input  $\mathbf{I}$ . How can we achieve this? First, we initialize the weights randomly. Then for the given input vector  $\mathbf{I}$ , suppose the network generates the vector  $\mathbf{O}$  at its output. An error vector  $\mathbf{E} = \mathbf{T} - \mathbf{O}$  is then generated, and a supervised learning algorithm is used to adjust the Network parameters based on the error vector. A number of supervised learning algorithms that employ the above principle will be introduced in chapter 8.

◆ **Unsupervised Learning:** Unlike supervised learning, an unsupervised learning requires no teacher. Consequently, there is no target outputs. During the training phase, the neural net categorizes the received input

patterns into different classes. Thus when a new stimulus is applied in the application phase, the neural net responds indicating the class to which the stimulus belongs. In case, the stimulus does not fall in any of the known classes, a new class may be generated.

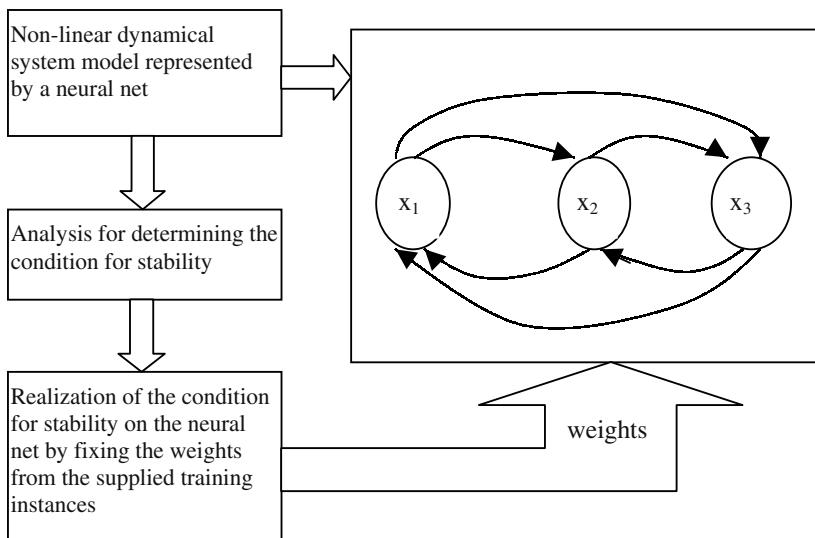
Though unsupervised learning requires no teacher, it requires some guidelines for class formation. Generally, classes are formed based on the features of the objects, such as color, shape, size etc. When guideline to preferences in feature selection for classification is not given, the classification is done based on the topology and network parameters. Consequently, such classification may not suit to selective applications. In order to utilize the power of classification of a neural net, the feature-selecting criteria are embedded in neural net design.



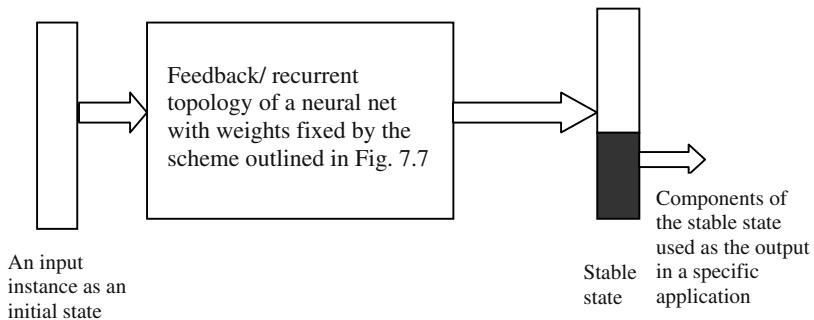
**Fig. 7.6:** Illustrating stable points, unstable points and limit cyclic behavior of a non-linear dynamical system in a 3-dimensional state space represented by  $x_1$ ,  $x_2$ ,  $x_3$ . The solid dots denote the initial position of the state trajectory; the arrows denote the direction of state transitions.

Most unsupervised learning networks have a dynamic behavior. Given an initial state of the neurons, an unsupervised neural net continues updating its system states until the network attains an *equilibrium state*. At the equilibrium state, the network remains stationary, i.e., it does not have any future change of states.

Stability analysis of non-linear dynamical system is an important issue for determining the parameters of the neural net. In Fig. 7.6, the environment supplies a random input instance to the non-linear dynamical system that yields a stable focus or unstable focus or limit cycles in its response. Analysis of stability is performed to determine the condition for stability of the system. Usually, the condition for stability describes a relationship between network stable states and parameters such as weights. Thus for a pre-defined set of stable states we can fix the parameters of the network (Fig. 7.7). In the application (recognition) phase, an unknown instance mapped at the network is defined as the initial state. The network is allowed to undergo state transition following a pre-defined dynamics. When the equilibrium state is reached, a few components of the equilibrium state vector may be used as the required output instance of the system (Fig. 7.8). A number of unsupervised learning algorithms will be introduced in chapter 9.



**Fig. 7.7:** Unsupervised learning on an artificial neural net with feedback.



**Fig. 7.8:** Illustrating the use of a recurrent neural net in the application phase. The network is supplied an input instance as its initial state vector. The network continues updating its state vector until a stable state is reached. A few components of the stable state vector is then used as the output for the given problem.

◆ **Competitive Learning:** Competitive and co-operative learning process are usually represented as artificial neural systems with self-exciting recurrent connections and neighbor inhibiting (competitive) or neighbor exciting (cooperative) connections [15]. A general form of competitive/cooperative learning is described by the following dynamics:

$$\frac{dx_i}{dt} = F_i(x_1, x_2, \dots, x_n), \quad (7.8)$$

for  $i = 1, 2, \dots, n$ .

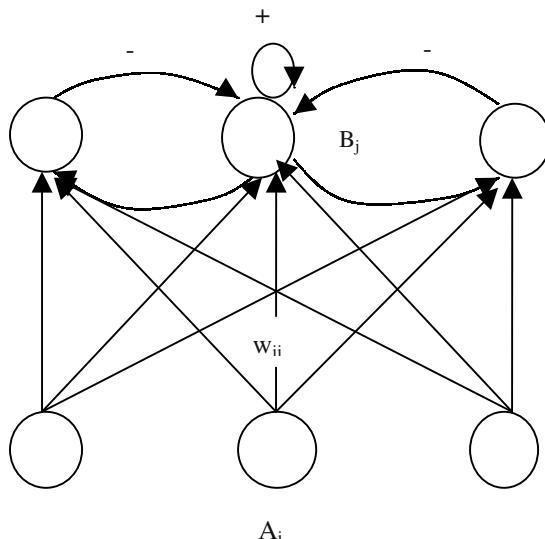
The dynamics is called competitive if

$$\left. \begin{array}{l} \frac{\partial F_i}{\partial x_i} \geq 0 \\ \frac{\partial F_i}{\partial x_j} \leq 0 \text{ for } j \neq i. \end{array} \right\} \quad (7.9)$$

and

On the other hand, the dynamics is called cooperative if

$$\begin{aligned}
 & \frac{\partial F_i}{\partial x_i} \geq 0 \\
 & \text{and} \quad \frac{\partial F_i}{\partial x_j} \geq 0 \text{ for } j \neq i.
 \end{aligned}
 \quad \left. \right\} \quad (7.10)$$



**Fig. 7.9:** A competitive learning neural network.

A competitive learning network generally consists of two layers: the input layer and the competitive/output layer. The input vector, submitted at the input layer, is passed on to the competitive layer through a set of forward connection of weights. The neurons in the competitive layer compete with others by generating a positive signal to itself and a negative signal to the other neurons. Once the competition is complete, the connection weights of the winner neuron  $j$  from the neuron  $i$  of the input layer is governed by the following weight adaptation dynamics:

$$\frac{dw_{ij}}{dt} = S(a_i) [-w_{ij} + S(b_j)] \quad (7.11)$$

where

$a_i$  is the signal strength of the neuron  $A_i$  in the input layer,

$b_j$  is the signal strength of the neuron  $B_j$  in the competitive layer,

$S$  is a Sigmoid type non-linearity over  $a_i$ ,

$w_{ij}$  is the connection weight from neuron  $A_i$  of the first layer to neuron  $B_j$  in the competitive layer.

A typical diagram of a competitive learning network is given in Fig. 7.9.

Until now A number of competitive/ cooperative learning models have been proposed. Among the significant contributions, the works by Grossberg [4-5], Amari [1], Amari and Arbib [2], Takuchi and Amari [14], and Rumelhart and Zibser [12] need special mention. The details of competitive learning will be undertaken in a separate chapter in the latter part of the book.

♦**Reinforcement learning:** In reinforcement learning, weights are reinforced for properly performed actions and penalized for poorly performed actions. A general form of reinforcement learning equation is

$$\Delta w_{ij} = \alpha [r - \theta_i] e_{ij} \quad (7.12)$$

where

$r$  = a scalar success/ failure index provided by the environment,

$\theta_i$  = reinforcement threshold value for the  $i$ -th neuron,

$e_{ij}$  = canonical eligibility of weight from neuron  $j$  to neuron  $i$ ,

$w_{ij}$  = connectivity strength from neuron  $j$  to neuron  $i$ , and

$\alpha$  = learning rate, the typical value of which lie in  $(0, 1)$ .

For determining  $e_{ij}$ , we should have some knowledge about the probability distribution that the computed output  $b_i$  at neuron  $i$  is equal to the desired output  $d_i$  at the same neuron.

Let

$$g_i = P\{b_i = d_i \mid \text{given } w_i \text{ and } A\} \quad (7.13)$$

where

$w_i$  is the weight vector associated with neuron  $i$ , and

$A$  is the input vector to the  $i$ -th neuron.

Then  $e_{ij}$  is defined as

$$e_{ij} = \frac{\partial \ln(g_i)}{\partial w_{ij}} \quad (7.14)$$

Besides the above 4 common types of neural learning, one most interesting learning paradigm that needs special mention is the well-known *Hebbian Learning*. A brief outline to this learning principle is presented below.

*"When a neuron  $a_i$  is near enough to excite another neuron  $a_j$ , and repeatedly or persistently participates in firing it, some growth process or metabolic changes takes place in one or both neurons, such that efficiency of  $a_i$  in firing  $a_j$  increases [6]."*

Let  $w_{ij}$  be the connectivity strength from neuron j to neuron i. Then the simplest Hebbian learning rules is given by

$$\Delta w_{ij} = a_i \cdot a_j \quad (7.15)$$

where  $a_i$  and  $a_j$  denote the respective signal i and j respectively. There are several modifications to the basic Hebbian learning, these are briefly outlined below:

#### Sejnowski's model [13]

$$\Delta w_{ij} = \eta (a_i - a_i') (a_j - a_j') \quad (7.16)$$

where

$a_k'$  for  $k \in (i, j)$  denotes the average value of neuron  $a_k$ , and  
 $\eta$  is the learning rate in  $(0, 1)$ .

#### Klopf's model [9]

$$\Delta w_{ij} = (\Delta a_i) (\Delta a_j) \quad (7.17)$$

Usually, a decay term is added to the basic Hebbian learning model of (7.15), and thus the revised model becomes

$$\Delta w_{ij} = -w_{ij}(t) + a_i(t) a_j(t), \quad (7.18)$$

the continuos version of which is very popular, and is given by

$$\frac{dw_{ij}}{dt} = -w_{ij}(t) + a_i(t) \cdot a_j(t) \quad (7.19)$$

where the argument t of  $w_{ij}$ ,  $a_i$  and  $a_j$  denotes time or iteration in a program.

Grossberg [4-5], Hopfield [8] and others extended the model (7.19) further by imposing non-linear threshold function. The modified learning law, popularly called *signal Hebbian learning law* is presented in (7.20).

$$\frac{dw_{ij}}{dt} = -w_{ij}(t) + S(a_i(t)) \cdot S(a_j(t)) \quad (7.20)$$

where  $S$  is a sigmoid type non-linear inhibiting function.

Kosko [10-11] proposed a *differential Hebbian Learning* rule by replacing  $S(a_i)$  and  $s(a_j)$  by their time- derivatives. Thus,

$$\frac{dw_{ij}}{dt} = -w_{ij}(t) + \frac{ds(a_i)}{dt} \cdot \frac{ds(a_j)}{dt} \quad (7.21)$$

where

$$\frac{ds(a_k)}{dt} = \left( \frac{\partial s}{\partial a_k} \right) \left( \frac{da_k}{dt} \right), k \in (i, j) \quad (7.22)$$

A thorough examination of different Hebbian Learning rules is available in Barto [3].

## 7.4 Stability and Convergence

The notion of “ stability” is commonly used in a cybernetic system to study the dynamic equilibrium of the system. A neural net being a cybernetic system needs a thorough analysis of the stability of its dynamics. In fact most of the unsupervised neural models such as Hopfield nets are represented by differential/ difference equations. An energy function  $E$  of neuronal weights  $w_1, w_2, \dots, w_n$  is then employed to study the dynamic behavior of the system states. The problem in the present context is to minimize the Energy function of weights. An exhaustive search in the weight – space can determine the minima of the energy function. These minima are referred to as *stable points*. Thus finding the minima of the energy function indirectly corresponds to determining the stable points of the dynamics.

The energy function used to analyze the stability of an unsupervised learning system is popularly known as *Lyapunov energy function*. For

continuous energy function  $E$ , the condition of stability calls for checking whether  $dE/dt$  is negative. For a discrete system, we on the other hand, need to check whether the change in energy  $\Delta E$  is negative. When the energy function of weights is absolutely specified, we can determine the stable/ fixed points of the systems by setting either of the above conditions.

The notion of convergence on the other hand is the eventual minimization of error between the desired and computed output of selected neurons. Convergence of error is a prime consideration in supervised learning algorithms. Most supervised learning algorithms, such as back-propagation or Widrow-Hoff's ADALINES to be introduced in the next chapter, have been designed to adapt the weights in a manner so as to gradually reduce the error.

## 7.5 Three Important Theorems for Stability Analysis of Neural Dynamics

Three general theorems for stability analysis of neural dynamics are presented in this section. The theorems provide guidelines in the selection of Lyapunov Energy function for typical non-linear dynamics. They are popularly known as Cohen-Grossberg theorem, Cohen-Grossberg-Kosko theorem and Adaptive bi-directional associative memory theorem.

**Cohen-Grossberg Theorem:** *A non-linear dynamical system given by*

$$\frac{dx_i}{dt} = a_i(x_i)[b_i(x_i) - \sum_{k=1}^N w_{ik} d_k(x_k)], \quad (7.23)$$

for  $i = 1, 2, \dots, N$ .

The global stability of the system can be determined by the following Lyapunov energy function:

$$V(x) = - \sum_{i=1}^N \int_0^{x_i} b_i(\xi_i) \left( \frac{dd_i(\xi_i)}{dt} \right) d\xi_i + \frac{1}{2} \sum_{i=1}^N \sum_{k=1}^N w_{ik} d_i(x_i) d_k(x_i). \quad (7.24)$$

*Proof:* We compute

$$\begin{aligned}
 \frac{dV}{dt} &= \frac{dV(\mathbf{x}(t))}{dt} \\
 &= \sum_{i=1}^N \left[ \frac{\partial V}{\partial x_i} \right] \left[ \frac{\partial x_i}{\partial t} \right] \\
 &= - \sum_{i=1}^N b_i(x_i) \left[ \frac{dd_i(x_i)}{dt} \right] \left[ \frac{dx_i}{dt} \right] \\
 &\quad + \sum_{i=1}^N \sum_{k=1}^N w_{ik} \left[ \frac{dd_i(x_i)}{dt} \right] d_k(x_k) \left[ \frac{dx_k}{dt} \right] \\
 &= - \sum_{i=1}^N \left[ \frac{dx_i}{dt} \right] \left[ \frac{dd_i(x_i)}{dt} \right] \left\{ b_i(x_i) - \sum_{k=1}^N w_{ik} d_k(x_k) \right\} \tag{7.25}
 \end{aligned}$$

$$= - \sum_{i=1}^N a_i(x_i) \left[ \frac{dd_i(x_i)}{dt} \right] \left\{ b_i(x_i) - \sum_{i=1}^N w_{ik} d_k(x_k) \right\}^2 \tag{7.26}$$

$$< 0,$$

$$\text{if } a_i(x_i) \geq 0 \text{ and } \frac{dd_i(x_i)}{dt} \geq 0 \tag{7.27}$$

Thus, the condition for stability of the given system is

- i)  $a_i(x_i)$  is non-negative,

- ii)  $d_i(x_i)$  is monotonically non-decreasing.

**Cohen-Grossberg-Kosko theorem:** A dynamic system given by

$$\frac{dx_i}{dt} = a_i(x_i)[b_i(x_i) - \sum_{k=1}^N w_{ik} d_k(x_k)], \quad (7.28)$$

for  $i = 1, 2, \dots, N$ ,

and  $\frac{dw_{ik}}{dt} = -w_{ik} + d_i(x_i) d_k(x_k)$  (7.29)

is stable and  $V(x)$  defined below is its Lyapunov energy function.

$$V(x) = - \sum_{i=1}^N \int_0^{x_i} b_i(\xi_i) \left( \frac{dd_i(\xi_i)}{dt} \right) d\xi_i$$

$$+ \frac{1}{2} \sum_{i=1}^N \sum_{k=1}^N w_{ik} d_i(x_i) d_k(x_k) - \frac{1}{4} \sum_{i=1}^N \sum_{k=1}^N w_{ik}^2 \quad (7.30)$$

*Proof:* Proof of the theorem can be done by the same approach undertaken for the proof of the previous theorem.

**Adaptive Bi-directional Associative Memory Theorem:** A Hebbian type dynamical system given by

$$\frac{dx_i}{dt} = -a_i(x_i)[b_i(x_i) - \sum_j w_{ij} d_j(y_j)], \quad (7.31)$$

$$\frac{dy_j}{dt} = a_j(y_j)[b_j(y_j) - \sum_i w_{ji} d_i(x_i)], \quad (7.32)$$

$$\text{and } \frac{dw_{ij}}{dt} = -w_{ij} + d_i(x_i) d_j(y_j) \quad (7.33)$$

has a Lyapunov energy function of the form:

$$V(x, y) = \sum_i \int_0^{x_i} b_i(\xi_i) \frac{dd_i(\xi_i)}{dt} d\xi_i + \sum_j \int_0^{y_j} b_j(\xi_j) \frac{dd_j(\xi_j)}{dt} d\xi_j - \frac{1}{2} \sum_i \sum_j w_{ij}^2 d_i(x_i) d_j(y_j) \quad (7.34)$$

and the system is stable.

*Proof:* Proof of the theorem is similar with the proof of the Cohen-Grossberg theorem.

## 7.6 Conclusions

The chapter outlined 4 different type of learning algorithms. It stressed the need for convergence analysis of a supervised learning algorithm and stability analysis of an unsupervised learning dynamics. The convergence analysis is usually accomplished by showing that the difference between the target vector and the computed output vector diminishes with increase in learning epochs. The stability analysis of an unsupervised learning dynamics [7] is generally performed by examining the sign of  $dV/dt$ , where  $V$  denotes the Lyapunov energy function. Construction of a Lyapunov energy function for a given dynamics itself is a complex problem. Three popular Lyapunov energy functions, introduced in the chapter are commonly used to analyze the stability of most of the unsupervised learning dynamics.

## Exercise

1. Given the following dynamical system:

$$\frac{dw_{ij}}{dt} = S(a_j) [-w_{ij} + S(b_i)]$$

$= F_i(a_j, b_i)$ , say.

Determine the condition for competitive learning of the above dynamics.

**[Hints:** By definition of competitive learning we need to satisfy:

$$\frac{\partial F_i}{\partial a_j} = \frac{\partial F_i}{\partial S} - \frac{\partial S}{\partial a_j}$$

$$= -w_{ij} S'(a_j) < 0 \text{ if } S'(a_j) > 0$$

$$\text{Further, } \frac{\partial F_i}{\partial b_i} = \frac{\partial F_i}{\partial S(b_i)} - \frac{\partial S(b_i)}{\partial b_i}$$

$$= S(a_j) S'(b_i) > 0$$

which finally requires  $S(a_j) > 0$  and  $S(a_j)$  and  $S(b_i)$  to be monotonically increasing functions.]

2. The dynamics of a neuron is given by

$$\frac{dx_i(t)}{dt} = -A_i x_i(t) + B_i I_i$$

where  $A_i$  and  $B_i$  are constants and  $I_i$  is an external input to node  $I_i$ . Compute  $x_i(t)$  by directly solving the differential equation.

**[Hints:** The given differential equation can be written as

$$(D + A_i)x_i = B_i I_i$$

The complementary function of the above equation is given by  $C \exp(-A_i t)$ , where  $C$  is a constant.

The particular integral is given by

$$x_i = \frac{B_i I_i}{D + A_i}$$

$$= \frac{B_i}{A_i} I_i$$

$$\text{Thus, } x_i(t) = C \exp(-A_i t) + \frac{B_i}{A_i} I_i.$$

At  $t=0$ ,  $x_i(t) = x_i(0)$ , consequently from the last expression we find:

$$C = x_i(0) - \frac{B_i}{A_i} I_i$$

Substituting C in the expression of  $x_i$  we finally have:

$$x_i(t) = x_i(0) \exp(-A_i t) + \frac{B_i}{A_i} I_i \{1 - \exp(-A_i t)\}.$$

$$3. \text{ Given } \frac{dx_i}{dt} = -x_i \text{ for } i=1 \text{ to } n.$$

Show by a Lyapunov energy function that the dynamics is stable.

**[Hints:** Let

$$V(x_1, x_2, \dots, x_n) = x_1^2 + x_2^2 + \dots + x_n^2$$

be the selected Lyapunov energy function.

Then,

$$\begin{aligned} \frac{dV}{dt} &= \frac{\partial V}{\partial x_1} \frac{dx_1}{dt} + \frac{\partial V}{\partial x_2} \frac{dx_2}{dt} + \dots + \frac{\partial V}{\partial x_n} \frac{dx_n}{dt} \\ &= 2x_1(-x_1) + 2x_2(-x_2) + \dots + 2x_n(-x_n) \end{aligned}$$

$$N = -2 \sum_{i=1}^n x_i^2$$

$$< 0.$$

Since  $dV/dt$  is negative, the given dynamics is stable.]

4. Suppose, we have a system with the same dynamics given in exercise 2. Assuming  $x_i > 2 B_i I_i / A_i$ , determine the Lyapunov energy function for the dynamics. Show that the dynamics is stable by Lyapunov's  $dV/dt < 0$  criterion.

[**Hints:** Let

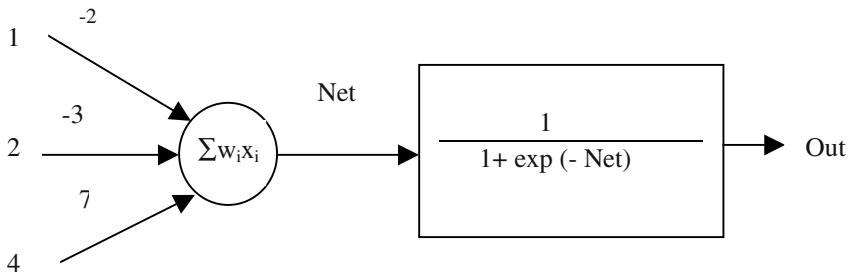
$$V(x_1, x_2, \dots, x_n)$$

$$= - \sum_{i=1}^n (-A_i x_i^2 / 2 + B_i I_i x_i)$$

Here,  $V(0, 0, \dots, 0) = 0$ ; and as  $x_i > 2 B_i I_i / A_i$ ,  $V(x_1, x_2, \dots, x_n) > 0$  for any  $(x_1, x_2, \dots, x_n) \neq (0, 0, \dots, 0)$ . Further,  $\partial V / \partial x_i$  exists for all  $i = 1$  to  $n$ . Thus  $V$  is a Lyapunov function for the proposed dynamics.

$$\begin{aligned} \frac{dV}{dt} &= \sum_{i=1}^n \frac{\partial V}{\partial x_i} \frac{dx_i}{dt} \\ &= - \sum_{i=1}^n (-A_i x_i + B_i I_i)^2 \\ &< 0. ] \end{aligned}$$

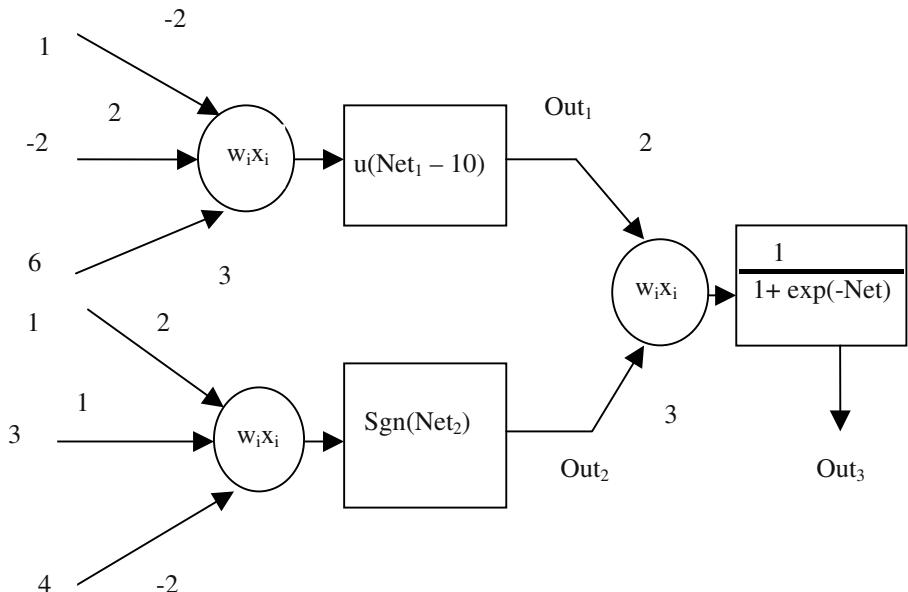
5. Determine Net and Out for neuron shown in Fig 7.10.



**Fig. 7.10:** An illustrative neuron.

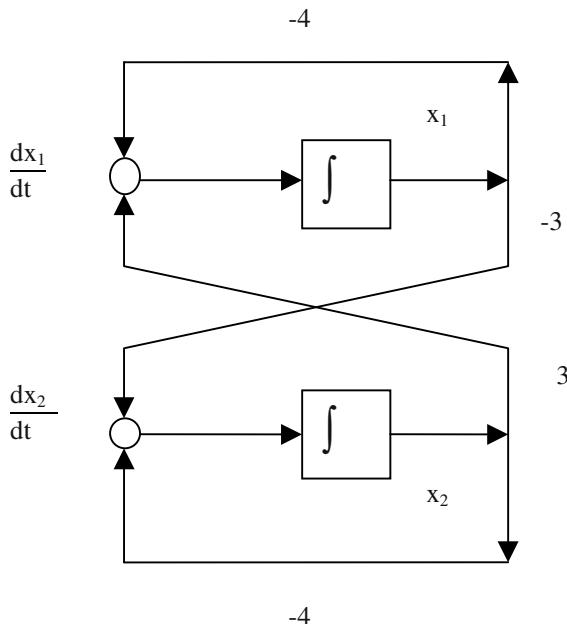
[Answer: Net= 22 units, Out= 0.999 units]

6. Compute the outputs  $\text{Out}_1$ ,  $\text{Out}_2$  and  $\text{Out}_3$  of the neurons in **Fig. 7.11**.



**Fig. 7.11:** One neural net comprising of 3 neurons.

[Answer:  $\text{Out}_1 = 1$  unit,  $\text{Out}_2 = -1$  unit and  $\text{Out}_3 = 0.303$  units]



**Fig. 7.12:** An electrical recurrent neural network.

- 7a) Write the state equations for the electrical neural network shown in **Fig. 7.12**  
 b) Judge the stability of the neural net by a suitable Lyapunov energy function.

[**Hints:** a) The state equations are given by

$$\frac{dx_1}{dt} = -4x_1 + 3x_2$$

$$\text{and } \frac{dx_2}{dt} = -3x_1 - 4x_2.$$

- b) Let  $V(x_1, x_2) = x_1^2 + x_2^2$  be the Lyapunov energy function for the given dynamics. Then

$$\frac{dV}{dt} = \frac{\partial V}{\partial x_1} \frac{dx_1}{dt} + \frac{\partial V}{\partial x_2} \frac{dx_2}{dt}$$

$$= 2x_1(-4x_1 + 3x_2) + 2x_2(-3x_1 - 4x_2)$$

$$= -8(x_1^2 + x_2^2)$$

$$< 0.$$

Hence, the dynamics is stable. ]

7. The weight adaptation policy in a Back-propagation neural net with momentum term is given below:

$$\Delta w(k) = \eta \frac{\partial E}{\partial w} + \beta \Delta w(k-1)$$

where  $E$  denotes an error surface of weights  $w$ ,  $\eta$  and  $\beta$  are two constants,  $w(k)$  denotes a weight at iteration  $k$ , and  $\Delta w$  denotes change in weight.

- a) Determine the transfer function with  $\Delta w(k)$  as the output and  $\partial E / \partial w$  as the input.
- b) Also determine the condition for stability of the system.

[**Hints:** (a) The given dynamics can be represented as

$$\Delta w(k) - \beta \Delta w(k-1) = \eta \frac{\partial E}{\partial w}$$

$$\text{or, } (1 - \beta z^{-1}) \Delta w(k) = \eta \frac{\partial E}{\partial w}$$

$$\text{or, } \frac{\Delta w(k)}{\partial E / \partial w} = \frac{\eta}{1 - \beta z^{-1}}.$$

- (b) For stability we require that the poles should lie inside  $|z| = 1$  circle. Thus,

$$|z - \beta| < 1,$$

which finally yields  $0 < \beta < 1$ .]

8. Linearize the following dynamics around the equilibrium point  $\mathbf{X}^* = [0 \ 1]^T$ . Check whether the Jacobian matrix yields all negative eigenvalues, and hence comment on the stability of the dynamics.

$$\frac{dx_1}{dt} = -x_1 + (x_1 + x_2 - 1)^2$$

$$\text{and } \frac{dx_2}{dt} = x_1 - 2x_2 + 2 + x_1^3$$

[Hints: Let  $x_1 = x_1^* + p$ , and  $y_1 = y_1^* + q$ . Given

Also let  $f_1(x, y) = -x_1 + (x_1 + x_2 - 1)^2$  and

$$f_2(x, y) = x_1 - 2x_2 + 2 + x_1^3.$$

$$\text{Therefore, } \frac{dx_1}{dt} = \frac{d(x_1^* + p)}{dt} = \frac{dp}{dt},$$

$$\text{and } \frac{dx_2}{dt} = \frac{d(x_2^* + q)}{dt} = \frac{dq}{dt}.$$

Further,  $f_1(x_1, x_2) = f_1(x_1^* + p, x_2^* + q)$

$$= f_1(x_1^*, x_2^*) + \left( \frac{\partial f_1}{\partial x_1} \right)_{\substack{x_1=x_1^* \\ x_2=x_2^*}} \cdot p + \left( \frac{\partial f_1}{\partial x_2} \right)_{\substack{x_1=x_1^* \\ x_2=x_2^*}} \cdot q$$

$$= 0 + (-1)p + (0)q.$$

Similarly,  $f_2(x, y) = f_2(x_1^* + p, x_2^* + q)$

$$= f_2(x_1^*, x_2^*) + \left( \frac{\partial f_2}{\partial x_1} \right)_{\substack{x_1=x_1^* \\ x_2=x_2^*}} \cdot p + \left( \frac{\partial f_2}{\partial x_2} \right)_{\substack{x_1=x_1^* \\ x_2=x_2^*}} \cdot q$$

$$= 0 + (1)p + (-2)q.$$

Consequently, the linearized dynamics is given by

$$\frac{dx_1}{dt} = \frac{dp}{dt} = f_1(x_1, x_2) = f_1(x_1^* + p, x_2^* + q) = -p, \text{ and}$$

$$\frac{dx_2}{dt} = \frac{dq}{dt} = f_2(x_1, x_2) = f_2(x_1^* + p, x_2^* + q) = p - 2q.$$

In matrix-vector form, we re-write the last two equations by

$$\begin{pmatrix} \frac{dp}{dt} \\ \frac{dq}{dt} \end{pmatrix} = \begin{pmatrix} -1 & 0 \\ 1 & -2 \end{pmatrix} \begin{pmatrix} p \\ q \end{pmatrix}.$$

$= \mathbf{J} [p \quad q]^T$  say, where  $\mathbf{J}$  is the Jacobian at the equilibrium point.

The above expression describes the linearized version of the given dynamics.

To study the stability, we set  $\text{Det}(\mathbf{J} - \lambda \mathbf{I}) = 0$ , which yields:

$$\begin{vmatrix} (-1 - \lambda) & 0 \\ 1 & (-2 - \lambda) \end{vmatrix} = 0,$$

or,  $(-1 - \lambda)(-2 - \lambda) = 0$ ,

or,  $\lambda = -1$  or  $-2$ .

Since both the eigenvalues are negative, the system is asymptotically stable.]

9. The linearization method for studying the stability of a non-linear dynamics at a given equilibrium point may fail when the linearized dynamics is stable but not asymptotically stable, i.e., if all the eigenvalues of the Jacobian have non-positive real parts, and if the real part of one or more eigenvalues of the Jacobian is zero. Demonstrate the above fact for the following non-linear system at the equilibrium point  $X^* = [0 \quad 0]^T$ :

$$\frac{dx_1}{dt} = -x_2 + ax_1(x_1^2 + x_2^2), \text{ and}$$

$$\frac{dx_2}{dt} = x_1 + ax_2(x_1^2 + x_2^2).$$

[**Hints:** The Jacobian  $J$  is obtained as follows:

$$J = \begin{pmatrix} 3ax_1^2 + ax_2^2 & -1+2ax_1x_2 \\ 1+2ax_1x_2 & ax_1^2 + 3ax_2^2 \end{pmatrix}$$

At the equilibrium point  $[x_1^* \ x_2^*]^T = [0 \ 0]^T$ , the Jacobian is found to be

$$J = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$

Setting  $\text{Det}(J - \lambda I) = 0$ , we obtain the eigenvalues  $\lambda = +j$  or  $-j$ . Thus the eigenvalues have zero real part.

Assuming  $x_1 = x_1^* + p$  and  $x_2 = x_2^* + q$ , it can be shown that the linearized dynamics is of the form:

$$dp/dt = -q \text{ and } dq/dt = p.$$

Combining the above equations we obtain:  $d^2q/dt^2 = -q$ , which is the dynamics of a simple harmonic motion and thus the dynamics has small oscillations around the origin. Consequently the stability of the system cannot be proved as the eigenvalues have zero real part.]

10. The Hessian matrix  $H$  is a symmetric matrix, defined as follows:

$$\mathbf{H} = \left( \frac{\partial^2 J}{\partial x_i \partial x_j} \right)$$

where  $J$  is the Jacobian of a linearized dynamics. A dynamics is asymptotically stable if all the eigenvalues of the symmetric matrix  $\mathbf{H}$  have non-zero positive real parts. Determine Hessian matrix for the dynamics given in problem 10, and show that stability of the dynamics depends on the parameter  $a$ .

**[Hints:** For the dynamics given in problem 10, we obtain:

$$J = \begin{pmatrix} 3ax_1^2 + ax_2^2 & -1 + 2ax_1x_2 \\ 1 + 2ax_1x_2 & ax_1^2 + 3ax_2^2 \end{pmatrix}$$

$$H = \begin{pmatrix} \frac{\partial^2 J}{\partial x_1^2} & \frac{\partial^2 J}{\partial x_1 \partial x_2} \\ \frac{\partial^2 J}{\partial x_2 \partial x_1} & \frac{\partial^2 J}{\partial x_2^2} \end{pmatrix}$$

$$= \begin{pmatrix} 6a & 2a \\ 2a & 6a \end{pmatrix}$$

It is indeed important to note that  $\mathbf{H}$  is only a function of  $a$  and is independent of the equilibrium point  $[x_1^* \ x_2^*]^T = [0 \ 0]^T$ . The eigenvalues of  $H$  are obtained by setting  $[\mathbf{H} - \lambda I] = 0$ , which yields  $\lambda = 4a$  and  $6a$ . Thus the eigenvalues will be positive as  $a > 0$ , and consequently the system will be stable as  $a > 0$ .]

## References

- [1] Amari, S.-I., "Field theory of self-organizing neural nets," *IEEE Trans. on Systems, Man and Cybernetics*, SMC-13, pp. 741-748, 1983.
- [2] Amari, S. and Arbib, M. A., "Competition and cooperation in neural nets," In *Systems Neuroscience*, Metzler, J. (Ed.), Academic Press, NY, pp. 119-165, 1977.
- [3] Barto, A. G., "Reinforcement learning and adaptive critic method," In *Handbook of Intelligent Control*, White, D. A. and Sofge, D. A. (Eds.), Van Nostrand Reinhold, NY, pp. 469-491, 1992.
- [4] Grossberg, S., "Adaptive pattern classification and universal recording: I Parallel development and coding of neural feature detectors," *Biological Cybernetics*, vol. 23, pp. 121-134, 1976.
- [5] Grossberg, S., "Adaptive pattern classification and universal recording: II Feedback expectation, olfaction and illusions," *Biological Cybernetics*, vol. 23, pp. 187-202, 1976.
- [6] Hebb, D., *The Organization of Behavior: A Neuro-psychological Theory*, Wiley, NY, 1949.
- [7] Haykin, S., *Neural Networks: A Comprehensive Foundation*, Prentice-Hall, NJ, 1999.
- [8] Hopfield, J. J., "Neural networks and physical systems with emergent collective computational ability," *Proc. of the National Academy of Science*, USA, vol. 79, pp. 2445-2558, 1982.
- [9] Kolpf, A. H., "A drive-reinforcement model of single neuron function: an alternative to Hebbian neuronal model," *Proc. of the American J. of Physics, Special Issue on Neural Network Computing*, pp. 265-270, 1986.
- [10] Kosko, B., "Adaptive bidirectional associative memory," *IEEE Trans. on Systems, Man and Cybernetics*, vol. 18, no. 1, pp. 49-60, 1988.
- [11] Kosko, B., *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*, Prentice-Hall, NJ, 1992.
- [12] Rumelhart, D. E. and Zipser, D., "Feature discovery by competitive learning," *Cognitive Science*, vol. 9, pp. 75-112, 1985.

- [13] Sejnowski, T. J., "Strong covariance with nonlinearly interacting neurons," *J. Math Biology*, vol. 4, pp. 303-321, 1977.
- [14] Takeuchi, A. and Amari, S.-I., "Formation of topographic maps and columnar microstructures," *Biological Cybernetics*, vol. 35, pp. 63-74, 1979.
- [15] Yegnanarayana, B., *Artificial Neural Networks*, Prentice-Hall of India, New Delhi, 1988.

# 8

# Supervised Neural Learning Algorithms

*The chapter presents supervised learning algorithms for training feed-forward neural networks. It begins with McCulloch-Pitts model and demonstrates its application in realization of binary logic functions. Rosenblatt's perceptron learning algorithm designed for the McCulloch-Pitts neuronal model is presented next. Application of the perceptron learning model in both linear and nonlinear classification problems is then introduced. The chapter also covered Widrow-Hoff's ADALINE model and discussed its application in translation and rotation invariant pattern recognition. The most important aspect of the chapter is the derivation of the classical back-propagation learning algorithm from the principles of gradient descent learning. The chapter ended with discussions on Radial Basis Function neural nets and modular neural nets.*

## 8.1 Introduction

In the last chapter, we outlined the principles of supervised learning. The present chapter provides several algorithms of supervised learning and outlines the scope of their application in pattern classification problems. Most of the

supervised learning algorithms, to be introduced shortly, employ a trainer who provides the input-output instances of the problem. The learning algorithm adapts the weights/ thresholds of the neurons in a manner, so that for a given input instance the network correctly reproduces the desired (target) output instance.

Most of the supervised learning algorithms are initialized with a random set of weights and thresholds. The learning algorithm in each program iteration (or learning epoch) evaluates the *errors* at the output of each neuron and attempts to adapt the weights so as to minimize the errors. Supervised learning thus aims at determining the location of the global minimum on the error surface of weights (or thresholds). The dimension of the network usually plays a vital role in the *steady-state error margin* after the network converges to one minimum on the error surface.

When the error surface of the network weights is rough with several local minima, there is a chance that the learning algorithm may settle down at one of the local minima. This phenomenon, well known as trapping at local minima, is of major concern with supervised learning. One way to avoid this problem is to slowly adapt the network weights and add a momentum term to the weight adaptation policy so that the network elapses some more time before to settle down at minima.

The topology of the network also is an important consideration for the realization of a learning algorithm. The supervised learning algorithms, to be introduced in the chapter, are all realized with feed-forward neural networks. Recently, supervised learning has also been employed on recurrent neural topology [5]. The discussion on this, however, goes beyond the scope of the book.

The chapter is classified into 8 sections. In section 8.2, we present the classical McCulloch-Pitts model. The perceptron model is introduced in section 8.3. The adaptive linear combiner model proposed by Widrow-Hoff is presented in section 8.4. Section 8.5 covers the well-known back-propagation algorithm proposed by Rumelhart. The radial basis function neural net is outlined in section 8.6. Modular neural net is presented in section 8.7. The chapter ends with a discussion on the scope of the proposed neural algorithms in different applications.

## 8.2 McCulloch-Pitts Model

A McCulloch-Pitts (MP) neuron consists of 2 basic units: i) a weighted summer, and ii) a non-linearity [9]. The weighted summer computes the sum of the weighted inputs and generates an analog signal Net. The signal Net is applied to

the input of a non-linearity, which yields a binary output Out. The non-linear function in the MP model is basically a unit step function, given by

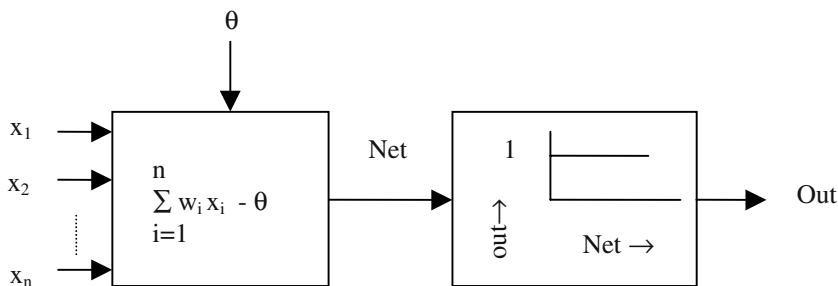
$$\left. \begin{array}{l} \text{Out} = 1, \text{Net} > 0 \\ \text{Out} = 0, \text{Net} \leq 0. \end{array} \right\} \quad (8.1)$$

Net in the McCulloch-Pitts model is formally written as

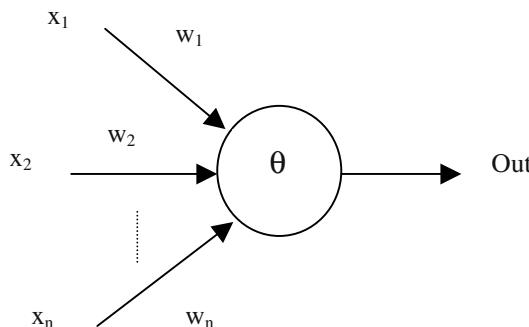
$$\text{Net} = \sum_{i=1}^n w_i x_i - \theta \quad (8.2)$$

where

- $x_i$  is an input for  $i=1$  to  $n$ ,
- $w_i$  is the weight of input  $x_i$ , and
- $\theta$  is a bias term.

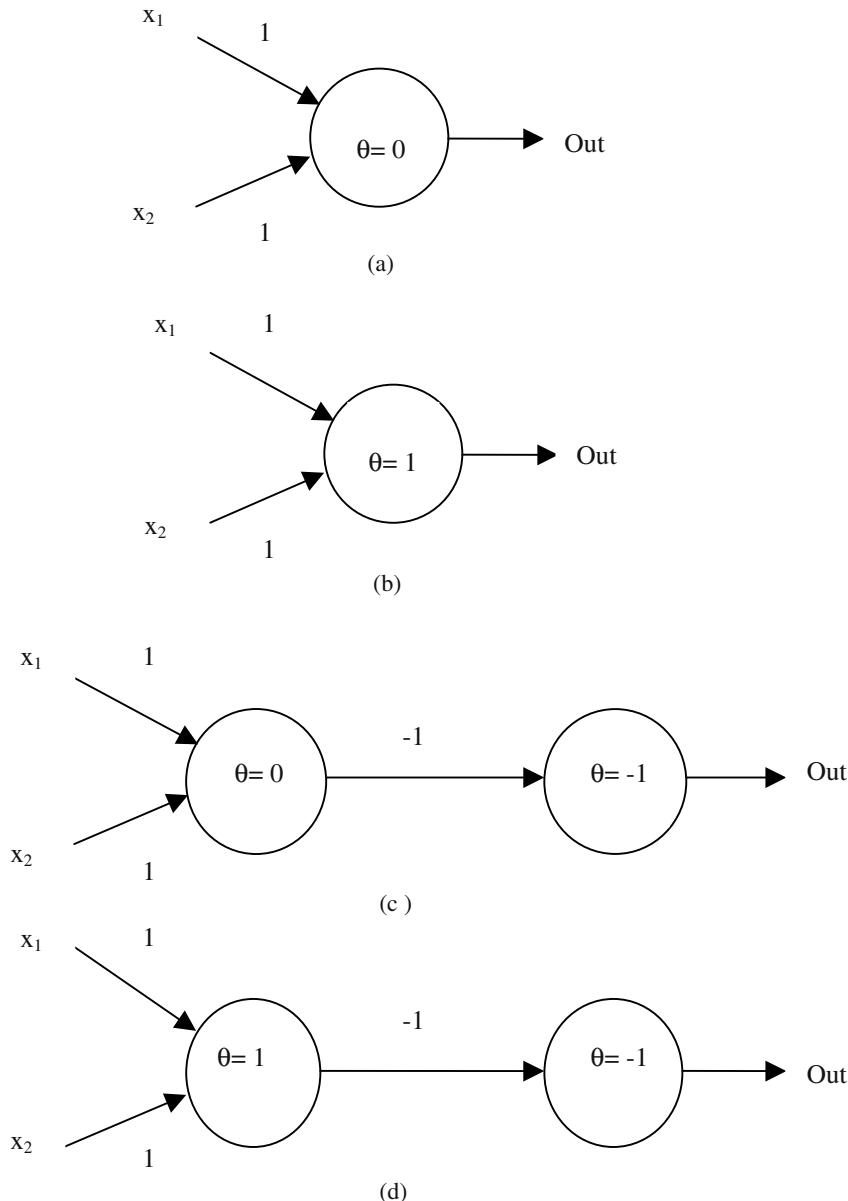


**Fig. 8.1:** The McCulloch-Pitts neuron containing a weighted summer and a step type non-linearity.



**Fig. 8.2:** A symbolic representation of Fig. 8.1.

The schematic architecture of a McCulloch-Pitts neuron and its symbolic representation are presented in Fig. 8.1 and Fig. 8.2 respectively. McCulloch-Pitts neurons can perform elementary logic operations like OR, AND, NOR and NAND (Fig. 8.3).



**Fig. 8.3:** Realization of 2-input (a) OR, (b) AND, (c) NOR and (d) NAND functions using McCulloch-Pitts neurons.

It can easily be verified that the networks presented in Fig. 8.3 support the truth table of the respective logic functions. For instance, in Fig. 8.3(a) we have

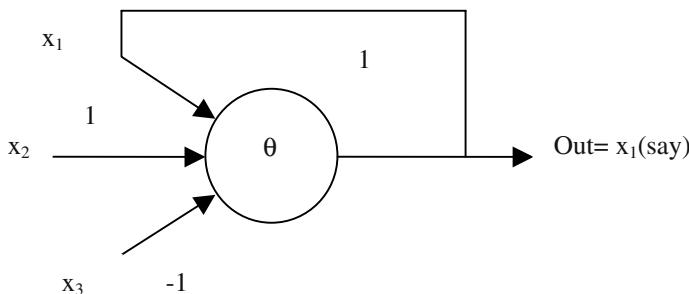
$$\text{Out} = u(x_1 + x_2) \quad (8.3)$$

for  $(x_1, x_2) \in (0, 1)$ ;  $u$  in expression (8.3) denotes a unit step function.

It is clear from (8.3) that  $\text{Out}=1$ , when at least one input of  $(x_1, x_2)$  is 1, and 0 otherwise. Thus Fig. 8.3(a) represents the OR-function. The justification of other networks in Fig. 8.3 can be given similarly.

McCulloch-Pitts model can also be used as a memory cell. For instance let us consider a neuron (Fig. 8.4) that requires a unit delay for signal processing. Thus the dynamics of the neuron is given by

$$x_1(t+1) = u[x_1(t) + x_2(t) - x_3(t) - \theta]. \quad (8.4)$$



**Fig. 8.4:** McCulloch-Pitts model as a memory element.

It is clear from Fig. 8.4 that McCulloch-Pitts model can be used as a memory cell.

### 8.3 The Perceptron Learning Model

Rosenblatt [13, 14] designed a learning law for weight adaptation in the McCulloch-Pitts model. A scheme of Rosenblatt's learning principle is presented in Fig. 8.5.

Let

$\mathbf{X}_k = [x_1(k) \ x_2(k) \ \dots \ x_n(k)]$  be a input vector at time instant  $k$ ,

$\mathbf{W}_k = [w_1(k) \ w_2(k) \ \dots \ w_n(k)]$  be the weight vector at time  $k$ ,

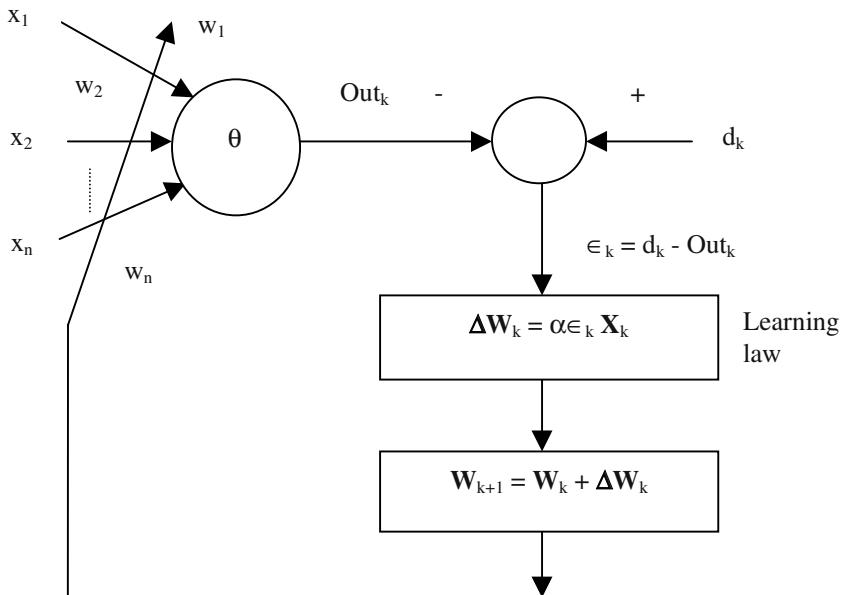
$\Delta \mathbf{W}_k$  be a change in weight vector  $\mathbf{W}_k$  at time k,

$\text{Out}_k$  be the output scalar of the McCulloch-Pitts neuron at time instant k,

$d_k$  be the desired output at time instant k,

$\epsilon_k$  be the error signal at time instant k,

$\alpha$  be the learning rate, such that  $\alpha > 0$ .



**Fig. 8.5:** The perceptron learning scheme.

To study the convergence of the perceptron learning algorithm, we first define  $\text{Net}_k$  following the McCulloch-Pitts neuronal model.

$$\begin{aligned}
 \text{Net}_k &= \sum_{i=1}^n w_i(k) x_i(k) - \theta \\
 &= \sum_{i=1}^n w_i(k) x_i(k) + (-1) x_0(k) \text{ (say)} \\
 &= \sum_{i=0}^n w_i(k) x_i(k), \text{ where } w_0 = -1
 \end{aligned}$$

$$= \mathbf{X}_k^T \mathbf{W}_k. \quad (8.5)$$

$$\begin{aligned} \text{Out}_k &= f(\text{Net}_k) \\ &= \left. \begin{array}{l} = 1, \text{ if } \text{Net}_k \geq 0, \\ = 0, \text{ otherwise.} \end{array} \right\} \end{aligned} \quad (8.6)$$

Then, we define error

$$\begin{aligned} \epsilon_k &= d_k - \text{Out}_k \\ &= d_k - f(\text{Net}_k) \\ &= d_k - f(\mathbf{X}_k^T \mathbf{W}_k) \end{aligned} \quad (8.7)$$

The perceptron learning law [10,11], which is used to adapt the weights in Fig. 8.5 is given by

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \Delta \mathbf{W}_k, \quad (8.8)$$

where

$$\Delta \mathbf{W}_k = \alpha \epsilon_k \mathbf{X}_k. \quad (8.9)$$

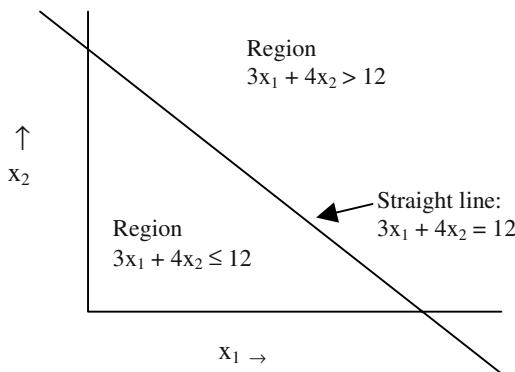
The learning laws (8.8) and (9.9) are used in the perceptron learning algorithm presented below:

### **The Perceptron Learning Algorithm**

1. Compute activation for input pattern  $\mathbf{X}$ .
2. Compute the error  $\epsilon$ .
3. Modify the connection weights by adding to it the term  $\alpha \epsilon \mathbf{X}$ .
4. Repeat steps 1, 2 and 3 for each input pattern.
5. Repeat step 4 until error is zero or very small for all the input patterns.

#### **8.3.1 Linear Classification by Perceptrons**

Miniski and Papert [11] proved an important property of perceptrons, stated as follows: “A perceptron is capable of learning a function that it can represent.” Thus binary AND, OR, NAND and NOR functions, which can be realized with perceptrons can also be learned by a perceptron. What does the above statement actually mean? It means that if a function say, an AND function, can be configured on a perceptron with judiciously selected weights, then by invoking the perceptron learning algorithm, one set of weights that describes the function can be determined. This idea led to the foundation of linear classification by perceptrons.

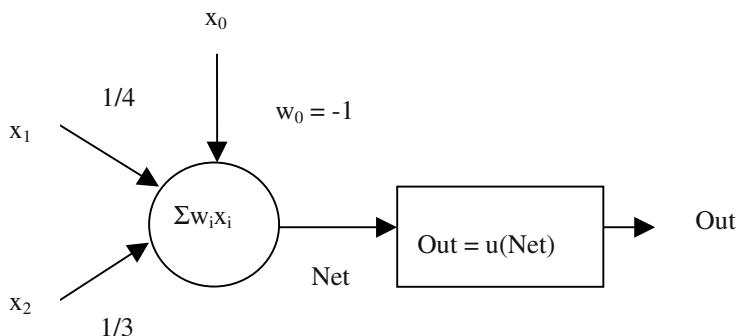


**Fig. 8.6:** Linear classification of a 2-D region.

To illustrate the concept of linear classification by perceptrons, suppose we are given the following inequality:

$$3x_1 + 4x_2 \leq 12. \quad (8.10)$$

Suppose, we like to classify the entire 2-D space \$(x\_1, x\_2) \in \mathbb{R}^2\$ into 2 classes, one class that satisfy the inequality and the other class that does not satisfy it. Fig. 8.6 illustrates the classification by a straight line: \$3x\_1 + 4x\_2 = 12\$. The region below the straight line satisfies the given inequality, and the region above it does not. Since a straight line separates the two regions, we call it a linear classification. But how can we design a perceptron to classify the region? The answer to this is explained below.



**Fig. 8.7:** A perceptron classifier for \$3x\_1 + 4x\_2 \leq 12\$.

Consider a perceptron with 2-inputs  $x_1$  and  $x_2$  and a bias input  $x_0$ . For convenience in realization, we re-write the inequality:  $3x_1 + 4x_2 \leq 12$  as  $x_1(1/4) + x_2(1/3) - 1 \leq 0$ . Consequently, we set the weights  $w_1 = 1/4$ ,  $w_2 = 1/3$ , and  $x_0 = 1$  in Fig. 8.7.

Does the above settings satisfy the inequality? We take 4 points  $(0, 0)$ ,  $(4, 0)$ ,  $(0, 3)$  and  $(4, 4)$ . The first 3 points satisfy the inequality and thus Net  $\leq 0$  and Out = 0. The last point  $(4, 4)$  does not satisfy the inequality, and Net =  $4(1/4) + 4(1/3) - 1 = 4/3 > 0$ , and thus Out = 1. So, the design of the perceptron supports the classification of the 2-D space into 2 distinct regions.

### 8.3.2 Convergence of the Perceptron Learning Algorithm

Let there be 2 classes  $C_1$  and  $C_2$ , where  $C_1$  and  $C_2$  correspond to the training instances for which the corresponding output Out is 1 and 0 respectively. Iterating  $k=0$  to  $K$  in the perceptron learning equation (8.8) we obtain:

$$\mathbf{W}_1 = \mathbf{W}_0 + \alpha_{\in 0} \mathbf{X}_0, \quad (8.11)$$

$$\begin{aligned} \mathbf{W}_2 &= \mathbf{W}_1 + \alpha_{\in 1} \mathbf{X}_1, \\ &= \mathbf{W}_0 + \alpha_{\in 0} \mathbf{X}_0 + \alpha_{\in 1} \mathbf{X}_1 \end{aligned} \quad (8.12)$$

.... .... ....

$$\mathbf{W}_{K+1} = \mathbf{W}_K + \alpha_{\in 0} \mathbf{X}_0 + \alpha_{\in 1} \mathbf{X}_1 + \dots + \alpha_{\in K} \mathbf{X}_K \quad (8.13)$$

Assuming  $\mathbf{W}_0$  and  $\in_0$  to be zero, we obtain:

$$\mathbf{W}_{K+1} = \sum_{j=1}^K \alpha_j \mathbf{X}_j. \quad (8.14)$$

Let there be a weight vector  $\mathbf{W}^*$  that can linearly separate the classes  $C_1$  and  $C_2$ . Pre-multiplying both sides of the last expression by the row vector  $\mathbf{W}^{*T}$ , we have:

$$\mathbf{W}^{*T} \mathbf{W}_{K+1} = \alpha \sum_{j=1}^K \mathbf{W}^{*T} \mathbf{X}_j. \quad (8.15)$$

Since output vectors  $\mathbf{X}_j$  are misclassified,  $\in_j \mathbf{W}^{*T} \mathbf{X}_j$  is strictly positive. To verify this, consider the case when  $\mathbf{W}^{*T} \mathbf{X}_j$  is positive. Since  $\mathbf{W}^*$  correctly classifies all the input vectors, then the target value of  $\mathbf{X}_j$  is  $d_j = 1$ , and  $\in_j = 1 - (-1) > 0$ , and consequently,  $\in_j \mathbf{W}^{*T} \mathbf{X}_j$  is positive. Following similar reasoning for the case when  $\mathbf{W}^{*T} \mathbf{X}_j$  is negative, we conclude that  $\in_j \mathbf{W}^{*T} \mathbf{X}_j$  is positive. We thus select a positive number  $a$ , where

$$a = \operatorname{Min}_{j} (\in_j \mathbf{W}^{*T} \mathbf{X}_j). \quad (8.16)$$

Then from (8.15),

$$\mathbf{W}^{*T} \mathbf{W}_{K+1} \geq K a. \quad (8.17)$$

The Cauchy-Schwartz inequality for two vectors  $\mathbf{A}$  and  $\mathbf{B}$  in finite dimensional real space is  $\|\mathbf{A}\|^2 \|\mathbf{B}\|^2 \geq \|\mathbf{A}^T \mathbf{B}\|^2$  and when applied to  $\mathbf{W}^*$  and  $\mathbf{W}_{K+1}$  yields

$$\|\mathbf{W}_{K+1}\|^2 \geq \frac{|\mathbf{W}^{*T} \mathbf{W}_{K+1}|^2}{\|\mathbf{W}^*\|^2} \quad (8.18)$$

where  $\| . \|$  is the Euclidean norm of a vector and  $| . |$  indicates the absolute value of its real-valued argument.

Combining (8.17) and (8.18) we get:

$$\|\mathbf{W}_{K+1}\|^2 \geq \frac{(K a)^2}{\|\mathbf{W}^*\|^2}. \quad (8.19)$$

To find a finite value of  $K$ , we take the square of the Euclidean metric on both sides of the update rule:

$$\mathbf{W}_{j+1} = \mathbf{W}_j + \alpha \in_j \mathbf{X}_j, \quad [\text{Combined (8.8) \& (8.9) with suffix } j]$$

$$\text{i.e., } \|\mathbf{W}_{j+1}\|^2 = \|\mathbf{W}_j\|^2 + \|\alpha \in_j \mathbf{X}_j\|^2 + 2 \alpha \in_j \mathbf{W}_j^T \mathbf{X}_j \quad (8.20)$$

Defining

$$Q = \operatorname{Max}_j \|\alpha \in_j \mathbf{X}_j\|^2 \quad (8.21)$$

and using the fact  $\in_j \mathbf{W}_j^T \mathbf{X}_j$  is negative, as  $\mathbf{X}_j$  is misclassified, we write:

$$\|\mathbf{W}_{j+1}\|^2 \leq \|\mathbf{W}_j\|^2 + Q \quad (8.22)$$

Adding the inequalities (8.22) for  $j=1$  to  $K$  we have:

$$\|\mathbf{W}_{K+1}\|^2 \leq Q \cdot K. \quad (8.23)$$

Thus from (8.19) and (8.23) we obtain:

$$Q \cdot K \geq \|\mathbf{W}_{K+1}\|^2 \geq \frac{(K a)^2}{\|\mathbf{W}^*\|^2} \quad (8.24)$$

Dividing all sides by  $Q \cdot K$  we finally arrive at the inequality:

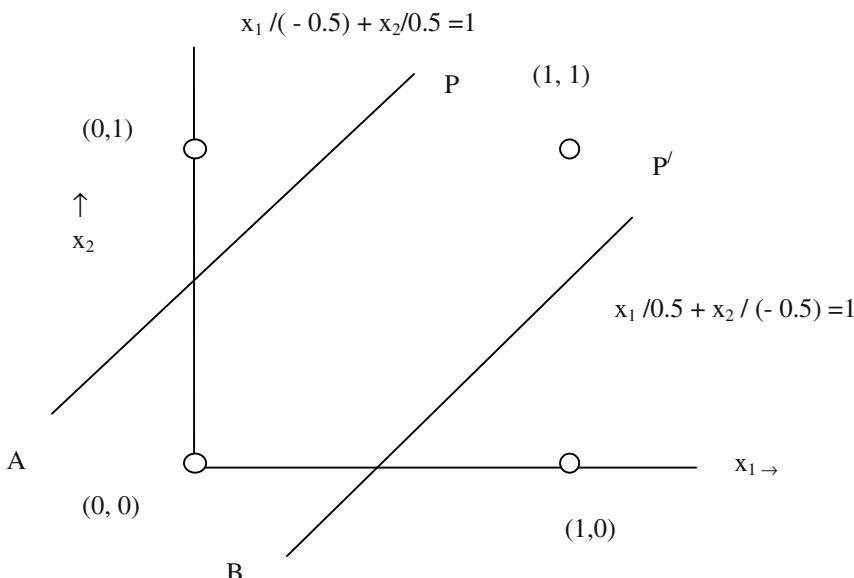
$$1 \geq \frac{(K a)^2}{\|W^*\|^2} \quad (8.25)$$

from which it is clear that  $K$  cannot grow without bound as it would violate the inequality. Thus  $K$  must be finite. The upper bound of  $K$  is

$$K = K_{\max} = \frac{Q \|W^*\|^2}{a^2} \quad (8.26)$$

### 8.3.3 Multilayered Perceptron Classifier

It is now clear that a perceptron can perform linear classification. This section illustrates that multilayered perceptrons can classify the non-linearly separable regions. As an example, consider the input space of a 2-input XOR gate. Suppose, we want to classify its input space into two regions: one region for those inputs  $x_1, x_2$  for which output  $y=1$ , and the other region for those coordinates  $(x_1, x_2)$  for which  $y=0$ . This is feasible by classifying the regions by 2 straight lines AP and  $BP'$  (Fig. 8.8).



**Fig. 8.8:** The input space of a 2-input XOR gate is partitioned into 2 regions by two straight lines  $AP$  and  $BP'$ . The region within the lines represents one class, and the region outside the lines describe the other class.

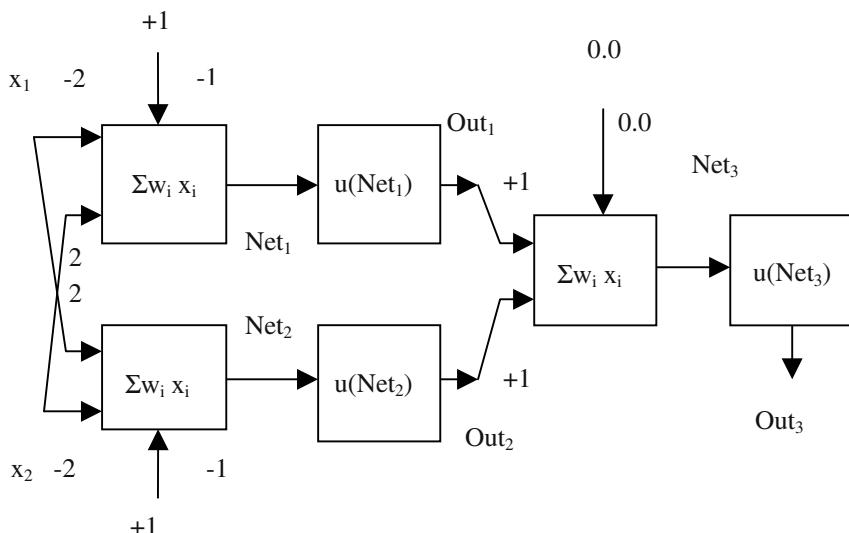
We can represent the region within the lines AP and BP' by the following 2 inequalities:

$$x_1/(-0.5) + x_2/0.5 \leq 1 \quad (8.27)$$

$$\text{and} \quad x_1/0.5 + x_2/(-0.5) \leq 1 \quad (8.28)$$

The inequality represented by (8.27) can be realized with a perceptron with parameters:  $w_1 = -1/0.5 = -2$ ,  $w_2 = 1/0.5 = 2$ , and  $x_0 = +1$ . Similarly, we realize the inequalities (8.28) by a perceptron with  $w_1 = 2$ ,  $w_2 = -2$  and  $x_0 = +1$ . The joint occurrence of the two inequalities is defined by a OR function. The OR function can also be realized with a third perceptron with parameters:  $w_1 = 1$ ,  $w_2 = 1$  and  $x_0 = 0.0$ . Fig. 8.9 describes the complete design.

In Fig. 8.9,  $\text{Out}_1 = 0$  for  $(x_1, x_2) = (0, 0), (1, 1)$  and  $(1, 0)$ , and  $\text{Out}_2 = 0$  for  $(x_1, x_2) = (0, 0), (1, 1)$  and  $(0, 1)$ . Thus an OR function realized with the third perceptron segregates the points  $(0, 0)$  and  $(1, 1)$  from the points  $(1, 0)$  and  $(0, 1)$ . It may be noted that  $\text{Out}_3 = 0$  for  $(x_1, x_2) = (0, 0)$  and  $(1, 1)$  and  $\text{Out}_3 = 1$  for  $(x_1, x_2) = (1, 0)$  and  $(0, 1)$ .



**Fig. 8.9:** Design of a multi-layered perceptron classifier for a XOR function.

## 8.4 Widrow-Hoff's ADALINE Model

In early 1960's Widrow and his co-worker Hoff proposed a new model of neuron, called ADaptive LINEar combiner (ADALINE) [17-19]. Fig. 8.10 presents a schematic diagram of an ADALINE neuron. Here, like perceptron learning we have a Least Mean Square (LMS) learning rule and a Signum type non-linearity outside the loop.

Let

$\mathbf{X}_k = [x_1(k) \ x_2(k) \ \dots \ x_n(k)]$  be a binary input vector of the ADALINE at iteration k,

$\mathbf{W}_k = [w_1(k) \ w_2(k) \ \dots \ w_n(k)]$  be a set of weights of the respective components of the input vector  $\mathbf{X}_k$  at instant k,

$d_k$  be the desired response at time k,

$Net_k$  be the weighted sum of the input signal components  $x_1(k), x_2(k), \dots, x_n(k)$ ,

$$Sgn(Net_k) = +1, Net_k > 0$$

$$= -1, Net_k < 0.$$

$\epsilon_k$  be the error signal at iteration k.

The LMS learning rule is given by

$$\Delta \mathbf{W}_k = \alpha \epsilon_k \mathbf{X}_k / \|\mathbf{X}_k\|^2 \quad (8.29)$$

where

$$\begin{aligned} \epsilon_k &= d_k - Net_k \\ &= d_k - \mathbf{X}_k^T \mathbf{W}_k. \end{aligned} \quad (8.30)$$

$$\text{Thus, } \Delta \epsilon_k = - \mathbf{X}_k^T \Delta \mathbf{W}_k. \quad (8.31)$$

From (8.29) and (8.31) we have:

$$\Delta \epsilon_k = - \alpha \epsilon_k \mathbf{X}_k^T \mathbf{X}_k / \|\mathbf{X}_k\|^2 \quad (8.32)$$

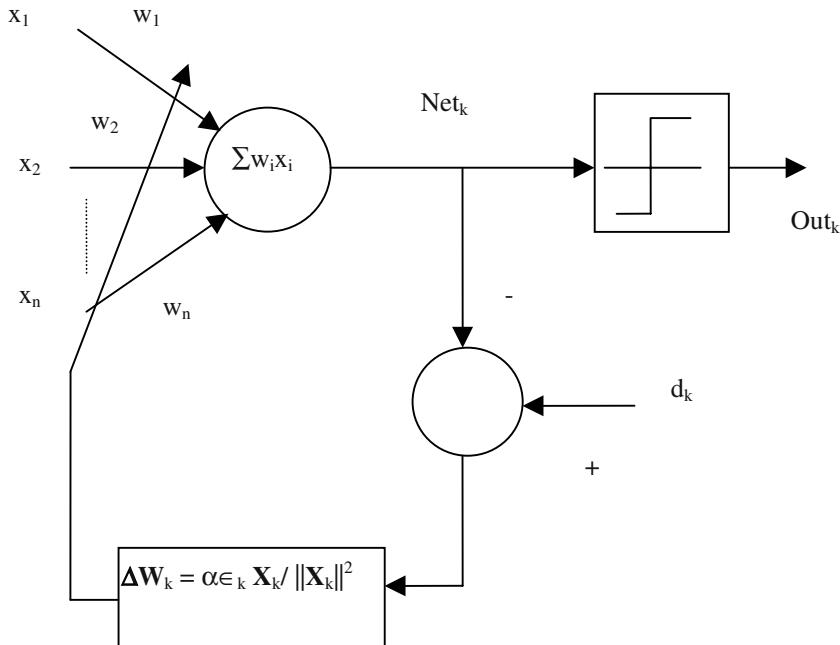
$$= - \alpha \epsilon_k. \quad (8.33)$$

$$\Rightarrow (E - 1 + \alpha) \epsilon_k = 0 \quad (8.34)$$

$$\Rightarrow E = 1 - \alpha \quad (8.35)$$

$$\text{or, } \epsilon_k = \epsilon_0 (1 - \alpha)^k. \quad (8.36)$$

Thus  $\epsilon_k$  is stable when  $0 < \alpha < 2$ ,  
 unstable when  $\alpha > 2$ , and  
 oscillatory when  $\alpha = 2$ .



**Fig. 8.10:** A typical scheme of ADALINE neuron.

One important characteristic of Widrow-Hoff's neural net is that training in such network is given by employing a *minimum disturbance principle* [19], stated below:

*Select the neuron from a given layer, whose analog output Net is closest to zero; reverse the polarity of its digital output Out and check whether the reversal reduces the error norm defined below at the output layer. If yes, adapt the weights of the selected neuron by the LMS algorithm, otherwise re-complement the digital output Out.*

To define the error norm, let us consider a typical feed-forward neural net of three layers: one input layer, one output layer and one hidden layer (Fig. 8.11).

The hidden layer is so named for its inaccessibility from the input and the output layer. Suppose, we have an input vector  $\mathbf{X} = [x_1 \ x_2]$ , one target vector  $\mathbf{Z} = [z_{d1} \ z_{d2}]$ , mapped at the input and the output layer of the network. Further, suppose that the network on submission of the input vector  $\mathbf{X}$  generates a vector  $\mathbf{Y} = [y_1 \ y_2 \ y_3]$  at the hidden layer, and  $\mathbf{Z} = [z_1 \ z_2]$  at the output layer. A scalar error norm  $\|e\|$  given by

$$\|e\| = \left[ \sum_i (z_{id} - z_i)^2 \right]^{1/2} \quad (8.37)$$

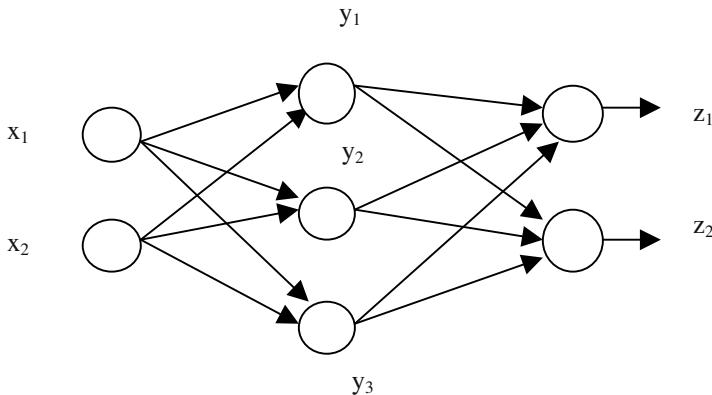
may be utilized to realize the minimum disturbance principle stated above. An outline to Widrow-Hoff's training algorithm is presented below.

### **Widrow-Hoff's training algorithm**

1. Initialize selected-layer = first-layer.
2. Identify the neuron from the selected-layer having its analog output Net closest to zero. Invert its digital output Out and check whether the inversion reduces the error norm given by (8.37). If yes adapt its weights by the LMS algorithm. If no, re-invert its digital output Out. Discard the selected neuron from the set of neurons in the layer under consideration.
3. Repeat step 2 until all neurons from the selected-layer are exhausted.
4. Assign selected-layer = Next-to(selected-layer), and repeat step 3 until selected-layer = output-layer.
5. Re-initialize selected-layer = first-layer.
6. Identify two neurons one at a time from the selected-layer having their analog outputs Net closest to zero. Invert their digital outputs Out and check whether the inversion reduces the error norm given by (8.37). If yes adapt their weights by the LMS algorithm. If no, re-invert their digital outputs Out. Discard the selected combination from the set of possible combinations in the layer under consideration.
7. Repeat step 6 until all neurons from the selected-layer are exhausted.
8. Assign selected-layer = Next-to(selected-layer), and repeat step 7 until selected layer = output layer.
9. If error norm > prescribed-limit then repeat steps 5-8 with three neurons instead of two neurons in step 6.
10. End.

The algorithm discussed above consists of 10 steps. Step 1 initializes the selected-layer as the input layer. Step 2 adapts weights in the selected layer following the minimum disturbance principle. In step 3 we repeat step 2 one by one for all neurons in the first layer. Step 4 determines the next layer as the selected-layer, and adjusts weights of all neurons in the selected layer by employing the minimum disturbance principle. Step 5 re-initializes the selected-layer as the first layer. Step 6 selects two neurons one at a time from the selected

layer by minimum disturbance principle, and adapts weights of the selected neurons. Step 7 repeats step 6 for all possible combinations of neurons. Step 8 selects next layer for adaptation of weights of the neurons therein by executing the steps under step 7. The process of weight adaptation is continued until the last layer is reached. In case the error norm is not within a prescribed limit, three neurons are selected jointly for weight adaptation following the method outlined in steps 5 to 8.



**Fig. 8.11:** A 3-layered network to illustrate the minimum disturbance principle.

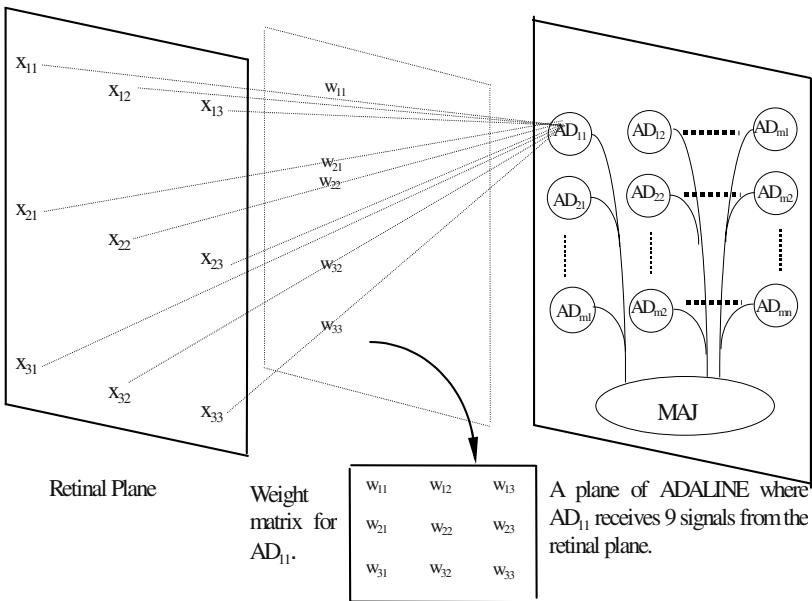
It is worthwhile to mention here that selection of more than 3 neurons one at a time is not needed for most pattern recognition applications [19]. Widrow-Hoff's algorithm presented above, thus, does not attempt to adapt weights of more than 3 neurons simultaneously.

One interesting characteristic of Widrow-Hoff's ADALINEs is their capability in recognizing translation, rotation and size invariant patterns. Widrow designed a plane of ADALINEs, whose digital outputs Out are connected to the input of a MAJority voter circuit (MAJ) (Fig. 8.12). The MAJ circuit generates a single bit output. The output is one if majority of its binary inputs is one, and zero if majority of them is zero.

A number of planes of ADALINE are needed to represent a visual pattern. As an example, assume that a visual image consists of  $(64 \times 64)$  pixels, and we want to recognize this image by ADALINEs. To handle this problem, suppose we employ 128 planes of  $(16 \times 16)$  ADALINE neurons, each having a weight matrix of  $(4 \times 4)$ . Consequently, number of pixels on the retinal plane, which is  $(16 \times 16) \times (4 \times 4) = (64 \times 64)$ , matches with that of the visual image. The number of planes, which in this example is 128, has been selected arbitrarily. It should, however, be noted that the resolution of the recognition system depends

greatly on the number of planes of ADALINEs. The more is the number of planes, the higher is the resolution of the recognition system.

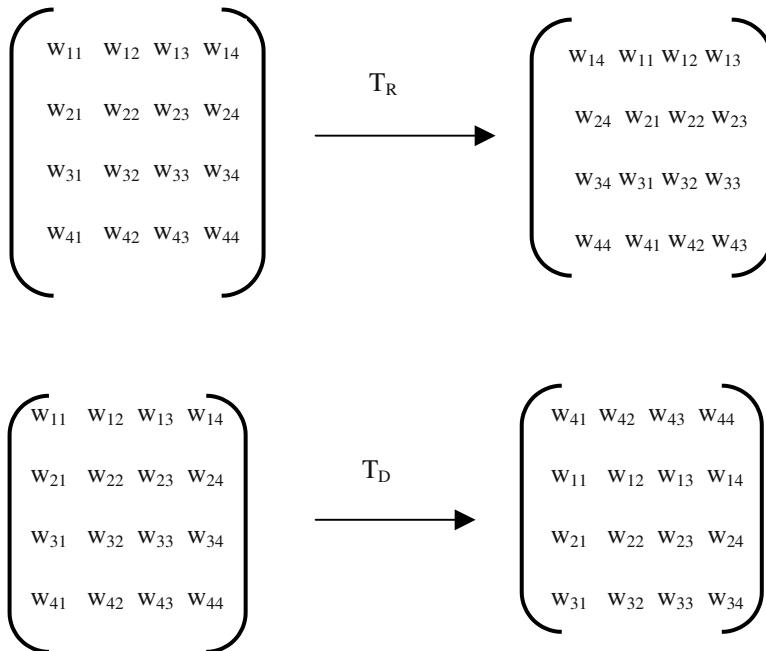
The weight matrix for ADALINE  $AD_{11}$  in Fig. 8.12 is selected randomly. The weight matrix of other ADALINEs on the same plane are determined by translating the weight matrix  $W_{11}$  of  $AD_{11}$  right and down by desired number of pixel positions. The operators  $T_R(W_{11})$  and  $T_D(W_{11})$  denote *translate right* and *translate down* of a weight matrix by one pixel position (Fig. 8.13). The operator  $T_D^m T_R^n(W_{11})$  denotes translate down by  $m$ -pixel and translate right by  $n$ -pixel of a weight matrix  $W_{11}$ . The selection of weights on a plane of ADALINE is illustrated in Fig. 8.14.



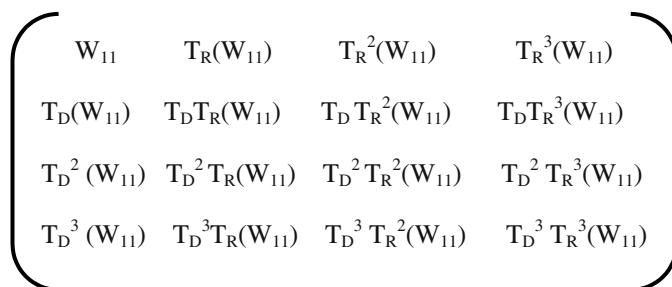
**Fig. 8.12:** One plane of ADALINEs with a weight matrix  $W_{11}$  for neuron  $AD_{11}$ . A  $(3 \times 3)$  weight matrix, instead of a  $(4 \times 4)$  matrix as discussed in the text, is shown in the figure for space limitation.

To implement  $90^\circ$ ,  $180^\circ$  and  $270^\circ$  rotational invariant pattern recognition, Widrow considered 4 planes as a group, and a MAJ circuit is employed to evaluate the majority of the binary signals generated by the 4 planes under a given group. Thus 128 plane of ADALINEs are divided into  $128/4 = 32$  groups, and we obtain a 32-bit scrambled pattern corresponding to a given input image. In order to ensure rotational insensitivity of the input patterns, Widrow proposed a scheme for the selection of the weight matrices in the planes of ADALINEs under each group. Suppose, the weight matrix on the first plane

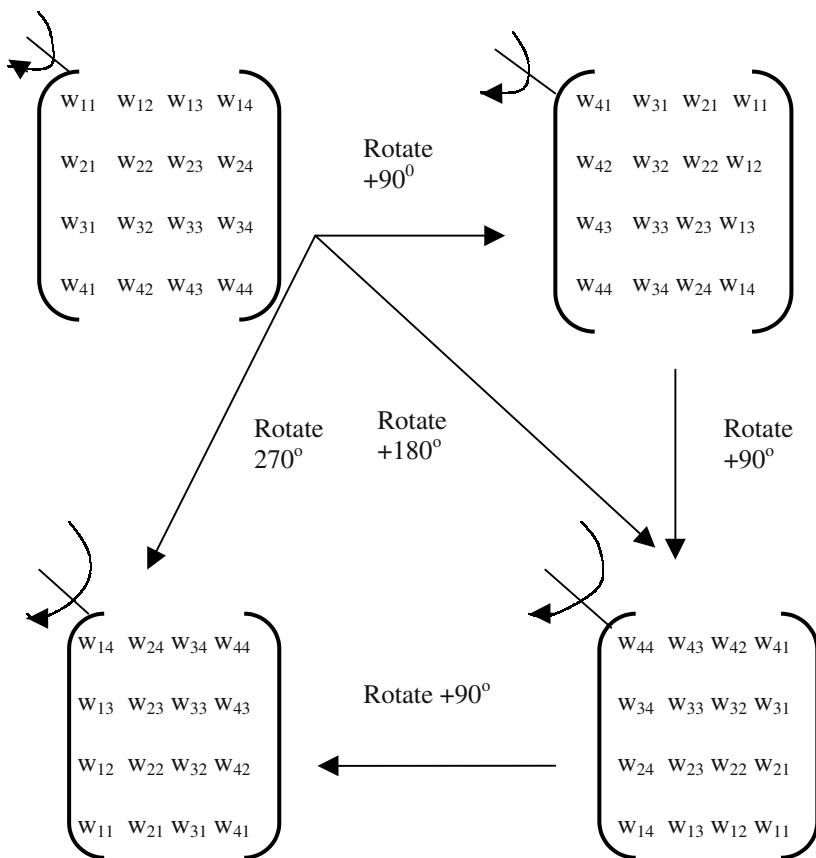
under each group is selected by satisfying the technique for translational invariance presented earlier. The weight matrix  $AD_{ij}$  in the second, the third and the fourth plane under a group are constructed by rotating the weight matrix of  $(i, j)$ th ADALINE on the first plane clockwise by an angle of  $90^\circ$ ,  $180^\circ$  and  $270^\circ$  respectively (Fig. 8.15).



**Fig. 8.13:** The translate-right  $T_R$  and the translate-down  $T_D$  by one pixel operators.



**Fig. 8.14:** The weight matrix of a plane of ADALINE, where  $T_D^i T_R^j (W_{11})$  denotes translate down by  $i$ -pixels and translate right by  $j$ -pixels of  $W_{11}$  respectively.

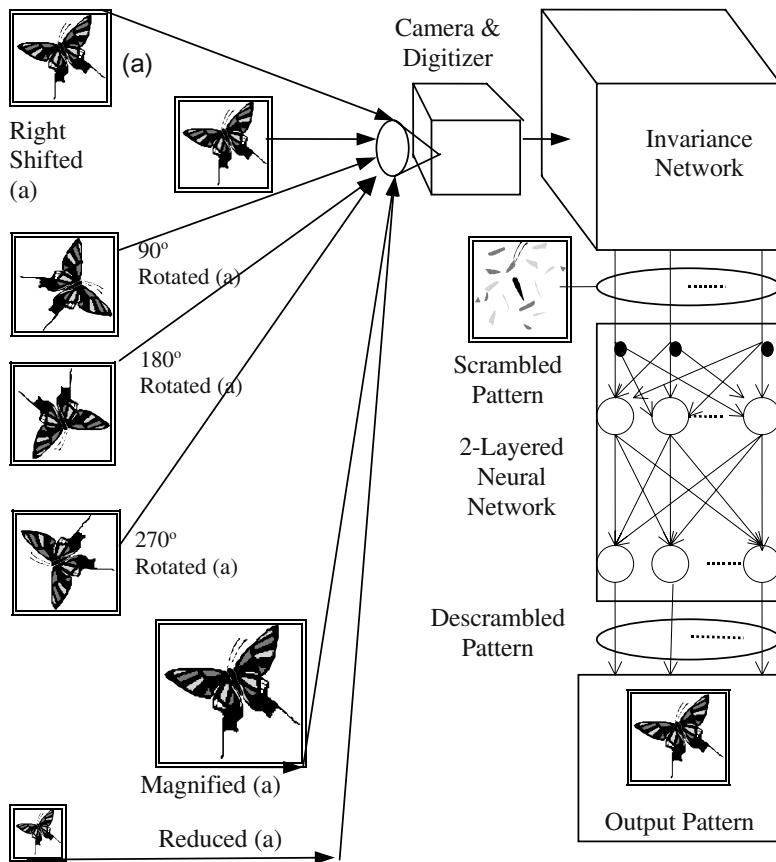


**Fig. 8.15:** The weight matrix  $AD_{ij}$  of the first plane being rotated clockwise by  $90^\circ$ ,  $180^\circ$  and  $270^\circ$  respectively with respect to left-topmost element of the matrix as the pivot.

The complete scheme of translation and rotation invariant pattern recognition is presented in Fig. 8.16. The camera in Fig. 8.16 grabs a scene and the digitizer quantizes the received image into gray levels. The invariance network converts the digitized image into a unique scrambled pattern irrespective of translation and rotation of the input image. The feed-forward neural net pre-trained with Widrow's minimum disturbance principle is employed to generate a de-scrambled pattern from the scrambled pattern.

To incorporate size-invariance in the pattern recognition scheme, Widrow considered a point of expansion (or contraction) on the retinal plane [19]. When the retinal image is expanded, the invariance network should also take care of

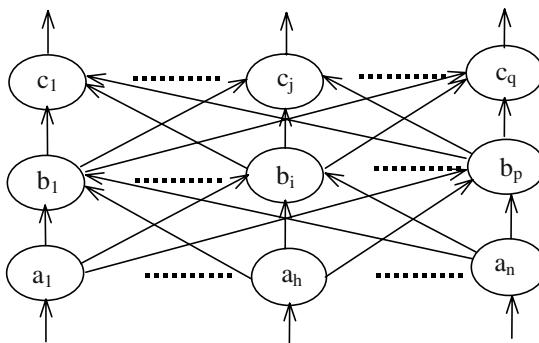
the expansion of the weight matrices for the ADALINEs on the planes. The amplitude of the weights must be scaled in inverse proportion to the square of the linear dimension of the retinal pattern. By adding many more planes, the invariance network can be built around this idea to be insensitive to pattern size, translation and rotation.



**Fig. 8.16:** The scheme for translation, rotation and size- invariant pattern recognition.

## 8.5 The Back-Propagation Learning Algorithm

As already discussed, Back-propagation learning algorithm is one of the most popular supervised learning algorithms. It employs a feed-forward topology of neurons, denoted by circles, (vide Fig. 8.17), with a number of layers (here 3) and each layer comprising of a number of neurons. The bottommost layer of the figure is the input layer, the topmost layer is the output layer and the intermediate layers (here only 1) are called the hidden layer. The neurons in the hidden and the output layers of the back-propagation algorithm receives weighted output from the neurons of the previous layer, which are summed up and then passed on to a sigmoid type non-linearity (Fig. 8.18 (a) & (b)). The neurons in the input layer simply pass the weighted output to the next layer following the connectivity between these two layers.



**Fig. 8.17:** A feed-forward topology of a neural net used in the Back-propagation learning.

The back-propagation algorithm is based on the principle of gradient descent learning. We will shortly outline this principle and use it for deriving the training procedure of the algorithm.

Let  $W_{p,q,k}$  (or simply  $W_{pq}$ ) be the weight connected between neurons  $p$  in layer  $(k-1)$  with neuron  $q$  at the  $k$ -th (output) layer (Fig. 8.18(c)). Further, let  $E$  be the Euclidean norm of the error vector, for a given training pattern, produced at the output layer. Formally,

$$E = (1/2) \sum_{\forall r} (t_r - \text{Out}_r)^2 \quad (8.38)$$

where  $t_r$  and  $\text{Out}_r$  denote the target (scalar) output and the computed output at node  $r$  in the output layer.

The gradient descent learning requires that for any weight  $W_{pq}$ ,

$$W_{pq} \leftarrow W_{pq} - \eta(\partial E / \partial W_{pq}) \quad (8.39)$$

where  $\eta (> 0)$  is the learning rate.

The computation of  $\partial E / \partial W_{pq}$ , however, is different in the last layer from the rest of the layers. We now consider two types of output layers: output layer with and without non-linearity of neurons.

When the neurons in the output layer contain no non-linearity (see Fig. 8.19),

$$\begin{aligned} Out_q &= Net_q = \sum_{\substack{\forall p \\ \text{in the} \\ \text{penultimate layer}}} W_{pq} \cdot Out_p \end{aligned} \quad (8.40)$$

Now,  $\partial E / \partial W_{pq}$

$$\begin{aligned} &= (\partial E / \partial Out_q) (\partial Out_q / \partial W_{pq}) \\ &= - (t_q - Out_q) Out_p \end{aligned} \quad (8.41)$$

$$\text{Consequently, } W_{pq} \leftarrow W_{pq} + \eta(t_q - Out_q) Out_p \quad [\text{by (8.38)}] \quad (8.42)$$

Denoting,  $(t_q - Out_q)$  by  $\delta q$  we have

$$W_{pq} \leftarrow W_{pq} + \eta \cdot \delta q \cdot Out_p \quad (8.43)$$

Now, we consider the case, when the neurons in all the layers including the output layer contain sigmoid type non-linearity. The network structure with two cascaded weights is given in Fig. 8.19.

$$\text{Here, } Out_p = 1/(1+e^{-Net_p}) \quad (8.44)$$

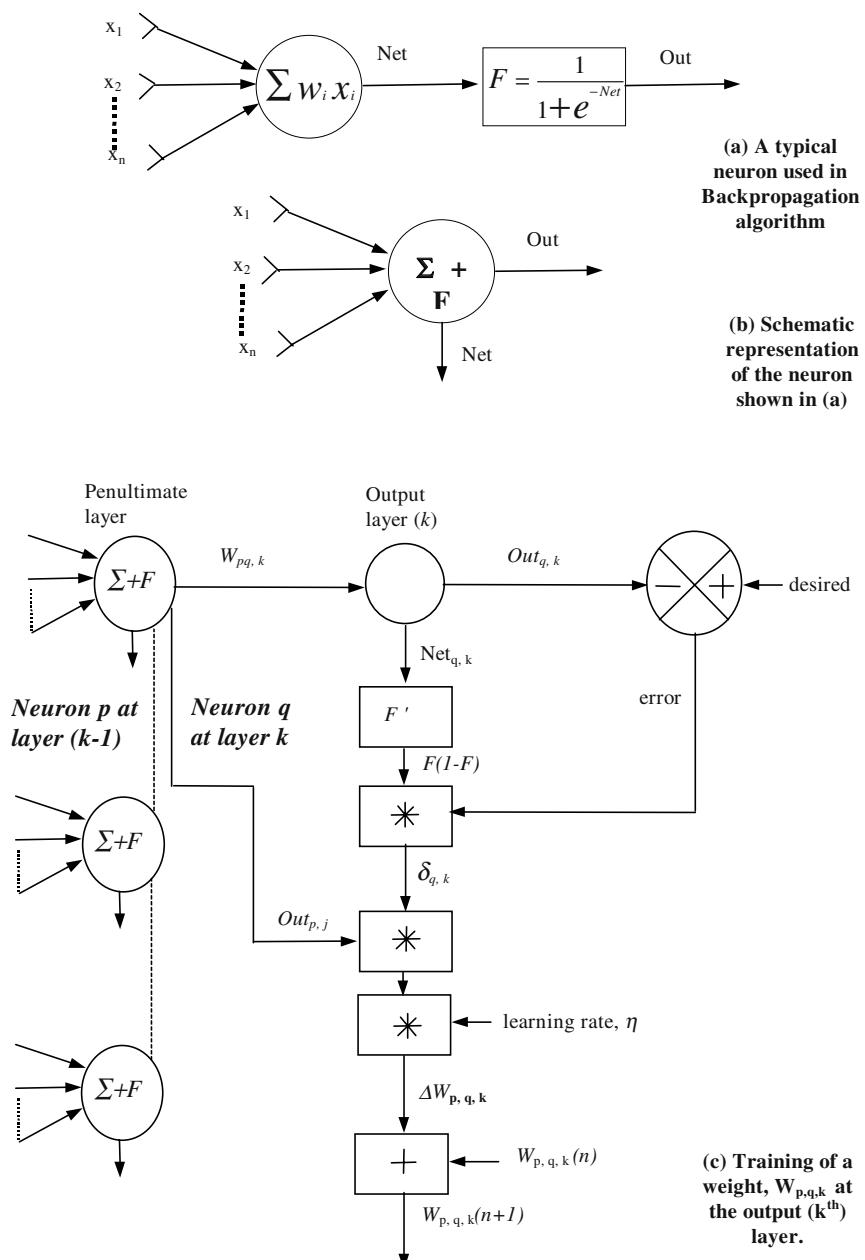
$$Net_p = \sum_r W_{rp} \cdot Out_r \quad (8.45)$$

where the index  $r$  corresponds to neurons in the hidden layer  $m$ .

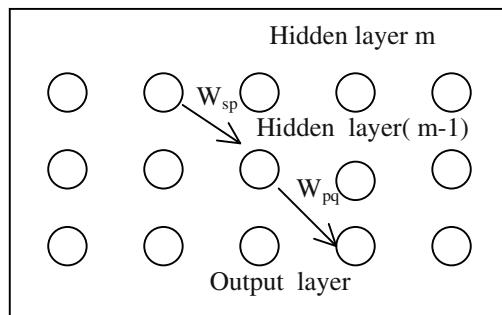
$$Out_r = 1/(1+e^{-Net_r}) \quad (8.46)$$

$$Net_r = \sum_i W_{ir} \cdot Out_i \quad (8.47)$$

where  $w_{ir}$  are the weights connected to neuron  $r$  from its preceding layer.



**Fig. 8.18:** A schematic diagram of the Back-propagation learning algorithm.



**Fig. 8.19:** Defining the weights in a feed-forward topology of neurons.

Now, for the output layer with sigmoid type non-linearity we have

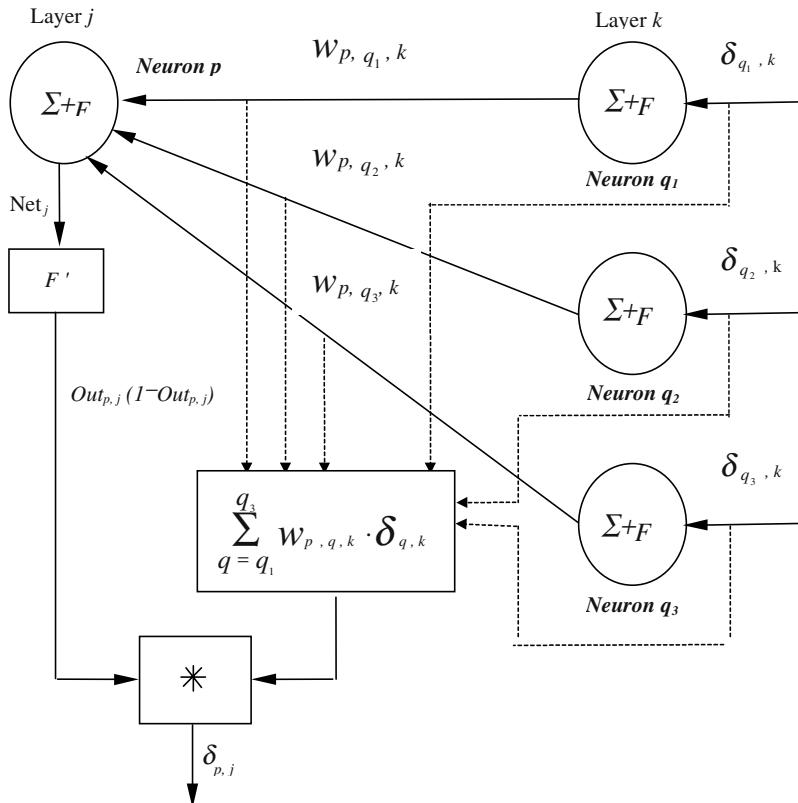
$$\begin{aligned}
 & \frac{\partial E}{\partial W_{pq}} \\
 &= (\frac{\partial E}{\partial Out_q}) (\frac{\partial Out_q}{\partial Net_q}) (\frac{\partial Net_q}{\partial W_{pq}}) \\
 &= -(t_q - Out_q) Out_q (1 - Out_q) Out_p \\
 &= -\{ (t_q - Out_q) Out_q (1 - Out_q) \} Out_p \\
 &= -\delta q \cdot Out_p \text{ (say).}
 \end{aligned} \tag{8.48}$$

The readers may now compare this result with that given in Fig. 8.18, where this is written as  $\delta_{q,k} Out_{p,j}$ .

Now, we compute the updating rule for  $W_{sp}$ :

$$\begin{aligned}
 & \frac{\partial E}{\partial W_{sp}} \\
 &= \sum_r (\frac{\partial E}{\partial Out_r}) (\frac{\partial Out_r}{\partial Net_r}) (\frac{\partial Net_r}{\partial Out_p}) (\frac{\partial Out_p}{\partial Net_p}) (\frac{\partial Net_p}{\partial W_{sp}}) \\
 &= \sum_r - (t_r - Out_r) Out_r (1 - Out_r) W_{pr} \cdot Out_p (1 - Out_p) Out_s \\
 &= - \sum_r \delta r W_{pr} \cdot Out_p (1 - Out_p) \cdot Out_s \\
 &= - Out_p \cdot (1 - Out_p) \cdot Out_s \sum_r \delta r \cdot W_{pr}
 \end{aligned} \tag{8.49}$$

The schematic diagrams explaining the computation of weights and the evaluation of the propagated error at layer  $j$  are given in Fig. 8.18(c) and Fig. 8.20 respectively. A slightly extended version of classical back-propagation algorithm is presented next.



**Fig. 8.20:** Computation of the back-propagated error at layer  $j$ .

### Extended Back-propagation Algorithm

1. Initialize instance  $i:=1$ .
2. Supply input components for the  $i$ -th instance to the input of the neural net; make a forward pass and compute the outputs.
3. Compute the error vector  $E_i$  at the output layer by taking the componentwise difference of the target vector and the computed output vector, i.e.,  $E_{ij} = T_{ij} - O_{ij}, \forall j$ , where  $E_{ij}, T_{ij}$  and  $O_{ij}$  denote the  $j$ -th component of the  $i$ -th error vector, target vector and output vector respectively.
4. Repeat steps 2 and 3 for  $i=1$  to  $n$ .

5. Determine root mean square (RMS) value of error, denoted by **ERROR**, whose  $j$ -th component is given by

$$(\text{ERROR})_j = \left[ \sum_{i=1}^n E_{ij}^2 / n \right]^{1/2}.$$

6. Back-propagate the error from the RMS value of error components of the last layer to the preceding layers and adapt the weights of the network layerwise starting with the last layer.
7. Repeat steps (2 – 6) until  $\Sigma (\text{ERROR})_j^2$  is sufficiently small.  
 $\forall j$

The back-propagation algorithm [15] presented above receives the input components of training instances for submission to the input of the network, and determines the error at the output layer by taking the difference of the target vector and the computed output vector. The process is repeated for all the training instances, and the cumulative RMS value of error taking into account of all the training instances are determined at step 5. Step 6 is the main step in the algorithm. It includes 2 sub-steps. The first sub-step refers to computation of the back-propagated errors to each neuron of all preceding layers. The second sub-step involves updating of the weights based on the back-propagated error. After all the weights in the network are adapted, a scalar performance index of the algorithm is evaluated by summing up all the components of the mean square errors, as outlined in step 7 of the algorithm. If the result exceeds a pre-determined threshold, a next pass for weight adaptation is recommended by repeating steps 2 through 6 once again.

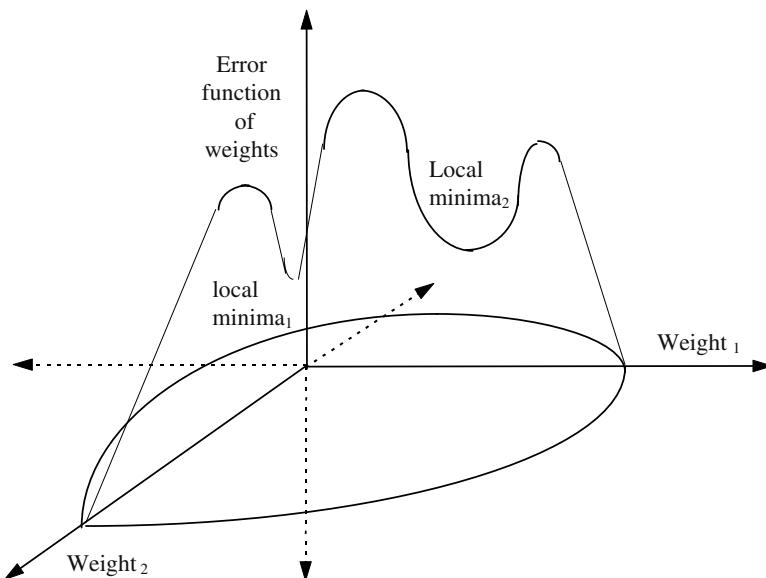
**Limitations of the Back-propagation Algorithm:** The back-propagation algorithm suffers from two major limitations, namely network paralysis and trapping at local minima. These issues are briefly outlined below.

**Network paralysis:** As the network receives training, the weights are adjusted to large values. This can force all or most of the neurons to operate at large Outs, i.e., in a region where  $F'(\text{Net}) \rightarrow 0$ . Since the error sent back for training is proportional to  $F'(\text{Net})$ , the training process comes to a virtual standstill. One way to solve the problem is to reduce  $\eta$ , which, however, increases the training time.

**Trapping at local minima:** Back-propagation adjusts the weights to reach the minima (vide Fig. 8.21) of the error function (of weights). However, the network can be trapped in local minima. This problem, however, can be solved by adding momentum to the training rule or by statistical training methods applied over the back-propagation algorithm [16].

**Adding momentum to the training rule:** To eliminate the problem of trapping at local minima, recently a momentum term is added to the right hand side of the adaptation rule [16]. Formally,

$$W_{p,q,k}(n+1) = W_{p,q,k}(n) + \eta \delta_{q,k} \text{Out}_{p,j} + \alpha \Delta w_{p,q,k}(n-1) \quad (8.50)$$



**Fig. 8.21:** Valleys in error function cause Back-propagation algorithm trapped at local minima.

The last term in the right hand side of expression (8.50) corresponds to momentum. The addition of the momentum term forces the system to continue moving in the same direction on the error surface, without trapping at local minima. A question may naturally arise: why to call this momentum term? The answer came from the analogy of a rolling ball with high momentum passing over a narrow hole.

## 8.6 The Radial Basis Function Neural Net

A radial basis function is a multidimensional function that describes the distance between a given input vector and a pre-defined center vector. There are different types of radial basis functions. Most common among them is the Gaussian function, which is mathematically defined as follows:

$$f(r_i) = \exp(-r_i^2 / \sigma_i^2), \quad (8.51)$$

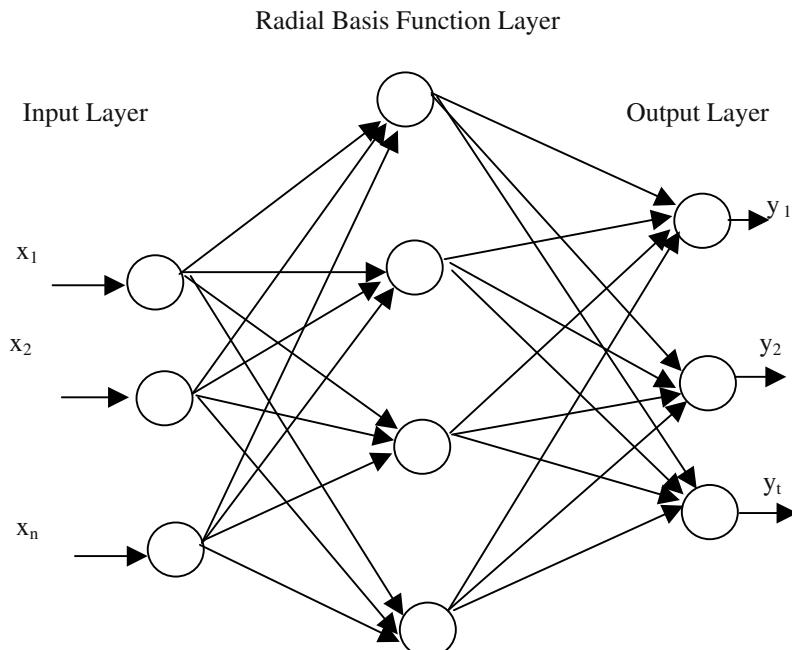
where  $r_i$  is the Euclidean distance between a given input vector  $X_i$  and a center vector  $C_i$ , i.e.,

$$r_i = \| X_i - C_i \| \quad (8.52)$$

and  $\sigma_i^2$  is the variance of the neighborhood  $m$  number of points  $X_k$ ,  $1 \leq k \leq m$ , around the cluster center  $C_i$ . Formally,

$$\sigma_i^2 = (1/m) \sum_{k=1}^m \| (X_k - C_i) \|^2 \quad (8.53)$$

The Radial Basis Function (RBF) neural net comprises of 3 layers: the input layer, the RBF layer (hidden layer) and the output layer (Fig. 8.22). The input layer simply transfers the input vector  $X = [x_1 \ x_2 \ \dots \ x_n]$  through scalar weights to the next layer. In most applications the scalar weights are assigned unity values. Thus the whole input vector appears to each neuron in the hidden layer. The hidden layer comprises of neurons that perform the radial basis function over the incoming vector that appears at the input of each RBF neuron.



**Fig. 8.22:** Schematic diagram of a Radial Basis Function Network.

Prior to using RBF neural net for pattern clustering, the cluster center  $C_1$  and the variance  $\sigma_i^2$  for each neuron in the RBF layer needs to be determined. Fixing the number of clusters and hence the number of neurons in the RBF layer generally depend on the nature of the problem itself. This is usually done by unsupervised learning algorithms such as nearest neighbor clustering, fuzzy C-means clustering or SOFM.

The output layer yields a vector  $Y = [y_1 \ y_2 \dots \ y_n]$  by linear combination of the outputs of the hidden layer like a single layered perceptron classifier. The training algorithm of a RBF neural net is outlined below.

1. Initialize the connection weights from the hidden layer to the output layer randomly.
2. **Repeat** for each input pattern  $X_k$ 
  - a) Compute the outputs  $y_i$  by  $y_i = \sum_{j=1}^h W_{ij} f_j$  for all  $i$ , where  $i$  is the number of the output layered neurons, and  $j$  is the selected neuron in the hidden layer,  $f_j$  is the RBF of the selected node  $j$  in the hidden layer. Repeat this for all  $i$  in the output layer.
  - b) Adjust weights  $W_{ij}$  by  

$$W_{ij} = W_{ij} + \alpha e_i f_j$$
 for all  $i, j$ .  
 where  $e_i$  is the difference of the desired output and computed output at the  $i$ -th output layered node.
  - c) Compute the total error by taking the sumsquare errors at all nodes at the output layer.
- Until** the error is less than a prescribed limit.

The RBF neural net is applicable to any pattern recognition problem where the number of input patterns are quite large and there exists pattern clusters, and the output patterns are a few and have correspondence to clusters of the input patterns.

## 8.7 Modular Neural Nets

In theory, a sufficiently large neural net can be trained to accomplish the task of function approximation, irrespective of the complexity of the function. Convergence of the training algorithm for such networks, however, requires a reasonable amount of time, and thus an alternative approach to handle the problem is recommended. In this section, we present a modular approach to neural net design that includes natural decomposition of a function of large complexity into simple functions and realization of each function by a separate neural net. A significant improvement in training time can be attributed to the following two reasons [8]:

1. The modules generally address simpler tasks, and thus can be trained in less iteration in comparison to that of a single large network.

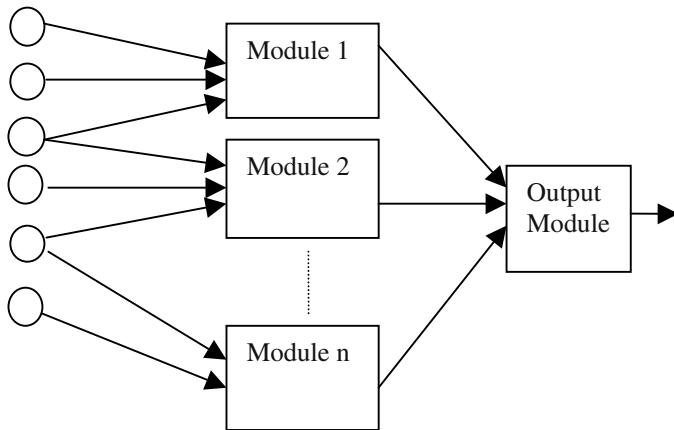
2. Modules can be trained concurrently and independently.

Besides the training time, modular architecture of neural network provides the following additional advantages.

1. **Reducing the scope of conflict in training:** Supervised training algorithms usually attempt to adjust the weights based on information aggregated from signals received from different neurons. Performance of the training algorithms is adversely affected when these signals conflict with each other. For example, in back-propagation algorithm a weight in a non-outermost layer is adjusted by taking the weighted sum of back-propagated error signals flowing from the connected nodes in the next higher layer. The error signal at one node of the next layer may be positive, while the others may be negative. Thus the resulting change in weight may be too small, thereby leading to a large convergence time. Modular fragmentation of a given function to simpler functions requires training small nets, and thus scope of conflict in training is reduced.
2. **Enhancing the scope of generalization:** A trained feed-forward neural net is expected to perform reasonably well on test data, distinct from the training data set. This is well known as the *generalization characteristic* of the network. A non-modular neural net used as a function approximator, generally fails to generalize complex nonlinear functions. Decomposition of complex functions into simpler functions help in better approximation of the simple functions, and thus modular nets are more reliable for generalization.
3. **Provision for interpretation:** Neural networks used for reasoning in expert systems generally are unable to give an interpretation for a conclusion. Modular neural nets that provides users the access right to intermediate results at the output of each module help in constructing an interpretation for a given conclusion.
4. **Overcoming the hardware constraints:** A large non-modular neural net does not easily fit on a silicon chip. Consequently, there is a hardwired constraint for its implementation. On the other hand, a modular version of the system can easily be realized on silicon by configuring each module on one chip and then by establishing inter-connectivity among the chips following the topology of the modular architecture.

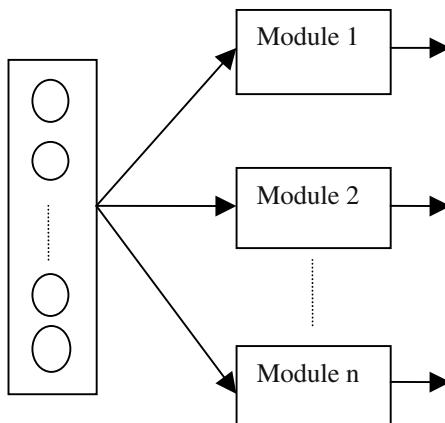
The architecture of a modular neural net takes different forms depending on the users' requirement. Four typical architectures, which are commonly used in practice, are presented below.

- 1. Input decomposition:** A system with multiple inputs may be decomposable into subsystems in such a way that the inputs of each subsystem constitute a proper subset of the inputs of the entire system, as shown in Fig. 8.23. The inputs of different sub-systems under this configuration need not be completely disjoint.



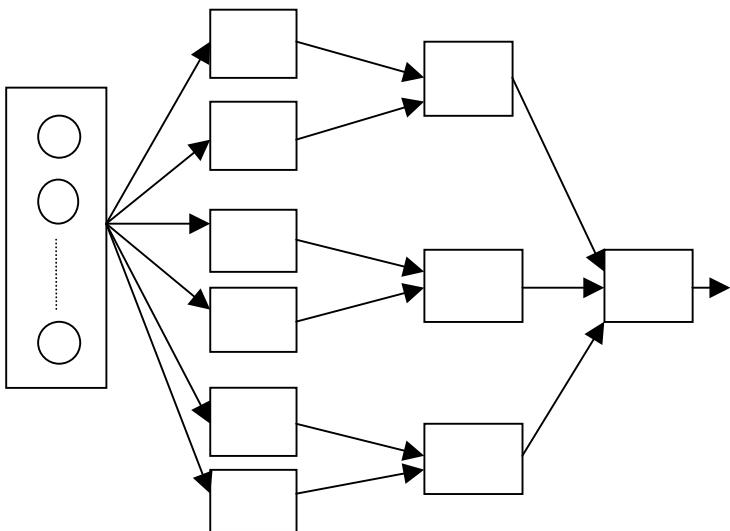
**Fig. 8.23:** Decomposition of the inputs of different modules.

- 2. Output decomposition:** Sometimes a task can be divided into sub-tasks that can be learned independently. For instance in attempting to recognize letters of the English alphabet, we may group similar letters together to form each class and train one network for each group. The basic architecture of output modularity is shown in Fig. 8.24 below.

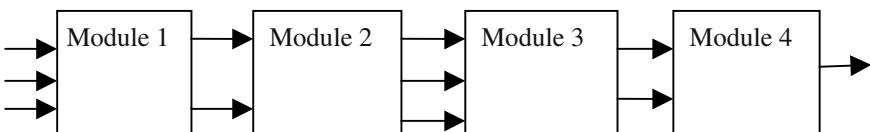


**Fig. 8.24:** A simple scheme of output modularity.

- 3. Hierarchical decomposition:** A multi-input multi-output system can sometimes be decomposed into simpler multi-input multi-output modules arranged in a hierarchy (Fig. 8.25). The outputs of lower level modules act as the input of higher level modules. The simplest type of hierarchical decomposition is a pipelined architecture (Fig. 8.26).



**Fig. 8.25:** A scheme of hierarchical decomposition.



**Fig. 8.26:** A pipelining scheme.

- 4. Mixture of expert networks:** In this architecture, the opinions of multiple expert modules operating independently, are combined in a final decision making module (Fig. 8.27). The decision-making module may form a consensus of probabilistic combination of the outputs of these modules, or may employ a competitive approach to determine which of the modules has its say for any given element of the input space.

Consider the problem of supervised learning. The training set  $T = \{(\mathbf{X}_j, \mathbf{D}_j)\}$  for  $j=1$  to  $n$  consists of  $n$ -pair of observations, where  $\mathbf{X}_j$  and  $\mathbf{D}_j$  denote an  $I$ - and  $O$ -dimensional input and output vectors respectively. The probabilistic model that relates an input vector  $\mathbf{X}_j$  to its output  $\mathbf{D}_j$  is presented below.

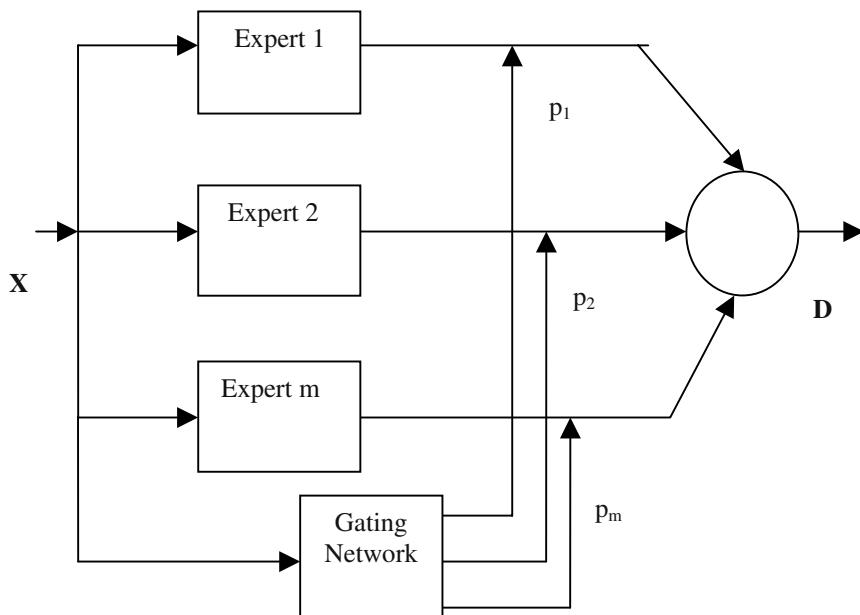
Let  $\mathbf{D}$  be a random vector,  $\theta_i$  be a system parameter and  $p_i$  denote the *a priori* probability (Fig. 8.27) that the density function  $\mathbf{D}$  is given by  $f(\mathbf{D}, \theta_i)$  for  $i = 1$  to  $m$  with

$$\sum_{i=1}^m p_i = 1. \quad (8.54)$$

Then the sum,

$$\sum_{i=1}^m p_i f(\mathbf{D}, \theta_i) = 1 \quad (8.55)$$

known as mixture density function, represents the unconditional density of  $\mathbf{D}$ .



**Fig. 8.27:** A simple schematic for mixture of expert network.

In the above model, parameters  $\theta_1, \theta_2, \dots, \theta_m$  and  $p_1, p_2, \dots, p_m$  are unknown. In a mixture of expert networks, one expert is dedicated to learn the probability density function  $f(\cdot, \theta_i)$  or the associated parameter with each  $i$ , and a gating network is assigned to learn  $p_i$ .

## 8.8 Conclusions

The chapter presented various supervised algorithms for training a feed-forward neural network. After a network is trained, it can be used for recognition of patterns. A perceptron can be used as a linear classifier. A number of perceptrons can be used together for non-linear classification of patterns. The back-propagation algorithm introduced in the chapter is useful for recognition of complex objects.

When the number of training instances is large, the back-propagation algorithm requires significant amount of time for convergence to a minimum on the error surface. A radial basis function neural net is needed under this circumstance. The RBF-network employs radial basis function at the hidden layer of the network. The RBF-neurons are tuned in a manner so that for a given input instance only one neuron representative of the class has its output close to one, while the outputs of all other RBF-neurons are close to zero.

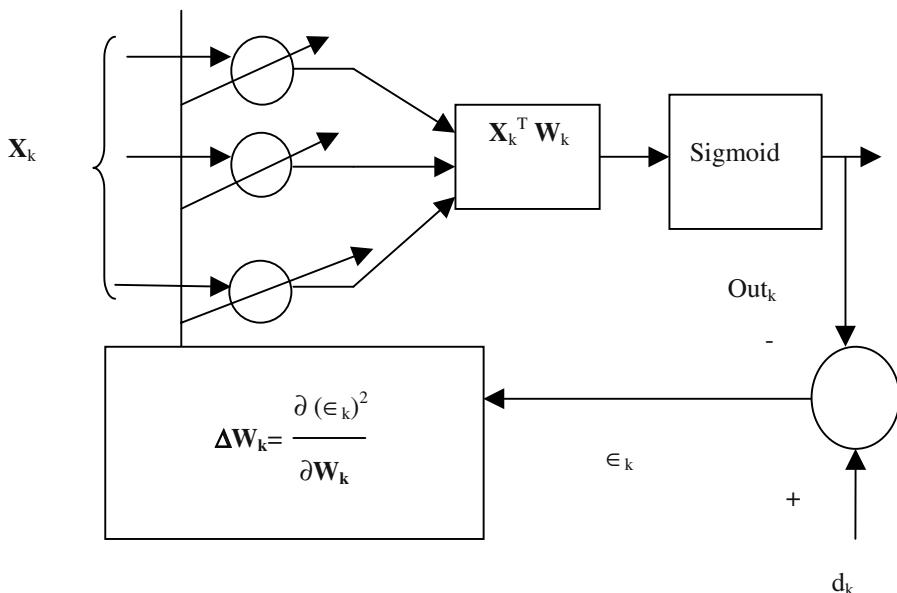
Modular neural nets outlined in this chapter are currently gaining importance for their increasing applications in pattern recognition. The main advantage of these networks is their small training time and better accuracy in recognition of patterns. Most modular neural networks employ back-propagation algorithm for training of the individual networks. Usually individual networks are trained in parallel with a small set of training instances. This ensures a good training within a small learning epochs.

Supervised learning networks have been used for recognizing objects from their images [2], path planning of a robot from its temporal obstacle map [1], target-tracking [6], mood identification of persons from their facial images [4], parameter identification of engineering systems [3] and many others [7, 8]. In subsequent chapters of the book we shall outline some of these applications.

## Exercise

1. In Fig. 8.28 we present a Sigmoid ADALINE having an input Vector  $\mathbf{X}_k$  and a weight vector  $\mathbf{W}_k$ . The desired input is a scalar  $d_k$  and

$$\text{Out}_k = S(\mathbf{X}_k^T \mathbf{W}_k), \text{ where } S \text{ is a Sigmoid function.}$$



**Fig. 8.28:** An ADALINE with a Sigmoid type non-linearity.

Show that the learning algorithm for the system is given by

$$W_{k+1} = W_k - 2 \epsilon_k S'(\text{Net}_k) X_k$$

where  $\text{Net}_k = X_k^T W_k$  and  $S'(\text{Net}_k) = \partial S / \partial \text{Net}_k$ .

[**Hints:** The learning law is given by

$$\begin{aligned} \Delta W_k &= \frac{\partial(\epsilon_k)^2}{\partial W_k} \\ &= 2 \epsilon_k \frac{\partial \epsilon_k}{\partial W_k} = -2 \epsilon_k \frac{\partial S(\text{Net}_k)}{\partial W_k} = -2 \epsilon_k \frac{\partial S}{\partial \text{Net}_k} \frac{\partial \text{Net}_k}{\partial W_k}. \end{aligned}$$

Hence, the result follows.]

2. Design a perceptron to classify the input space of a two-input NAND gate into two classes, depending on its output.

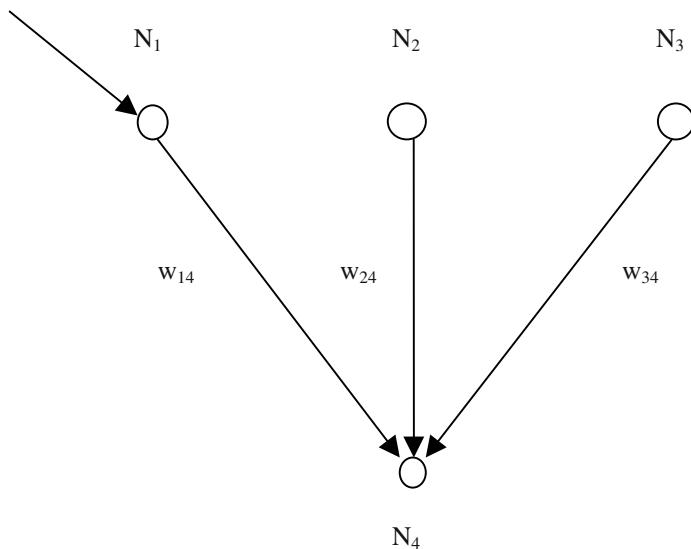
[**Hints:** Draw the truth table of a two-input NAND function. Plot the coordinates  $(0,0)$ ,  $(0, 1)$ ,  $(1,0)$  and  $(1,1)$  of the input variables on a graph

paper and draw a straight line to segregate the above co-ordinates into two classes depending on the truth value of the corresponding output variable, say  $y$ , Realize the straight line by a perceptron.]

3. Design a perceptron to classify the region satisfying the inequality:  $3x_1 + 5x_2 > 15$  from the remaining region in the real space.

[**Hints:** Follow the idea presented in section 8.3.2].

4. A fragment of a neural network comprising of 4 neurons is shown in Fig. 8.29.  $N_1, N_2$  and  $N_3$  are three neurons at the last but one layer, and neuron  $N_4$  is present in the output (last) layer. Given  $\text{Out}(N_1) = n_1(\text{say}) = 0.9$  units,  $\text{Out}(N_4) = n_4(\text{say}) = 0.5$ , error  $e$  at  $N_4 = 0.3$  units, weights  $w_{14} = 0.6$ ,  $w_{24} = 0.4$  and  $w_{34} = 0.7$  and learning rate  $\alpha = 0.03$ , update the value of  $w_{14}$  by Back-propagation algorithm. Also compute the back-propagated error at neuron  $N_1$ .



**Fig. 8.29:** A fragment of a neural net consisting of 4 neurons.

[**Hints:**  $\Delta w_{14} = \alpha n_4 (1 - n_4) e n_1$

$$= 0.03 \times 0.5 \times (1 - 0.5) \times 0.3 \times 0.9$$

Evaluate:  $w_{14}(\text{next}) = w_{14}(\text{current}) + \Delta w_{14}$ .

Back-propagated error at  $N_1$

$$\begin{aligned}
 &= e \times w_{14} \times n_1 \times (1 - n_1) \\
 &= 0.3 \times 0.6 \times 0.9 \times (1 - 0.9).]
 \end{aligned}$$

5. Volume of an object is a function of its radius and height. Given the following table of a number of cylinders. Show how many cluster centers should we select? Also indicate how to determine the mean and variance of each cluster center. How will you determine volume of the object using an RBF neural net if you know its radius and height?

Cylinder no.	Radius in cm	Height in cm	Volume
1	10	12	1200
2	11	12	1210
3	20	28	4000
4	21	28	4040
5	40	48	10,000
6	41	47	10,042
7	80	90	30,000
8	81	91	31,545
9	82	90	31,879
10	82	91	33,676

[**Hints:** There are 4 clusters: {1, 2}, {3, 4}, {5, 6}, {7, 8, 9, 10}. Determine the mean and variance over radius and heights for each cluster and call them the cluster centers. The RBF neural net in this case has 2 inputs: radius and height and only one output: volume.]

## References

- [1] Banerjee, R. K., Chowdhury, A. S., Chakraborty, I. and Konar, A., "Benchmark analysis for the path planning of a mobile robot using neural nets," *Proc. of the Int. Conference on Knowledge Based Computer Systems (KBCS-2002)*, Bombay, India, 2002.
- [2] Biswas, B. and Konar, A., "Speaker identification from voice using neural nets," *J. of Scientific and Industrial Research*, vol. 61, pp. 599-606, 2002.
- [3] Fiesler, E. and Beale, R. (Eds.), *Handbook of Neural Computation*, IOP and Oxford University Press, NY, 1997.

- [4] Jain, L. C., Halci, U., Hayashi, I., Lee, S. B. and Tsutsui, S. (Eds.), *Intelligent Biometric Techniques in Fingerprint and Face Recognition*, CRC Press, Boca Raton, Florida, 1999.
- [5] Jang, J.-S. R., Sun, C.-T. and Mizutani, E., *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*, Prentice-Hall, NJ, 1997.
- [6] Konar, A. and Jain, L. C., *Mobile Robots: An Experimental Approach*, World Scientific, Singapore, 2005(to appear).
- [7] Lau, C., *Neural Networks: Theoretical Foundations and Analysis*, IEEE Press, NY, 1992.
- [8] Leondes, C. T. (Ed.), *Neural Network Systems Techniques and Applications*, Academic Press, NY, 1998.
- [9] McCulloch, W. S. and Pitts, W., “A logical calculus of the ideas immanent in nervous activity,” *Bull. Math. Biophys.*, vol. 5, pp. 115-133, 1943.
- [10] Minsky, M., *Computation: Finite and Infinite Machines*, Prentice-Hall, NY, 1967.
- [11] Minsky, M. and Papert, S., *Perceptrons*, MIT Press, Cambridge, 1988.
- [12] Paul, B., Konar, A. and Mandal, A. K., “Fuzzy ADALINE for gray image recognition,” *Neurocomputing*, Elsevier, North Holland, vol. 24, pp. 207-223, 1999.
- [13] Rosenblatt, F., *The Perceptron: a perceiving and recognizing automation*, Technical Report 85-460-1, Cornell Aeronautical Laboratory, 1957.
- [14] Rosenblatt, F., *The perceptron: a probabilistic model for information storage in the brain*, Psych. Rev., vol. 65, pp. 365- 408, 1958.
- [15] Rumelhart, D. E., McClelland, J. L. and the PDP Research Group, *Parallel and Distributed Processing*, vol. I and II, MIT Press, Cambridge, 1986.
- [16] Schalkoff, R. J., *Artificial Neural Networks*, McGraw-Hill, 1997.
- [17] Widrow, B. and Hoff, M., *Adaptive Switching Circuits*, WESCON Convention Record, pp. 96-104, 1960.

- [18] Widrow, B. and Stearns, S. D., *Adaptive Signal Processing*, Prentice-Hall, NJ, 1985.
- [19] Widrow, B. and Winter, R., “Neural nets for adaptive filtering and adaptive pattern recognition,” *IEEE Computer*, pp. 25-39, 1988.

# 9

# Unsupervised Neural Learning Algorithms

*This chapter provides a thorough review of the classical algorithms on unsupervised neural learning. It begins with a brief introduction to recurrent neural topology and then presents in detail both binary and continuous Hopfield nets, their stability analysis and applications. The chapter also presents a detailed overview to adaptive resonance theory and its application in solving the ‘stability plasticity conflict’ problem in classical pattern recognition. Finally, the chapter introduces fuzzy associative memory neural nets and outlines algorithms for pattern classification by the proposed neural nets. Concluding remarks are listed at the end of the chapter.*

## 9.1 Introduction

In contrast to supervised learning, an unsupervised learning does not require a teacher; that is there is no target output. During the encoding phase, the neural network receives different excitations at its input, and organizes these received patterns into different categories. When a stimulus is later applied in the

---

recognition/application phase, the network responds by indicating the class to which the stimulus belongs. If the stimulus cannot be categorized among the known classes, a new class is formed to describe the non-association of the stimulus with the existing classes. The behavior of the neural network presented above can be illustrated with a typical object classification problem. Suppose, a person is asked to classify chairs, tables, desks, benches into four categories. He is given varieties of the above items. From the similarity of shape, size and utility, the person can easily classify these objects into four categories. Now, suppose he is given a duster. What will be the reaction of the person? The person will definitely say that the duster cannot be classified within the four classes. So, he will place the duster in a fifth category or class.

Though unsupervised learning does not involve a teacher, it requires some guidelines in determining features to form classes. For example, shape size, color, constituents, etc. may be used as features for classification of objects into several clusters. Thus without any knowledge of features, classification of objects into categories is not feasible.

Section 9.2 of the chapter introduces recurrent neural networks with special emphasis on Hopfield networks. The discussion on Hopfield nets includes their topology, stability analysis and applications. Section 9.3 provides an introduction bi-directional associative memory. In section 9.4, the principles of adaptive resonance theory have been introduced. Encoding and recall of fuzzy associative memory has been outlined in section 9.5. Conclusions are listed in section 9.6.

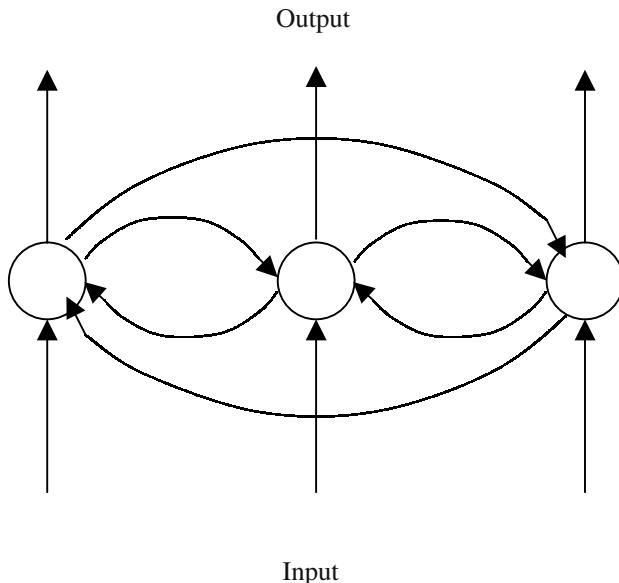
## **9.2 Recurrent Neural Nets**

Neural networks used for supervised learning in chapter 8 have a feed forward geometry, i.e., signals injected at the inputs can only traverse in the forward direction. A different type of neural nets that allow propagation and/or transfer of signal both in forward and backward directions is commonly used in many unsupervised learning algorithms. In these networks, signals are transferred in forward direction by a feed forward mechanism, whereas a reversal in signal flow is activated by a feedback mechanism.

Hopfield network [7] is an ideal member of the recurrent neural network family. In this network, any node  $j$  can receive signals from all other nodes  $i$ . Thus Hopfield network is a single layered network with exhaustive connectivity among the nodes. Both discrete and continuous Hopfield nets are prevalent in the current literature. Models of discrete and continuous Hopfield nets are presented below.

### 9.2.1 Discrete Hopfield Network

A discrete Hopfield network [7-8] is an auto-associative network, where the dynamics of the network is described by discrete state equations. Fig. 9.1 describes the topology of a Hopfield net. Given  $m$  pattern vectors, the components of which are assigned at the input of the discrete Hopfield net.



**Fig. 9.1:** A typical Hopfield neural net.

Let

- $w_{ji}$  be the connection weights from node  $i$  to node  $j$ ,
- $P_{i,p}$  be the  $i$ -th component of the input binary pattern vector  $\mathbf{P}$ , and
- $m$  be the total number of input patterns,
- $O_j(t)$  be the activation level of node  $j$  at time  $t$ ,
- $P_j$  is the  $j$ -th component of the input pattern, and
- $F_h$  is a hard-limiting synaptic function.

- **Encoding**

For storing  $m$ -patterns, an encoding rule is constructed as follows:

$$w_{ji} = \sum_{p=1}^m P_{j,p} P_{i,p}, \text{ for } i \neq j \\ = 0, \text{ for } i = j \quad \left. \right\} \quad (9.1)$$

- Recall

The following steps are executed in order to determine the closest stable pattern with respect to the supplied input excitation pattern  $P_j$ .

1. Assign  $O_j(t) = P_j$  at time  $t=0$
2. Compute  $O_j(t+1)$  by invoking expression (9.2)

$$O_j(t+1) = F_h \left( \sum_i w_{ji} O_i(t) \right) \quad (9.2)$$

where

$$\begin{aligned} F_h(a(t+1)) = & \begin{cases} 1, & \text{if } a(t+1) > 0 \\ -1, & \text{if } a(t+1) < 0 \\ a(t), & \text{if } a(t+1) = 0 \end{cases} \end{aligned} \quad (9.3)$$

3. Repeat step 2 until  $O_j(t+1) = O_j(t)$  for all nodes  $j$ . The states  $O_j$  of nodes  $j$  under this condition is said to be in equilibrium. The vector  $[...O_j...]_{(1 \times m)}$  for  $j=1$  to  $m$  thus obtained is the stored stable pattern that has the nearest match with the unknown input pattern.

Let  $\mathbf{X}_i$  be the  $m$ -dimensional bipolar patterns having components +1 to -1 to be stored in a discrete Hopfield net, and  $I_m$  is a  $(m \times m)$  identity matrix. The weight matrix for the Hopfield net can be computed by expression (9.4) [4].

$$\mathbf{W} = \sum_{i=1}^m (\mathbf{X}_i^T \mathbf{X}_i - I_m) \quad (9.4)$$

The above result directly follows from the encoding rule presented in expression (9.1).

**Example 9.1:** This example illustrates how we can construct the weight matrix for desired stable states of a system.

Let

$$\begin{aligned} \mathbf{X}_1 &= [ 1 \ -1 \ 1 ] \\ \mathbf{X}_2 &= [ -1 \ 1 \ -1 ] \\ \text{and } \mathbf{X}_3 &= [ -1 \ -1 \ 1 ] \end{aligned}$$

be three such stable states. The encode weight matrix  $\mathbf{W}$  for such system is computed [4] as

$$\mathbf{W} = (\mathbf{X}_1^T \mathbf{X}_1 - I_3) + (\mathbf{X}_2^T \mathbf{X}_2 - I_3) + (\mathbf{X}_3^T \mathbf{X}_3 - I_3)$$

$$= \begin{pmatrix} 0 & -1 & -1 \\ -1 & 0 & -1 \\ -1 & -1 & 0 \end{pmatrix}.$$

Now, suppose we want to verify that  $\mathbf{X}_1 = [1 \ -1 \ -1]$  is a stable state. For such verification, we can compute the following steps of the recall algorithm.

Here,  $\mathbf{O}(0) = [1 \ -1 \ -1]$ ,

$$\mathbf{O}(1) = F_h(\mathbf{X}_1 \mathbf{W}) = F_h([2 \ 0 \ 0]) = [1 \ -1 \ -1].$$

The above result clearly shows that  $\mathbf{O}(1) = \mathbf{O}(0)$ , and thus  $\mathbf{X}_1 = [1 \ -1 \ -1]$  is a stable state.

Now, suppose we excite the network with an unknown state  $\mathbf{X}_4 = [+1 \ +1 \ -1]$ . We are interested to see where the network settles down.

Here,  $F_h(\mathbf{X}_4 \mathbf{W}) = F_h[0 \ 0 \ -2] = [1 \ 1 \ -1] = \mathbf{X}_4$ .

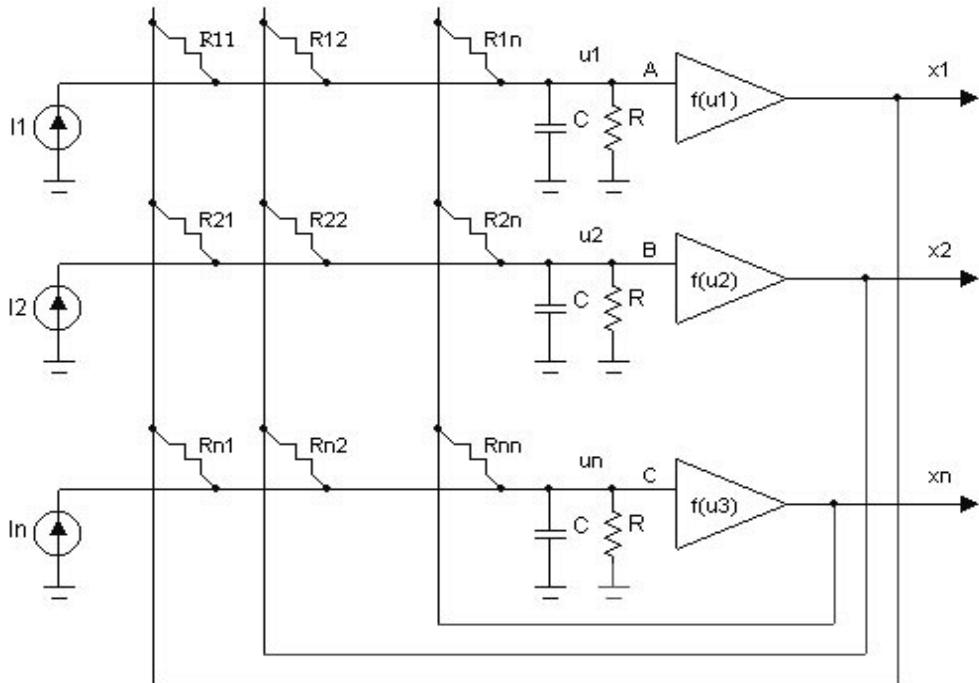
The above result shows that  $\mathbf{X}_4$  too is a stable state. It is thus interesting to note that the system may have more stable states than those encoded by the weight matrix.

One important observation [4] about Hopfield net is that the network having  $n$  nodes has a capacity to store  $O(n)$  patterns, rather than  $O(2^n)$  patterns. The second observation about the network is that the network has a tendency to stabilize at local minima. This is good for pattern association, but disadvantageous for optimization applications.

### 9.2.2 Continuous Hopfield Neural Net

A continuous Hopfield neural net [8] is a dynamical system that can be described by a set of coupled non-linear differential equations. It is well-known that a dynamical system can be realized in different forms, depending on the nature of energy by which the system is excited. For example, the motion of a car can be described by a differential equation, where the mechanical power generated by the engine is utilized to keep the car in motion. Similarly, current or voltage excitations are applied at the input of a dynamical electric circuit. Besides mechanical and electrical, the dynamical system may also employ other forms of energy such as light or heat, depending on the domain of its

application. In this section, we consider an electrical realization of a continuous Hopfield net dynamics (vide Fig. 9.2).



**Fig. 9.2:** Electrical realization of a continuous Hopfield network

To obtain the dynamics of Fig 9.2, we first apply Kirchoff's current law (KCL) at junction A. Thus we have:

$$I_1 + \frac{x_1 - u_1}{R_{11}} + \frac{x_2 - u_1}{R_{12}} + \dots + \frac{x_n - u_1}{R_{1n}} = \frac{u_1}{R} + C \frac{du_1}{dt} \quad (9.5)$$

where  $u_1, u_2, \dots, u_n$  are the potential at the input of the amplifiers respectively. In the last expression, we used  $u_1$  only, but  $u_2, u_3, \dots, u_n$  are needed in the differential equation models of the network.

Defining

$$w_{ij} = \frac{1}{R_{ij}} \quad \text{for all } i, j \quad (9.6)$$

$$\beta_i = \sum_{j=1}^n \frac{1}{R_{ij}} + \frac{1}{R} \quad \text{for } i = 1, 2, 3, \dots, n \quad (9.7)$$

and  $\theta_i = I_i$ ,

we re-write equation (9.5) as follows :

$$C \frac{du_1}{dt} = \sum_{j=1}^n w_{1j} x_j - \beta_1 u_1 + \theta_1 \quad (9.8)$$

For the  $i$ -th module in the network, we thus have the following differential equation:

$$C \frac{du_i}{dt} = \sum_{j=1}^n w_{ij} x_j - \beta_i u_i + \theta_i \quad (9.9)$$

Thus for an  $n$ -module network like the one shown in Fig. 9.2, we have  $n$ -differential equations of the form (9.10), where  $1 \leq i \leq n$ .

Hopfield employed a specialized Lyapunov energy function to determine the stable points of the dynamics represented by equation (9.9). Let  $E$  be an energy function, given by

$$E(t) = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} x_j x_i + \sum_{i=1}^n \beta_i \int_0^{x_i} f^{-1}(\xi_i) d\xi_i - \sum_{i=1}^n \theta_i x_i \quad (9.10)$$

where  $f(u_i) = x_i$  and thus  $u_i = f^{-1}(x_i)$

$$\begin{aligned} \frac{dE}{dt} &= -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \left( w_{ij} x_j \frac{dx_i}{dt} + w_{ij} x_i \frac{dx_j}{dt} \right) + \sum_{i=1}^n \beta_i u_i \frac{dx_i}{dt} \\ &\quad - \sum_{i=1}^n \theta_i \frac{dx_i}{dt} \end{aligned} \quad (9.11)$$

With symmetric weights, i.e.  $w_{ij} = w_{ji}$ , we find

$$\begin{aligned}
 \frac{dE}{dt} &= - \sum_{i=1}^n \frac{dx_i}{dt} \left[ \left( \sum_{j=1}^n w_{ij} x_j \right) - \beta_i u_i + \theta_i \right] \\
 &= - C \sum_{i=1}^n \left[ \frac{dx_i}{dt} \right] \left[ \frac{du_i}{dt} \right] \\
 &= - C \sum_{i=1}^n \left[ \frac{dx_i}{dt} \right] \frac{d}{dt} (f^{-1}(x_i)) \\
 &= - C \sum_{i=1}^n \frac{\partial f^{-1}(x_i)}{\partial x_i} \left[ \frac{dx_i}{dt} \right]^2. \tag{9. 12}
 \end{aligned}$$

Thus if  $\frac{\partial f^{-1}}{\partial x_i} > 0$ , then  $\frac{dE}{dt} < 0$ , and consequently the continuous Hopfield net has a stable dynamics. But what does  $\frac{\partial f^{-1}}{\partial x_i} > 0$  mean? It means that  $f^{-1}(x_i)$  is a monotonically increasing function of  $x_i$ .

The clustering process by the continuous Hopfield net is outlined below:

1. Assign random connection weights  $w_{ji} = w_{ij} = +1$  or  $-1$  for all  $i \neq j$  and  $0$  for all  $i=j$ .
2. Submit  $\theta_i \leftarrow P$  for an unknown pattern  $P$ ; Initialize  $u_i \leftarrow 0$  for all  $i$  and  $x_j \leftarrow 0$  for all  $j$ ; Compute  $\Delta u_i = (1/C) (\theta_i) \Delta t$  where  $\Delta t = 1$ , say.
3. Set  $u_i \leftarrow u_i + \Delta u_i$  where  

$$\Delta u_i \leftarrow (1/C) \left( \sum_{j=1}^n w_{ij} x_j - \beta_i u_i + \theta_i \right) \Delta t$$

and  $x_j = f(u_j)$  for all  $i, j \in [1, n]$ .

4. Repeat step 3 until  $u_i$  for all  $i$  converges, i.e.,  $|\Delta u_i| \leq \delta$ , where  $\delta$  is a pre-assigned positive number, however small.

Any unknown pattern  $P$  thus can be classified to one fixed point in the state-space by invoking the learning algorithm. Since the algorithm does not ensure convergence to global stable points, Genetic Algorithm (GA), vide Chapter 12, may be employed for the adjustment of the weight matrix  $W$  of the Hopfield net to minimize energy function  $E(t)$ . It is to be noted that for setting  $w_{ij} = w_{ji}$  and  $w_{ii} = 0$ , we have to adjust  $w_{ij}$  for  $i < j$  only, and then we can automatically ensure  $w_{ij} = w_{ji}$ . In using GA, we can use  $E(t)$  as the fitness function. Further, assuming gain of the amplifier to be sufficiently high, we can ignore the integral term in (9.10). Thus, the fitness function is simplified to

$$E(t) = -(\frac{1}{2}) \sum_{i=1}^n \sum_{j=1}^n w_{ij} x_j x_i - \sum_{i=1}^n \theta_i x_i \quad (9.13)$$

It is indeed important to note that given  $\theta_i$  for  $i=1, 2, \dots, n$ ,  $E(t)$  cannot be minimized just by adjusting  $w_{ij}$ , as  $x_j$ 's are unknown. Thus we need to describe (9.13) as a function of  $\theta_i$  and  $w_{ij}$  only. This can be accomplished by solving equation (9.14) for known  $\theta_i$ .

$$C \frac{du_i}{dt} = - \sum_{j=1}^n w_{ij} f(u_j) - \beta_i u_i + \theta_i \quad (9.14)$$

Once  $u_i$  for  $i=1$  to  $n$  are evaluated, we will substitute  $x_j = f(u_j)$  in (9.13). Consequently,  $E(t)$  in (9.13) is only a function of  $w_{ij}$  and  $\theta_i$ , and thus may be used as a suitable fitness function in the Genetic Algorithm for  $E$  minimization.

### 9.2.3 A Near-Equilibrium Consideration for Classification by Continuous Hopfield Nets

Under equilibrium condition,  $\frac{du_i}{dt} = 0$ . Consequently, equation (9.9) reduces to

$$C \frac{du_i}{dt} = 0 = \sum_{j=1}^n w_{ij} x_j - \beta_i u_i + \theta_i$$

$$\Rightarrow u_i = (1/\beta_i) [\sum_{j=1}^n w_{ij} x_j + \theta_i] \quad (9.15)$$

$$\Rightarrow x_i = f(u_i) = f \left[ \left( \sum_{j=1}^n w_{ij} x_j + \theta_i \right) / \beta_i \right] \quad (9.16)$$

Given an initial state  $x_i(0)$ , we can easily determine the neighboring equilibrium state by executing the (9.15) and (9.16) in turn until convergence in  $x_i$  for  $i=1,2,\dots,n$  takes place. A simplified algorithm for approxiamte classifaction of patterns by continuous Hopfield net is outlined below:

1. Assign random weights  $w_{ij} = +1$  or  $-1$  and  $w_{ii} = 0$
2. Initialize  $x_i(0) = 0$  for all  $i = 1, 2, \dots, n$ .
3. Compute  $x_i(t+1) = f \left[ \left( \sum_j w_{ij} x_j + \theta_i \right) / \beta_i \right]$  for all  $i=1, 2, 3, \dots, n$ . Here  $\theta_i$  is the input pattern.
4. Repeat step 3 until  $|x_i(t+1) - x_i(t)| < \delta$  where  $\delta$  is a small positive number.

### 9.2.4 Optimization Using Hopfield (Continuous) Neural Nets

Many optimization problems can be repesented by an energy-like function and solved by comparing the energy-like function of the problem with that of the Hopfield Net. To illustrate the idea of problem formulation, let us consider the well-known Travelling Salesman Problem (TSP). The problem is stated as follows. Given a set of cities and the distance between any two cities within the set, one has to determine the shortest tour that visits each city only once and returns to the starting point. It is easy to show that for  $n$  cities, there are  $n!$  possible tours. Determination of the shortest tour consequently requires evaluation of the distance of traversal for all  $n!$  tours. To solve this problem, we consider a Hopfield net having  $n^2$  units, represented by an  $(n \times n)$  array. The energy-like function  $E$  used for the TSP problem has 4 terms  $E_1, E_2, E_3$  and  $E_4$ , where

$$E = \alpha_1 E_1 + \alpha_2 E_2 + \alpha_3 E_3 + \alpha_4 E_4 \quad (9.17)$$

where  $\alpha_1, \alpha_2, \alpha_3$  and  $\alpha_4$  are normalizing constants.

Let

$O_{xi}$  be the activation level of a unit denoting that city  $x$  is the  $i$ -th city in the tour, and

$d_{xy}$  be the distance between cities  $x$  and  $y$ .

Now, to construct the terms  $E_1$ ,  $E_2$ ,  $E_3$  and  $E_4$  in (9.17), we consider an example of 5 cities. Let the 5 cities be denoted by  $C_1$ ,  $C_2$ ,  $C_3$ ,  $C_4$  and  $C_5$  and possible tour :  $C_3 \rightarrow C_1 \rightarrow C_5 \rightarrow C_4 \rightarrow C_2$  is represented by a matrix  $O$  of  $(5 \times 5)$  dimension.

$$\text{city} \quad \begin{matrix} & \text{order of visit} \longrightarrow \\ \downarrow & \end{matrix}$$

$$O = \left[ \begin{array}{ccccc} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{array} \right] \quad (9.18)$$

Following the definition of  $O_{xi}$ ,  $O_{31} = 1$  in (9.18) indicates that  $C_3$  is the first city in the tour. The significance of other elements in the matrix thus can be visualized easily.

The constraints used in constructing the terms  $E_1$  through  $E_4$  of the energy function are presented below.

- ◆ **Constraint 1:** Each city in a tour should included only once. This has been realized by setting only one element in each row of  $O$ -matrix to be 1 and all other elements in the same row to be 0. This can be expressed mathematically as to minimize  $E_1$ , where

$$E_1 = \sum_x \sum_i \sum_{i \neq j} O_{xi} O_{xj}. \quad (9.19)$$

- ◆ **Constraint 2:** The salesperson cannot visit more than one city at a particular time. This has been realized by setting only one element in a column to be 1, and the remaining elements in the column to be 0. Mathematically, the constraint can be expressed as to minimize  $E_2$  where

$$E_2 = \sum_i \sum_x \sum_{x \neq y} O_{xi} O_{yi}. \quad (9.20)$$

- ◆ **Constraint 3:** For  $n$ -cities there should be exactly  $n$  1's in the matrix  $O$ . This can be formalized as to minimize  $E_3$ , where

$$E_3 = \{ (\sum_x \sum_i \mathbf{O}_{xi}) - n \}^2. \quad (9.21)$$

◆ **Constraint 4:** This constraint ensures that the shortest tour should be favoured. Mathematically, this can be formatted as to minimize  $E_4$ , where

$$E_4 = \sum_x \sum_{x \neq y} \sum_i d_{xy} \mathbf{O}_{xy} (\mathbf{O}_{y,i+1} + \mathbf{O}_{y,i-1}). \quad (9.22)$$

Solving normalizing constraints in (9.17) as

$$\alpha_1 = A/2 \quad (9.23)$$

$$\alpha_2 = B/2 \quad (9.24)$$

$$\alpha_3 = C/2 \quad (9.25)$$

and  $\alpha_4 = D/2 \quad (9.26)$

we can compare the energy-function  $E$  of (9.17) with the typical energy function of a Hopfield net with the integral term in (9.10) being replaced by  $u_i$ . Consequently, the revised form of (9.10) is given by

$$E = -(\frac{1}{2}) \sum_i \sum_j w_{ij} x_i x_j + \sum_i \beta_i u_i - \sum_i \theta_i x_i. \quad (9.27)$$

On comparison, the weight  $w_{xi,yj}$  is formed to be

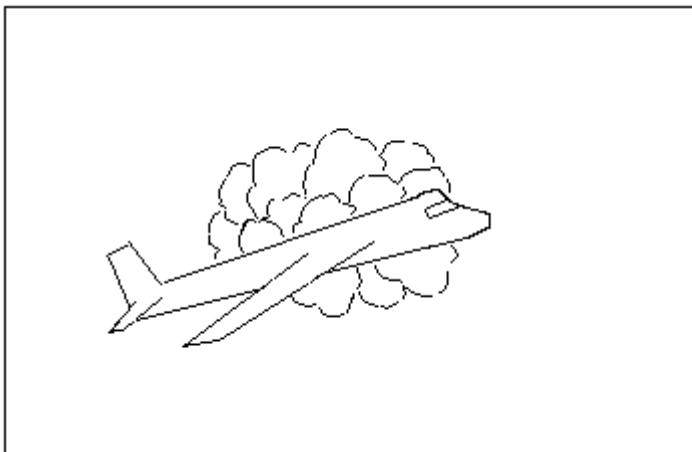
$$w_{xi,yj} = -A\delta_{xy}(1 - \delta_{ij}) - B\delta_{ij}(1 - \delta_{xy}) - C - Dd_{xy}(\delta_{j,i+1} + \delta_{j,i-1}) \quad (9.28)$$

where  $\delta_{ij} = 1$  for  $i = j$  and  $\delta_{ij} = 0$  otherwise.

Ideally the shortest tour corresponds to the global minima in  $E$  (vide 9.17). This calls for adjustment of the parameters:  $A$ ,  $B$ ,  $C$  and  $D$ , so that  $w_{xi,yj}$  for the minimum  $E$  can be evaluated. A Genetic Algorithm may be invoked to handle the above problem.

## 9.2.5 Application of Hopfield Neural Nets in Engineering Problems

It is clear from the discussions on Hopfield nets that Hopfield nets can be employed to determine the stable states from noisy nearby initial states. Consider for instance image of an aircraft occluded partially by cloud (Fig. 9.3). Suppose, we have to determine the shape of the complete aircraft from the given image [9].



**Fig. 9.3:** An aircraft having the left wing occluded by cloud.

If the length of the aircraft can be normalized with reference to the known sample image, then recovering the aircraft from its partial image is simple. For simplicity, suppose normalization of length of the aircraft is feasible online. To implement the recovery of the aircraft, we first construct a weight matrix  $\mathbf{W}$  of the Hopfield net. Since the image may have non-uniform brightness because of cloud, we first determine the edge of the image using Sobel mask [5]. The image now is scanned row-wise, and a vector  $\mathbf{X}_1$  is constructed by concatenating the rows one after another. The  $\mathbf{W}$  matrix now is computed by,

$$\mathbf{W} = \mathbf{X}_1^T \mathbf{X}_1 - \mathbf{I} \quad (9.29)$$

If we have  $n$  different views of the training sample, then we can construct image vectors  $\mathbf{X}_2, \mathbf{X}_3, \dots, \mathbf{X}_n$  likewise and then

$$\mathbf{W} = \sum_{i=1}^n (\mathbf{X}_i^T \mathbf{X}_i - \mathbf{I}). \quad (9.30)$$

The matrix thus obtained represents an encoded information about the sample aircraft. Now, to retrieve the aircraft from the given image occluded by cloud, we construct an image vector for Fig. 9.3 following the method we adopted to construct  $\mathbf{X}_1$ . Let the vector for the given image be  $\mathbf{X}$ . Now, to compute the output vector of discrete Hopfield net, we set:

$$\mathbf{O}(0) \leftarrow \mathbf{X}; t \leftarrow 0;$$

$$\mathbf{O}(1) = F_h(\mathbf{O}(0) \mathbf{W});$$

If  $\mathbf{O}(t+1) \neq \mathbf{O}(t)$ , we iterate  $\mathbf{O}(t+1) = F_h(\mathbf{O}(t)\mathbf{W})$  until  $\mathbf{O}(t+1) = \mathbf{O}(t)$ .

When the terminating condition is reached, we re-construct the image matrix of dimension  $(p \times m)$  from  $\mathbf{O}(t+1)$ , by transferring each consecutive  $m$  numbers from  $\mathbf{O}(t+1)$  to each row of a 2-D array  $\mathbf{M}$ ; until all  $p$ -rows are constructed. The image matrix  $\mathbf{M}$  thus retrieves the complete shape of the aircraft from its occluded view.

It is clear from the discussions on section 9.2.4 and 9.2.5 that Hopfield net can be used for learning as well as optimization problems. For applications in learning problems, we should have some knowledge about the sample data, configured as stable states of the problem. These stable states may subsequently be used to encode the weight matrix  $\mathbf{W}$  for the Hopfield net. After a weight matrix is evaluated, the recall algorithm presented in section 9.2.1 may be employed to determine the nearest stable point from an unknown problem state.

The example we considered so far demonstrates the application of a discrete Hopfield net. The same problem can be easily handled using a continuous Hopfield net as well. A brief outline of the scheme is presented below.

To start with we consider expression (9.9) as the primitive model for the problem. For  $i = 1$  to  $n$ , we have  $n$  expressions like (9.9). Combining them we form a state equation given by

$$C \frac{d\mathbf{U}}{dt} = \mathbf{W} \mathbf{X} - \boldsymbol{\beta} \mathbf{U} + \boldsymbol{\theta} \quad (9.31)$$

$$\text{where } \mathbf{U} = [u_1 \ u_2 \ \dots \ u_n]^T \quad (9.32)$$

$$\mathbf{W} = [w_{ij}]_{n \times n} \quad (9.33)$$

$$\mathbf{X} = [x_1 \ x_2 \ \dots \ x_n]^T \quad (9.34)$$

$$\boldsymbol{\beta} = \text{Diag } [\beta_1 \ \beta_2 \ \dots \ \beta_n] \text{ is a diagonal matrix} \quad (9.35)$$

$$\boldsymbol{\theta} = [\theta_1 \ \theta_2 \ \dots \ \theta_n]^T. \quad (9.36)$$

Further, assuming  $f(u_i)$  very large and  $f'(\xi_i)$  close to zero in Equation (9.10), we re-write it as follows:

$$E(t) = -(\frac{1}{2}) \sum_i \sum_j w_{ij} x_i x_j - \sum_i \theta_i x_i. \quad (9.37)$$

Equation (9.37) can be written in vector-matrix form as

$$E(t) = -(\frac{1}{2}) \mathbf{X}^T \mathbf{W} \mathbf{X} - \boldsymbol{\theta}^T \mathbf{X}. \quad (9.38)$$

Now, taking a look at (9.31), we compute the steady-state value of  $\mathbf{X} = \mathbf{X}^*$  by

setting  $\frac{d\mathbf{U}}{dt} = 0$ . This yields

$$\mathbf{W} \mathbf{X}^* = \boldsymbol{\beta} \mathbf{U}^* - \boldsymbol{\theta}^* \quad (9.39)$$

where “\*” above a parameter denotes its steady-state values. Given  $\mathbf{X} = f(\mathbf{U})$ , we now re-write (9.39) as

$$\mathbf{W} f(\mathbf{U}^*) = \boldsymbol{\beta} \mathbf{U}^* - \boldsymbol{\theta}^*$$

$$\text{or, } \boldsymbol{\theta}^* = \boldsymbol{\beta} \mathbf{U}^* - \mathbf{W} f(\mathbf{U}^*) \quad (9.40)$$

Thus substituting  $\boldsymbol{\theta}^*$  from (9.40) and  $\mathbf{X} = f(\mathbf{U})$ , we, vide (9.38) find

$$E = -(\frac{1}{2}) f^T(\mathbf{U}) \mathbf{W} f(\mathbf{U}) - (\boldsymbol{\beta} \mathbf{U}^* - \mathbf{W} f(\mathbf{U}^*))^T f(\mathbf{U}^*). \quad (9.41)$$

We now need to select a  $\mathbf{W}$  that minimizes  $E$ . This can be accomplished by a Genetic Algorithm with  $E$  as the fitness function.

Now, for a particular application, given  $\mathbf{X} = f(\mathbf{U})$  and  $\mathbf{U} = f^{-1}(\mathbf{X})$ , we find a  $\mathbf{W} = \mathbf{W}^*$  that minimizes  $E$ . A solution to (9.31) for a known  $\mathbf{X}$  and  $\boldsymbol{\theta} = 0$  may then be computed and the result  $\mathbf{U} = \mathbf{U}^*$ , the steady-state value of  $\mathbf{U}$ , is then substituted to compute  $\mathbf{X}^*$  by

$$\mathbf{X}^* = \mathbf{W}^{-1} (\boldsymbol{\beta} \mathbf{U}^* - \boldsymbol{\theta}^*) \quad (9.42)$$

with  $\boldsymbol{\theta} = \boldsymbol{\theta}^* = 0$ .

For a given image vector  $\mathbf{X}$  we thus can compute  $\mathbf{W}^*$ , and for a similar  $\mathbf{X}$  vector, we can identify the stable point  $\mathbf{X}^*$  by (9.42). When a number of image vectors  $\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n$  are available, we determine a  $\mathbf{W} = \mathbf{W}^*$  that minimizes

$$\sum_{i=1}^n E(\mathbf{X}_i) = \sum_{i=1}^n [-(\frac{1}{2}) f^T(\mathbf{U}_i) \mathbf{W} f(\mathbf{U}_i) - (\boldsymbol{\beta} \mathbf{U}_i^* - \mathbf{W} f(\mathbf{U}_i^*))^T f(\mathbf{U}_i^*)]. \quad (9.43)$$

Once a weight matrix  $\mathbf{W}=\mathbf{W}^*$  is evaluated, we can determine the nearest stable state for a given noisy state. Thus determination of the complete shape of an object, such as an aircraft, from its partially occluded image is feasible.

### 9.2.6 Boltzman Machines

A negative rate of change of energy  $dE/dt$  in Hopfield nets indicates that the dynamical systems has one or more stable points, but it fails to ensure that the dynamics get trapped at global minima. This phenomenon is useful in pattern association/clustering applications but it is undesirable for optimization or constraint satisfaction applications. *Simulated Annealing*, a technique that simulates the annealing process of metals, may be incorporated in a Hopfield net to find the global minima in the energy surface. Such systems are well-known as *Boltzmann machines*. In Boltzmann machines, local minima are avoided by adding randomness to the process of energy minimization [4]. Thus when the network approaches to a local minimum, it at least gets a chance to escape from getting trapped at the local minimum. A Boltzman Machine can be empowered with the above capability by incorporation of a stochastic rule for updating the binary states of the neurons. The probability that a neuron  $j$  will be active, denoted by  $p_j$ , is given by

$$p_j = 1 / [ 1 + \exp ( -\Delta E_j / T ) ] \quad (9.44)$$

where

$\Delta E_j$  = the total input received by neuron  $j$

and  $T$  = temperature of the network.

Under thermal equilibrium, the probability that a neuron will possess a global state is constant and obeys the Boltzmann distribution:

$$p_A / p_B = \exp [ - ( E_A - E_B ) / T ] \quad (9.45)$$

which indicates that the probability ratio of two states depends on their difference of energy.

Boltzman machines usually undergo a process of annealing, i.e., the temperature gradually decreases from a very high value. As the temperature settles down, the network also looses randomness of states, and at very low temperature it behaves like a Hopfield net.

To train Boltzmann machines, one set of neurons is selected as input, while the remaining neurons as outputs. The weight adaptation policy is given by

$$\Delta w_{ij} = \epsilon ( p_{ij}^+ - p_{ij}^- ) \quad (9.46)$$

where

$p_{ij}^+$  = probability that neurons i and j are both active at thermal equilibrium, when both input and output vectors are clamped,

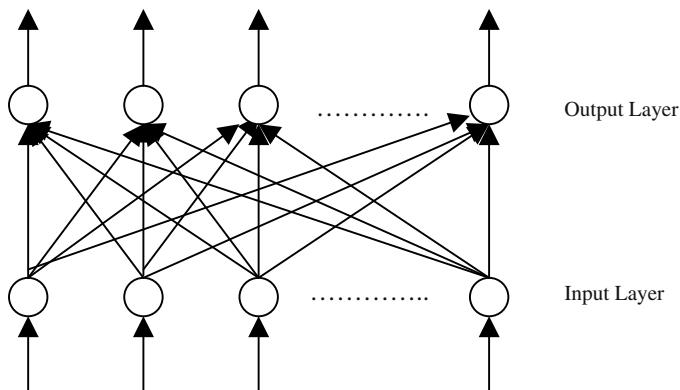
$p_{ij}^-$  = probability that neurons i and j are both active at thermal equilibrium, when only input vector is clamped and the network is free-running,

and  $\epsilon$  = a constant.

For a very small value of  $\epsilon$ , the weight updating algorithm performs like a gradient descent leaning [6].

### 9.3 Bidirectional Associative Memory

Bart Kosko in late 80's extended the concept of auto-association of neurons to association of neurons between two layers. Bidirectional Associative Memory (BAM) [10-11] is a two-layered recurrent network, having lateral feedback between neurons of the two layers. A typical bidirectional associative memory is shown in Fig. 9.4.



**Fig. 9.4:** A bidirectional associative memory.

Let

$\mathbf{P}$  be a input vector,

$P_{i,p}$  be the i-th component of the pattern vector  $\mathbf{P}$ ,

$Q_{j,q}$  be the j-th component of the pattern vector  $\mathbf{Q}$ , where p and q form an association pair, and

$w_{ji}$  = the connection weight from i-th to j-th neuron.

- **Encoding**

The encoding model is then given by

$$w_{ji} = \sum_{p=1}^m \sum_{q=1}^m P_{ip} Q_{jq} \quad (9.47)$$

where  $m$  denotes the number of pair of instances.

Bart Kosko has shown that if the forward weight matrix is  $\mathbf{W}$ , then the reverse weight matrix is given by  $\mathbf{W}^T$ , where  $T$  is the transposition operator.

- **Recall**

The recall process of a BAM consists of the following steps:

♦ Initialization: Set  $O_i(0) = P_i$ , where  $O_i(0)$  is the activation level of neuron  $i$  at time  $t=0$ , and  $P_i$  is the  $i$ -th component of the input pattern.

♦ Update:  $O_j(t+1) = F_h(\sum_i w_{ji} O_i(t))$  where  $O_j(t+1)$  is the activation level at neuron  $j$  at time  $(t+1)$ , and  $F_h$  is a nonlinear hard inhibiting function.

$$F_h(a(t+1)) = 1, \quad \text{if } a(t+1) > 0$$

$$= -1, \quad \text{if } a(t+1) < 0$$

$$= a(t), \quad \text{if } a(t+1) = 0$$

♦ Repeat the last step until  $O_j(t+1) = O_j(t)$  for all  $j$ . The pattern at the output layer then is considered to have an association with the input pattern  $P$ .

**Example 9.2:** Given input-output examples patterns given by  $\mathbf{A}_i - \mathbf{B}_i$  for  $i = 1$  to 3. Let

$$\mathbf{A}_1 = [1 \ -1 \ 1] \quad \text{and} \quad \mathbf{B}_1 = [1 \ -1]$$

$$\mathbf{A}_2 = [-1 \ -1 \ 1] \quad \text{and} \quad \mathbf{B}_2 = [-1 \ 1]$$

$$\mathbf{A}_3 = [1 \ 1 \ 1] \quad \text{and} \quad \mathbf{B}_3 = [1 \ 1].$$

The encoded  $\mathbf{W}$  matrix in this case is given by

$$\mathbf{W} = \sum_{i=1}^3 \mathbf{A}_i^T \mathbf{B}_i$$

$$= \begin{pmatrix} 3 & -1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix}.$$

Now, suppose  $\mathbf{A}_1 = [1 \ -1 \ 1]$  is given, we check whether we can retrieve  $\mathbf{B}_1$  from  $\mathbf{A}_1$  and  $\mathbf{W}$ .

To verify this, we compute

$$\begin{aligned} F_h(\mathbf{A}_1 \mathbf{W}) &= F_h([3 \ -1]) \\ &= [1 \ -1] \\ &= \mathbf{B}_1 \end{aligned}$$

So, we can correctly obtain  $\mathbf{B}_1$  from  $\mathbf{A}_1$  and encoded matrix  $\mathbf{W}$ . Now, let us check, whether we can obtain  $\mathbf{A}_2$  from  $\mathbf{B}_2$ . In this case, we compute:

$$F_h(\mathbf{B}_2 \mathbf{W}^T) = F_h([-4 \ 0 \ 0]).$$

Let the last outputs of the neurons in the input layer be the vector  $\mathbf{A}_3 = [1 \ 1 \ 1]$ . Then

$$F_h([-4 \ 0 \ 0]) = [-1 \ 1 \ 1].$$

Since  $[-1 \ 1 \ 1]$  is none of the given  $\mathbf{A}_i$ 's, we, iterate:

$$\begin{aligned} F_h([-1 \ 1 \ 1] \mathbf{W}) \\ &= F_h([-1 \ 3]) \\ &= [-1 \ 1] \\ &= \mathbf{B}_2 \end{aligned}$$

It is indeed important to note that the BAM system can correctly retrieve  $\mathbf{B}_1$  from  $\mathbf{A}_1$ . But while retrieving  $\mathbf{A}_2$  from  $\mathbf{B}_2$ , it moves to a new state  $[-1 \ 1 \ 1]$ .

Unfortunately, the last state does not match with any of the existing  $A_i$ 's. So, we checked the possibility of retrieving  $B_2$  from this new state. The answer, however, is favorable.

## 9.4 Adaptive Resonance Theory

Most of the pattern mapping neural nets suffer from the drawbacks that during learning of weights, the weight matrix tends to encode the presently active pattern, thus weakening the trace of patterns it had already learnt. The other problem that the common type of neural nets face is the forceful categorization of a new pattern to one of the already learnt classes. On occasions such categorization seems to be ridiculous as the nearest class of the current pattern may be significantly different with respect to the centre of the class (or cluster center). The problems of the lack of stability of the weight matrix and forceful categorization of a new pattern to one of existing classes, led to the proposal of a new architecture for pattern classification. This architecture is designed using *Adaptive Resonance Theory* (ART) [1-2] that takes into account of the so-called *stability-plasticity dilemma* in pattern recognition.

A minimal ART network consists of two layers  $F_1$  and  $F_2$ .  $F_1$  is the input layer and  $F_2$  is the output layer and there exists feedforward connections from  $F_1$  to  $F_2$  and feedback connections from  $F_2$  to  $F_1$ . The number of units or neurons in the  $F_2$  layer represents the different classes. When a input pattern  $a_l$  is applied to the  $F_1$  layer, the ART network determines the winning neuron in the  $F_2$  layer through a competition.

The selected winning neuron in the  $F_2$  layer then generates a feedback signal, which is compared with the input pattern  $a_l$ . If the two vectors match well, then the winning neuron in the  $F_2$  layer determines the category of the input pattern. Under this case both the top-down and bottom-up weights between the layers  $F_1$  and  $F_2$  are to be adjusted to reinforce the input pattern. If the match between the input pattern and the back pattern is poor then winning neuron in  $F_2$  layer is a wrong selection. This neuron is removed from the set of possible winners, and the competitive process of selection of a winner in  $F_2$  layer is continued until a suitable one with acceptable matching score is identified. If no neuron in  $F_2$  layer with acceptable matching score is available, then a new neuron, describing a new category of the input pattern is created.

The search process in an ART network is controlled by two subsystems:

- i) orienting subsystem,

and ii) attentional subsystem.

The orienting subsystem computes a dimensionless vigilance parameter that gives a measure of the match between the input pattern and the exemplar pattern of the chosen category. The attentional system includes a gain control process

that allows the neurons in the  $F_1$ -layer to be engaged only when an input pattern is present, and it also regulates the learning [3].

In an ART network information of neurons reverberates back and forth between the layers  $F_1$  and  $F_2$ . Once a winning neuron in the  $F_2$  layer is correctly identified, the neural network is said to be in resonance. Both feed forward and feedback weights in the network are adjusted during the period of resonance. No learning takes place in the ART network before the period of resonance is attained.

Both discrete and continuous ARTs are prevalent in the current literature [1-2]. An algorithm to the binary ART is outlined below:

Let

$w_{ji}$  = the weight from the  $i$ -th unit (neuron) in the  $F_1$  layer to  $j$ -th unit in the  $F_2$  layer,

$\mathbf{W}_j$  = the weight vector leading to the  $j$ -th neuron in the  $F_2$  layer from all neurons in the  $F_1$  layer,

$v_{ij}$  = the weight from the  $j$ -th unit in the  $F_2$  layer to the  $i$ -th unit in the  $F_1$  layer,

$\mathbf{V}_j$  = the weight vector from the  $j$ -th unit in the  $F_2$  layer to all neurons in the  $F_1$  layer.

- 1.a) Initialize  $w_{ji}$  for all  $j, i$  to random values in  $[0,1]$ . Initialize  $v_{ij} = 1$ , for all  $i, j$ . This enables the uncommitted units in the  $F_2$  layer also to compete in the same way as the learned units.
- 1.b) Enable all the neurons (units) in the  $F_2$  layer.
2. For a input binary pattern  $\mathbf{a}$  to  $F_1$  layer, determine the winning neuron  $k$  in the  $F_2$  layer by evaluating

$$k = \arg \max_j [\mathbf{W}_j^T \mathbf{a}] \quad (9.48)$$

3. A similarity measure between the winning prototype  $\mathbf{V}_k$  and the input  $\mathbf{a}$  is computed and compared with a vigilance parameter  $\rho$ , where  $\rho \in (0,1)$ . The similarity measure determines the fraction of bits of  $\mathbf{a}$  that are also present in  $\mathbf{V}_k$ . The condition thus is

$$\frac{\mathbf{V}_k^T \mathbf{a}}{\sum_{i=1}^M a_i} \geq \rho \quad (9.49)$$

Three possibilities now may arise:

- a) If the prototype associated with the winning neuron satisfies (9.49) then go to step 4 for weight adaptation.
  - b) If the vigilance test criteria (9.49) fails, then the k-th output is disabled and another winning neuron is selected by repeating steps 2 and 3.
  - c) If no neurons in  $F_2$  layer passes the vigilance test, then a new neuron is created at  $F_2$  layer, and its corresponding prototype vector  $\mathbf{V}_k$  is set equal to the input pattern  $\mathbf{a}$ , i.e.,  $\mathbf{V}_k = \mathbf{a}$ .
4. The following weight adaptation equations are used for updating the weights:

$$v_{ik}(t+1) = v_{ik}(t) \wedge a_i ; \quad i = 1, 2, 3, \dots, M \quad (9.50)$$

where  $\wedge$  is logical min operator.

$$w_{ki}(t+1) = v_{ik}(t+1) / [0.5 + \sum_{i=1}^M v_{ik}(t+1)] \quad (9.51)$$

The constant 0.5 is included in the denominator to avoid division by zero.

For each input pattern the above 4 steps are repeated. A schematic diagram of an ART network is presented in Fig. 9.5.

The ART network operates automatically by its gain parameter  $G$  and reset parameter  $R$ . The gain parameter  $G$  can assume two values: 1 and 0, as outlined below.

$$\begin{aligned} G &= 0, \text{ if one of the units in the } F_2 \text{ layer is ON.} \\ &= 1, \text{ if all the units in the } F_2 \text{ layer are OFF.} \end{aligned}$$

Formally,  $G$  can be computed by a step function  $u$ .

$$G = u \left[ \sum_{i=1}^M a_i - \sum_{j=1}^N y_j - 0.5 \right] \quad (9.52)$$

where  $u(x) = 1$  if  $x > 0$   
 $= 0$ , otherwise.

$y_j$  = the output of the j-th unit F<sub>2</sub> layer  
 $= 1$ , if unit j is a winner  
 $= 0$ , otherwise.

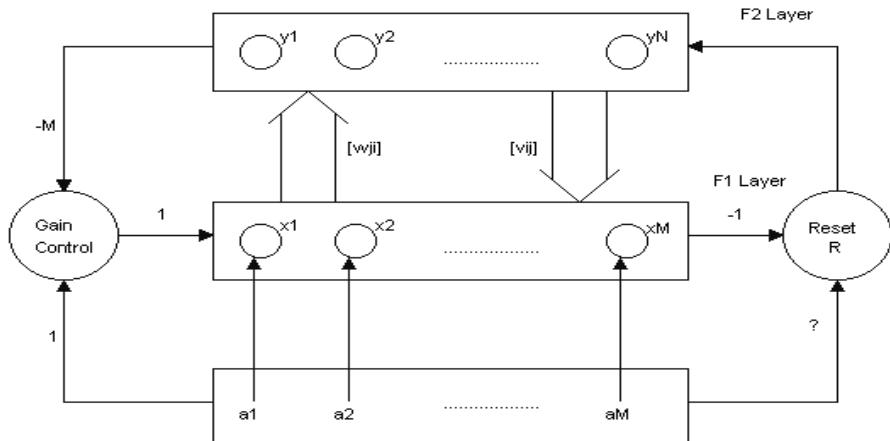


Fig. 9.5: An ART Network.

The reset parameter is defined as follows:

$$R = u \left[ \rho \sum_{i=1}^M a_i - \sum_{i=1}^M x_i \right] \quad (9.53)$$

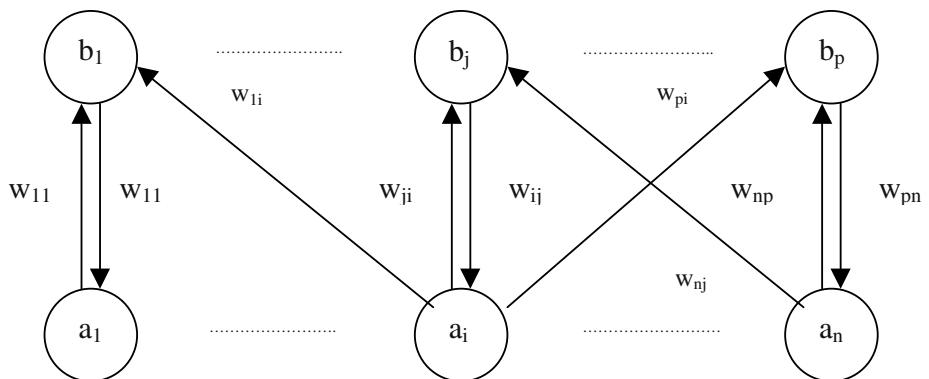
$$\text{where } x_i = u \left[ a_i + \sum_{j=1}^N u_{ij} y_j + G - 1.5 \right] \quad (9.54)$$

denotes the output of the  $i$ -th unit in the  $F_1$  layer. If the  $k$ -th unit in the  $F_2$  layer is the winner, then  $y_k = 1$  and  $y_j = 0$  for all  $j \neq k$ . Further, since one unit in the  $F_2$  layer is on,  $G=0$ . Then 2 out of 3 terms in (9.53) being 1,  $x_i=1$ . Again, if none of the units in the  $F_2$  layer is on, then  $y_j=0$  for all  $j$  and  $G=1$ . Under this case also,  $x_i=1$  if 2 out of 3 terms in (9.54) is 1. Expression (9.54) thus is referred to as the 2/3 rule.

When the vigilance test (vide (9.49)) succeeds, the argument of  $u(\cdot)$  in (9.54) is negative, and hence  $R=0$ . On the other hand, when the vigilance test fails, then argument of  $U$  in (9.54) is positive, and hence by (9.53)  $R=1$ . A unity value of reset indicates that the current winning unit is disabled and all the units in  $F_2$  layer are reset to OFF. Consequently  $G=1$ , and another winning unit will be selected.

## 9.5 Fuzzy Associative Memory

Introduced by Kosko [11] as a by-product of his work in Fuzzy theory. Fuzzy associative memory (FAM) is a two-layer hetero-associative Fuzzy classifier. FAM stores a fuzzy pattern pair  $(A_k, B_k)$ , where the  $k$ -th pattern pair is represented by fuzzy sets  $A_k = \{a_1^k, \dots, a_n^k\}$  and  $B_k = \{b_1^k, \dots, b_p^k\}$ . The FAM learns offline in discrete time and is represented by two-layer topology that includes  $n$  processing elements in layer A and  $m$  processing elements in layer B. Fig.9.6 presents a two-layer FAM.



**Fig.9.6:** Topology of a FAM.

Let

$\mu_{A_k}(a_i^k)$  = Membership of  $a_i$  to belong to fuzzy set  $A_k$ , where  $A_k \subseteq A$  and  $A$  is a universe,

$\mu_{B_k}(b_j^k)$  = Membership of  $b_j$  to belong to fuzzy set  $B_k$ , where  $B_k \subseteq B$  and  $B$  is a universe.

The encoding and recall of a fuzzy associative memory is now presented below.

- **Encoding**

The encoding process in a FAM is realized by Hebbian learning. In Hebbian learning when a neuron  $a_i^k$  excites  $b_j^k$ , the connectivity strength  $w_{ji}$  from neuron  $a_i^k$  to  $b_j^k$  is increased. In fuzzy associative memory this is realized by

$$w_{ji} = \text{Min} (\mu_{A_k}(a_i^k), \mu_{B_k}(b_j^k)) \quad (9.55)$$

where  $w_{ji}$  is the connection strength from the  $i$ -th neuron of layer A to the  $j$ -th neuron in layer B. The encoding procedure can store only one pattern.

- **Recall**

For recall of  $\mu_{B_k}(b_j^k)$  we use the following expansion:

$$\mu_{B_k}(b_j^k) = \text{Max}_{i=1}^n [\text{Min} (\mu(a_i^k), w_{ji})] \quad (9.56)$$

Further,  $\mu_{A_k}(a_i^k)$  can be evaluated by invoking the following expression:

$$\mu_{A_k}(a_i^k) = \text{Max}_{j=1}^P [\text{Min} (\mu(b_j^k), w_{ij})] \quad (9.57)$$

For simplicity in notations, we represent  $\mu_{A_k}(a_i^k)$  by  $A_k$  and  $\mu_{B_k}(b_j^k)$  by  $B_k$  only. Then the connectivity weight matrix  $\mathbf{W} = [w_{ji}]$  by (9.54) is given as follows.

$$\mathbf{W} = [w_{ji}] = (A_k)^T \circ (B_k) \quad (9.58)$$

Now, for a given  $A_k'$  we can complete  $B_k'$  by

$$B_k' = A_k' \circ \mathbf{W} \quad (9.59)$$

And for a given  $B_k'$  we can complete  $A_k'$  by

$$A_k' = B_k' \circ \mathbf{W}^T \quad (9.60)$$

**Example 9.3:** Given  $A_k$  and  $B_k$  as follows, we need to evaluate  $B_k'$  for a given  $A_k'$ .

Let

$$A_k = [ \begin{array}{ccc} 0.6 & 0.9 & 0.2 \end{array} ]$$

$$B_k = [ \begin{array}{cc} 0.8 & 0.7 \end{array} ]$$

Then

$$\begin{aligned} W &= A_k^T \circ B_k \\ &= \left[ \begin{array}{cc} 0.6 & 0.3 \\ 0.8 & 0.3 \\ 0.2 & 0.2 \end{array} \right] \end{aligned}$$

Now, with  $A_k' = [ \begin{array}{ccc} 0.5 & 0.8 & 0.1 \end{array} ]$

We find

$$B_k' = [ \begin{array}{cc} 0.8 & 0.3 \end{array} ], \text{ we compute } A_k' \text{ by (9.59)}$$

$$\begin{aligned} A_k' &= B_k' \circ W^T \\ &= [ \begin{array}{cc} 0.8 & 0.3 \end{array} ] \circ \left[ \begin{array}{ccc} 0.6 & 0.8 & 0.2 \\ 0.3 & 0.3 & 0.2 \end{array} \right] \\ &= [ \begin{array}{ccc} 0.6 & 0.8 & 0.2 \end{array} ]. \end{aligned}$$

## 9.6 Discussions

The chapter introduced different models of unsupervised learning using neural nets. Among the unsupervised learning algorithms, Hopfield net is most popular. It employs a recurrent neural topology and is capable of storing n-stable states with approximately n neurons. During the recognition phase, Hopfield nets can identify the nearest stable state from a noisy system state. The capability of re-constructing a stable state from a noisy state enhanced the scope of application of Hopfield nets in many engineering problems. The chapter addressed one such application of Hopfield nets in detecting aircraft from its partially occluded image. Hopfield net can also successfully be used in optimization problems. The application of Hopfield net in the well-known TSP -problem has also been narrated to illustrate its scope of application in optimization problems.

The chapter also presented a brief discussion on Boltzman machines. Such machines are needed to avoid possible trapping of the states at the local minima of the Lyapunov energy function. The latter part of the chapter introduced the notion of bidirectional associative memory, adaptive resonance theory neural nets and fuzzy associative memory. The bidirectional associative memory is a hetro-associative pattern classifier that stores pattern pairs and is capable to generate one pattern of each pair, when its counterpart pattern is supplied. Fuzzy associative memory is similar with BAM with the additional characteristics of storing fuzzy pattern pairs, other than bipolar binary patterns. The adaptive resonance theory introduced in the chapter leads to a general solution of the well known “*stability-plasticity dilemma*” of the classical pattern recognition theory.

## Exercise

1. The following is the firing rule of an arbitrary neuron  $i$  in a discrete Hopfield net:

$$\begin{aligned} V_i &= 1 \text{ if } \sum_{\substack{j \neq i}} w_{ij} V_j > th_i \\ &= 0, \text{ if } \sum_{\substack{j \neq i}} w_{ij} V_j > th_i \end{aligned}$$

where  $V_i$  denotes the output of neuron  $i$ , and  $th_i$  is its threshold. Let  $E$  be an energy function to study the stability of the Hopfield dynamics.

$$E = -\left(\frac{1}{2}\right) \sum_j \sum_i w_{ij} V_j V_i + \sum_i V_i th_i.$$

Compute  $\Delta E$  and show that it is unconditionally negative; hence determine the stability of the Hopfield dynamics.

**Hints:**  $\Delta E = -\Delta V_i (\sum_{j \neq i} w_{ij} V_j - th_i)$ .

When  $V_i$  is positive,  $\Delta V_i$  is also positive, and the parenthesized term is also positive (by firing rule of the  $i$ -th neuron). Hence,  $\Delta E$  is negative. Further a negative change in energy is a necessary condition for stability of a dynamics. Hence, the given dynamics is stable.]

2. Represent the following binary matrices by one-dimensional vectors and hence store them as stable points of a discrete Hopfield net.

$$\begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix}$$

[**Hints:** Represent the first matrix by a binary string:

$$[1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1]$$

and call it  $\mathbf{X}_1$ . Similarly express the other two matrices by vectors namely  $\mathbf{X}_2$  and  $\mathbf{X}_3$  respectively. Then evaluate the weight matrix  $\mathbf{W}$  by

$$\mathbf{W} = \sum_{i=1}^3 (\mathbf{X}_i^T \mathbf{X}_i - \mathbf{I}).$$

$\mathbf{W}$  denotes the stored information about the three vectors. In order to verify the stability of the Hopfield dynamics at the stable points, submit the  $\mathbf{X}_i$  as input and compute  $F_h(\mathbf{X}_i \mathbf{W})$  for  $i = 1$  to 3 and ensure that the result tallies with  $\mathbf{X}_i$ .

3. Use a BAM to store three pairs of vectors as follows:

$$\mathbf{A}_1 = [1 \ -1 \ 1] \text{ and } \mathbf{B}_1 = [-1 \ 1 \ 1 \ -1],$$

$$\mathbf{A}_2 = [-1 \ 1 \ 1] \text{ and } \mathbf{B}_2 = [1 \ -1 \ 1 \ 1],$$

$$\mathbf{A}_3 = [1 \ 1 \ 1] \text{ and } \mathbf{B}_3 = [-1 \ 1 \ 1 \ -1].$$

- a) Determine the weight matrix.
- b) Examine whether the BAM can accurately retrieve the associated vector  $\mathbf{B}_i$  given each of the original vectors  $\mathbf{A}_i$  for  $i=1$  to 3.

$$[\mathbf{Hints:} \text{ Compute } \mathbf{W} = \sum_{i=1}^3 \mathbf{A}_i^T \mathbf{B}_i.]$$

Next check whether  $F_h(\mathbf{A}_i \mathbf{W}) = \mathbf{B}_i$  for  $i=1$  to 3.]

## References

- [1] Carpenter, G. A. and Grossberg, S., “Adaptive Resonance Theory (ART),” In *The Handbook of Brain Theory and Neural Networks*, Arbib, M. A. (Ed.), MIT Press, Cambridge, MA, pp. 79-82, 1995.
- [2] Carpenter, G. A. and Grossberg, S., “A massively parallel architecture for a self-organizing neural pattern recognition machine,” *Computer Vision, Graphics and Image Processing*, Academic Press, vol. 37, pp. 54-115, 1987.
- [3] Freeman, J. A. and Sakpura, D. M., *Neural Networks: Algorithms, Applications and Programming Techniques*, MA, 1991.
- [4] Fu, L. M., *Neural Networks in Computer Intelligence*, McGraw-Hill, NY, 1994.
- [5] Gonzalez, R. C. and Woods, R. E., *Digital Image Processing*, Addison-Wesley, Reading, MA, 2000.
- [6] Hinton, G. E., “Deterministic Boltzman machine learning performs steepest descent in weight-space,” *Neural Computation*, vol. 1, pp. 143-150, 1989..
- [7] Hopfield, J. J., “Neural networks and physical systems with emergent collective computational ability,” *Proc. of Natl. Academy of Science*, USA, vol. 79, pp. 2554-2558, April 1982.
- [8] Hopfield, J. J., “Neurons with graded response have collective computational properties like those of two-state neurons,” *Proc. of Natl. Academy of Science*, USA, vol. 81, pp. 3088-3092, May 1984.
- [9] Jain, A. K., Mao, J. and Mohiuddin, K. M., “Artificial Neural Networks: a Tutorial,” *IEEE Computer*, pp. 31-44, March 1996.
- [10] Kosko, B., “Bidirectional Associative Memories,” *IEEE Trans. on Systems, Man and Cybernetics*, vol. 18, pp. 49-60, 1992.
- [11] Kosko, B., *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*, Prentice-Hall, NJ, 1991.

# 10

## Competitive Learning Using Neural Nets

*The chapter presented different models of competitive learning using neural networks. The first model is concerned with a two layer competitive learning network having a noise-free input realized with an on-center off-surround configuration. An analysis of the model has been presented in detail. The scope of realization of competition by Hebbian learning and the way-out to handle the limitation of Hebbian learning by Oja's principle have been discussed in detail. The chapter also introduced principal component analysis and self-organizing feature models and examined their applications in face recognition problem.*

### 10.1 Introduction

In competitive learning, learning laws are constructed to modulate the difference between the output signal and the synaptic weights of the neurons. According to

Grossberg [1], the general form of competitive learning is given by the following expression of synaptic dynamics:

$$\frac{dw_{ij}}{dt} = s_i[s_j - w_{ij}(t)] \quad (10.1)$$

where  $w_{ij}(t)$  denotes connectivity strength from neuron j to neuron i at time t;  $s_i$  and  $s_j$  denote the output signal of the neurons (units) i and j respectively, and

$$s_k = f(x_k(t)) \text{ for } k \in (i, j). \quad (10.2)$$

The function  $f$  is a non-linear inhibiting function like *Sigmoid* and  $x_k(t)$  is the weighted sum of signals received from other neurons at neuron k.

The expression (10.1) can be re-written as

$$\frac{dw_{ij}}{dt} = -s_i w_{ij}(t) + s_i s_j. \quad (10.3)$$

It may be noted that expression (10.3) has similarity with classical Hebbian learning where the term  $-s_i w_{ij}(t)$  in (10.3) is replaced by  $-\alpha w_{ij}(t)$ ,  $\alpha$  being a rate constant. Thus in Hebbian learning when  $s_i=0$ ,

$$\frac{dw_{ij}}{dt} = -\alpha w_{ij}(t) \quad (10.4)$$

i.e the synaptic weight forgets the already acquired knowledge. However, in competitive learning, if  $s_i=0$ ,

$$\frac{dw_{ij}}{dt} = 0 \quad (10.5)$$

$$\text{or, } w_{ij}(t) = \text{constant.} \quad (10.6)$$

Thus synaptic weights do not change when output of a neuron is zero. This is one fundamental difference between the competitive neural learning and the Hebbian learning.

A competitive learning neural net consists of two layers. An external input is presented to neurons of one layer, called the input layer. The neurons of the

input layer transfer signal to the neurons of the other layer (output layer) through feed-forward connection of weights. The neurons in the output layer competes with each other, leaving eventually one of the units (say  $i$ ) as the winner. The weights connected between neurons of the output layer are adjusted following a competitive learning rule like expression (10.1).

The first distinctive feature of competitive learning in contrast to Hebbian learning has already been mentioned. The second characteristic difference of the above two learning rules is noteworthy. While the Hebbian learning is distributed, i.e., all the weights are adjusted for every input pattern, the competitive learning is not distributed. In fact, input vectors leading to the same winning neuron in the output layer of a competitive learning network yields a weight vector for that neuron, which is an average of all the corresponding input vectors [10]. The basic competitive learning model (10.1) is extended further as given below.

- ◆ If the neurons in the input layer has a linear function  $s_j = x_j$ , then

$$\frac{dw_{ij}}{dt} = s_i [ x_j - w_{ij}(t) ] \quad (10.7)$$

- ◆ Bart Kosko [5] proposed a stochastic version of the basic competitive learning model (10.1). He considered a zero mean Gaussian white noise  $n_{ij}(t)$  acting over the competitive term in the weight dynamics. Thus

$$\frac{dw_{ij}}{dt} = s_i [ s_j - w_{ij}(t) ] + n_{ij}(t). \quad (10.8)$$

- ◆ An alternative form of random linear competitive learning proposed by Kosko [6] is given by

$$\frac{dw_{ij}}{dt} = s_i [ x_j - w_{ij}(t) ] + n_{ij}(t). \quad (10.9)$$

- ◆ In the differential competition, learning takes place only when there is a change in the post-synaptic neuronal activation. The elementary deterministic model of a differential competitive learning is given by

$$\frac{dw_{ij}}{dt} = ( ds_i / dt ) [ s_j - w_{ij}(t) ]. \quad (10.10)$$

With linear activation function  $s_j = x_j$ , linear differential competitive learning law is given by

$$\frac{dw_{ij}}{dt} = (ds_i / dt) [x_j - w_{ij}(t)]. \quad (10.11)$$

Random differential competitive learning (RDCL) and random linear differential competitive learning models (RLDCL) now can easily be stated by adding noise term to the right hand side of (10.10) and (10.11) respectively. Thus

RDCL:

$$\frac{dw_{ij}}{dt} = (ds_i / dt) [s_j - w_{ij}(t)] + n_{ij}(t), \quad (10.12)$$

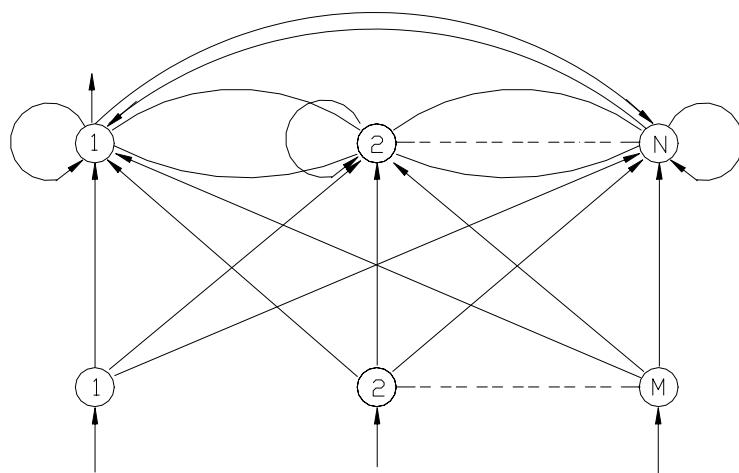
and RLDCL:

$$\frac{dw_{ij}}{dt} = (ds_i / dt) [x_j - w_{ij}(t)] + n_{ij}(t). \quad (10.13)$$

This chapter provides two distinctive models of competitive learning. The first one is concerned with a two layered network with feed-forward connections from the first to the second layer and recurrent connections among neurons of the second layer. The other one, popularly known as self-organizing feature map, will also be introduced in this chapter.

## 10.2 Two Layered Recurrent Networks for Competitive Learning

Consider a two layered network with feed-forward connections among neurons of the first (input) layer and the second layer and feedback connections among neurons of the second (output) layer (Fig. 10.1). In the second layer of such network, the self-feedback connections are excitatory, and cross-feedback connections to other neurons are either excitatory or inhibitory depending on the type of applications. Generally, the weights on the forward path are adaptive, and on the feedback path are fixed. Such a combination of both feed-forward and feedback connections results in some kind of competition among the neurons in the output layer, and hence such networks are called *competitive learning networks*.



**Fig 10.1:** A two-layered competitive learning network with feed-forward connections from the input to the output layer and feedback connections among neurons of the output layer.

Three different configurations of the network shown in Fig. 10.1 are of interest. They are briefly outlined below:

- ◆ **Configuration 1:** In this configuration, the output functions are linear, and the feedback connections in the output layer are made in an on center off- surround fashion. An on-center off-surround connection means the self-loops around neurons are self-excitatory, whereas cross-weights among neurons are inhibitory. Such network having no adaptive feed-forward or feedback weights are similar with short-term memories (STM) of human beings. The configuration 1 thus is of academic interest only and has no practical utility.
- ◆ **Configuration 2:** In this configuration, the output function of the neurons in the feedback (competitive) layer are nonlinear with adaptive feed-forward weights and fixed on-center off-surround connections in the output layer. Such networks can be used for *pattern clustering*. The neurons in the output layer here compete with each other, and the neurons that yields the nonzero output at equilibrium is called the *winner*.
- ◆ **Configuration 3:** The third configuration needs nonlinear output functions of the neurons in the output layers, and excitatory connections

to far away neurons in the output layers. The network thus can perform the task of *feature mapping*, and so it is called a self-organizing network. Such networks are capable to map the features of input patterns onto one or two dimensional feature space. It is indeed important to note that in configuration 3, the feed-forward weights are constants, but the feedback weights are adaptive. In the recall phase, the winning neurons corresponding to a given input pattern in the output layer is identified.

## 10.3 Components of a Competitive Learning Network

Competitive learning networks, introduced in this chapter, comprises of three major modules: i)a pre-processing layers, ii)a input layer, and iii)an output layer.

### 10.3.1 The Pre-processing Layer

The preprocessing layer is needed to take care of the well-known noise saturation problem [10] of the input layer. This can be realized by employing a *shunting activation model* with on-center off-surround configuration given by

$$\frac{dx_i}{dt} = -Ax_i + (B - x_i)I_i - (C+x_i)\sum I_j, \quad j \neq i \quad (10.14)$$

where

$I_i$  is the signal supplied at the  $i$ -th external input,

$x_i$  is the noise – filtered signal at  $i$ -th neuron of the input layer, and

A, B and C are constants of appropriate dimensions.

The steady – state signals received at the  $i$ - th neuron is given by

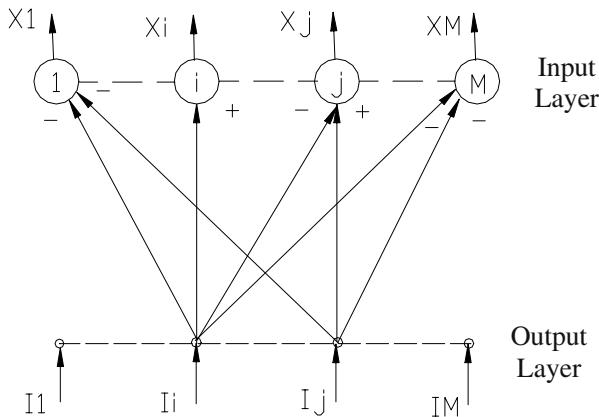
$$x_i(\infty) = \frac{((B+C)I_i - CI)}{(A+I)} \quad (10.15)$$

$$\text{where } I = \sum_{i=1}^M I_i \quad (10.16)$$

The output functions of  $x_i$ s employ a linear function

$$\left. \begin{aligned} f(x_i) &= x_i, \text{ for } x_i \geq 0 \\ &= 0, \text{ otherwise.} \end{aligned} \right\} \quad (10.17)$$

It is clear from (10.15) and (10.17) that when  $I_i < (CI / (B+C))$ ,  $x_i(\infty) \rightarrow 0$ . Consequently, the input should be greater than some minimum threshold value so as to create a positive activation level of the neurons. Thus the neurons in the input layers do not respond to noise input, if the noise amplitude is below some threshold value.



**Fig: 10.2:** The input layer showing a few connections with external inputs.

### 10.3.2 The Instar Connectivity from Neurons of the Input to the Output Layer

The neurons in the output (feedback) layer receive signals from all neurons in the previous layer. Such configuration, where a neuron receives signals from several neurons of previous layer, is called an *instar*. One typical instar configuration is shown in Fig. 10.2.

Let

$$\mathbf{X} = [x_1 \ x_2 \ \dots \ x_M]^T \text{ be an input vector,}$$

$$\mathbf{W} = [w_1 \ w_2 \ \dots \ w_M]^T \text{ be a weight vector of the respective inputs}$$

$$[x_1 \ x_2 \ \dots \ x_M]^T,$$

$$y(t) = \text{scalar output of a neuron in the feedback layer}.$$

Then the activation dynamics of the instar processing neuron is given by

$$\frac{dy}{dt} = -y(t) + \mathbf{W}^T \mathbf{X} \quad (10.18)$$

The first term in the (10.18) denotes a passive decay, and the second term corresponds to the contribution from  $[x_1 \ x_2 \ \dots \ x_M]$ ,

The steady – state solution of  $y(t)$  denoted by  $y(\infty)$  is given by

$$y(\infty) = \mathbf{W}^T \mathbf{X}. \quad (10.19)$$

### 10.3.3 Competitive Learning in the Output Layer

Given a input vector  $\mathbf{X} = [x_1 \ x_2 \ \dots \ x_M]^T$  in the input layer, the instar responds maximally if the weight vector  $\mathbf{W} = [w_1 \ w_2 \ \dots \ w_M]^T$  is moved towards the input vector. The learning law for weight adaptation is given by

$$\frac{d\mathbf{W}}{dt} = [\mathbf{X} - \mathbf{W}] f(y) \quad (10.20)$$

where  $f(y)$  is the output of the instar processing unit.

The discrete version of expression (10.20) is given by

$$\Delta\mathbf{W}(t) = \eta[\mathbf{X} - \mathbf{W}] f(y) \quad (10.21)$$

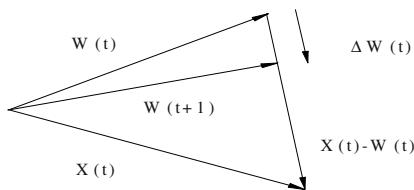
where  $\eta$  is the learning rate. When the output functions are binary,  $f(y)$  can take either 0 or 1 values. Therefore, the weights in the instar are adjusted only when the output of the instar  $f(y) = 1$ . Consequently, expression (10.21) is modified to

$$\Delta\mathbf{W}(t) = \eta[\mathbf{X} - \mathbf{W}]. \quad (10.22)$$

The updated weights are then computed by

$$\mathbf{W}(t+1) = \mathbf{W}(t) + \Delta\mathbf{W}(t) \quad (10.23)$$

Expressions (10.22) and (10.23) jointly can be represented by a vector diagram as shown in Fig. 10.3 below.



**Fig. 10.3:** The vector diagram representing the weight adaptation policy in the competitive learning layer.

The adaptation of weight vector should be continued by (10.22) and (10.23) in sequence until the weight vector reaches a steady – state value, following which there will be no further change in weight. Using expectation operation we have

$$E[\Delta W] = 0 \quad (10.24)$$

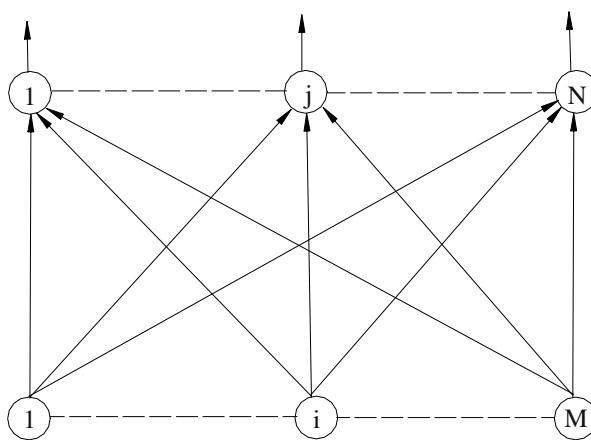
$$\Rightarrow E[\eta(X-W)] = 0 \quad (10.25)$$

$$\Rightarrow \eta(E(X) - E(W)) = 0 \quad (10.26)$$

or ,  $E(W)$  = average value of  $X$  .

Thus the weight vector at steady – state will be equal to the average of the set of input vectors  $X$ .

A single instar responds to a set of input vectors to capture the average behavior of the set. When there are  $M$  neurons in the input and  $N$  neurons in the competitive layer, there exist as many as  $N$  instars (Fig 10.4). Ideally the number ( $N$ ) of instars in the competitive layer corresponds to the number of different categories of the input pattern. An unknown input vector  $X$  supplied at the input layer can be categorized to one of the  $N$  classes by identifying the neuron with the maximum response in the competitive layer.



**Fig 10.4:** A group of  $N$  instars.

## 10.4 Hebbian Learning in the Competitive Layer

With *linear* output functions for the instar processing neurons, i.e.  $f(y)=y$ , we can represent output  $y$  as a scalar measure of similarity. In other words, given the input vector  $\mathbf{X}$ , the weights should be adjusted to yield a large output  $y$  on the average. Using Hebbian type learning, we can realize this by employing following learning rule:

$$\Delta w_i = \eta y x_i \quad (10.27)$$

where  $\eta$  is the learning rate. Now with the increase in  $y$ , the weights keep on growing. So the weights cannot converge, and consequently the weight-change will not be zero. We can formally prove this statement using expectation operator as follows:

$$E[\Delta w_i] = E[\eta \sum_j w_j x_j x_i] = \eta \sum_j w_j E[x_j x_i] \quad (10.28)$$

where it is presumed that the weight vector is statistically independent of the input vector.

With

$$\Delta \mathbf{W} = (\Delta w_1, \Delta w_2, \dots, \Delta w_M)^T, \quad (10.29)$$

$$\text{and } \mathbf{R} = E[\mathbf{X}\mathbf{X}^T], \quad (10.30)$$

we can easily evaluate

$$E(\Delta W) = \eta RW. \quad (10.31)$$

Here  $R$  is a positive semidefinite auto-correlation matrix. Since for any vector  $\mathbf{a}$  we have

$$\begin{aligned} & \mathbf{a}^T R \mathbf{a} \\ &= \mathbf{a}^T E(\mathbf{X}\mathbf{X}^T) \mathbf{a} \\ &= E(\mathbf{a}^T \mathbf{X} \mathbf{X}^T \mathbf{a}) \\ &= E(\mathbf{X}^T \mathbf{a})^2 \geq 0, \end{aligned} \quad (10.32)$$

$R$  will have only positive eigenvalues. The weight vector converges if

$$E(\Delta W) = 0. \quad (10.33)$$

Now, from (10.31) and (10.33) we have:

$$E(\Delta W) = \eta RW = 0, \quad (10.34)$$

$$\text{or, } RW = 0. \quad (10.35)$$

The expression (10.35) means that the resulting weight vector is an eigenvector with zero eigenvalue. Since  $R$  has positive eigenvalues, the weight vector is not stable. Thus any fluctuation of the weight vector with a component along an eigenvector of  $R$  will result in a weight vector, which would grow eventually, and the component along the eigenvector with the largest eigenvalue will dominate [10].

The above statement can be shown as follows. Projection of the average change of the weight vector onto one of the eigenvector  $\mathbf{q}_i$  of  $R$  yields

$$\mathbf{q}_i^T E[\Delta W] = \eta \mathbf{q}_i^T R W. \quad (10.36)$$

If  $\mathbf{W}$  is the eigenvector of  $\mathbf{q}_i$ , the projection  $\mathbf{q}_i^T E[\Delta W]$  is nonzero, since  $R\mathbf{q}_i = \lambda_i \mathbf{q}_i$ , and the eigenvectors  $[\mathbf{q}_i]$  are orthonormal. Thus,

$$\mathbf{q}_i^T E[\Delta W] = \eta \mathbf{q}_i^T R \mathbf{q}_i = \eta \lambda_i. \quad (10.37)$$

The projection yields a maximum value when  $\lambda_i = \lambda_{\max}$ , the largest eigenvalue. Let the eigenvector corresponding to  $\lambda_i = \lambda_{\max}$  be  $\mathbf{W}_0$ . Then

$$E(\Delta \mathbf{W}) = \eta \mathbf{R} \mathbf{W}_0 = \eta \lambda_{\max} \mathbf{W}_0. \quad (10.38)$$

Hence, the average change in weight vector is dominated by the component along the eigen vector with maximum eigenvalue.

For adaptation of weights, we use the following expression:

$$\begin{aligned} \mathbf{W}(m+1) &= \mathbf{W}(m) + E(\Delta \mathbf{W}) \\ &= \mathbf{W}(m) + \eta \lambda_{\max} \mathbf{W}_0. \end{aligned} \quad (10.39)$$

Starting with  $\mathbf{W}(0)=0$ , we find:

$$\mathbf{W}(m) = \eta \lambda_{\max} \mathbf{W}_0. \quad (10.40)$$

Since  $E(\Delta \mathbf{W})$  is positive (vide 10.38),  $\mathbf{W}(m)$  will be unstable. Thus with the Hebbian learning only, the weight vector  $\mathbf{W}(m)$  is unstable.

### Oja's Correction

To overcome the unstable behavior of the weight vector  $\mathbf{W}(m)$ , Oja [9] incorporated an additional decay term in (10.27) by considering the decay proportional to  $y^2$ , where  $y$  is an output of a neuron in the competitive layer. Oja's rule can formally be stated as follows:

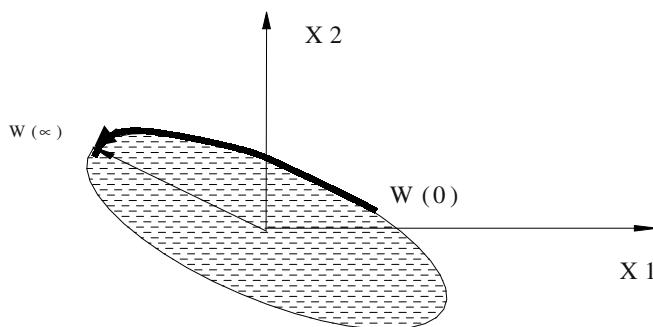
$$\begin{aligned} \Delta w_i &= -\eta y^2 w_i + \eta y x_i \\ &= \eta y (x_i - y w_i). \end{aligned} \quad (10.41)$$

It is evident from (10.41) that the change in weight depends on the difference of the input  $x_i$  and the back-propagated output  $y w_i$  of the  $i$ -th neuron in the competitive layer. The weight vector will eventually align with the eigenvector of  $\mathbf{R}$  corresponding to the largest eigenvalue [9].

Let  $\mu = E[\mathbf{X}]$ , then the co-variance matrix  $\mathbf{R}$  is defined as

$$\mathbf{R} = E[(\mathbf{X} - \mu)(\mathbf{X} - \mu)^T]. \quad (10.42)$$

With such co-variance matrix  $\mathbf{R}$ , the final weight vector will be aligned with the largest principal component passing through the origin (Fig. 10.5) [3]. It should be added that Oja's rule forces the weight vector  $\mathbf{W}$  to settle down to the normalized value  $\|\mathbf{W}\|=1$ .



**Fig 10.5:** The final weight vector  $w(\alpha)$  is along the direction of maximum variance of input data, denoted by points inscribed in an ellipse.

## 10.5 Analysis of Pattern Clustering Network

For pattern clustering we need a competitive learning network with nonlinear output functions for units in the feedback layer. Such networks are capable of producing larger activation on a single unit and small activation on other units at equilibrium. The behavior of the network thus leads to a winner-take all situations, since only one neuron in the feedback layer will have nonzero activation. The neuron having the nonzero activation is the winner in the present context. For similar patterns, a competitive learning network is expected to have a unique winner in the competitive layer. Consequently, with suitably adjusted feed-forward weights, each of the neurons in the feedback layer can be made to win for similar sets of patterns. The output function in the feedback layer is generally nonlinear, like  $f(x)=x^n$ ,  $n > 1$ , any hard limiting function or semi linear sigmoid function. The neurons in the feedback layer are connected in an on-center off-surround manner. Because of applications in clustering, these networks are referred to as pattern clustering networks.

In pattern clustering, pattern classes are formed on unlabeled input data, and hence the corresponding learning is unsupervised. The weights in competitive learning, on the contrary, are adjusted only after winner neuron in the feedback layer is identified for a given input pattern.

Consider a competitive layer problem for binary input patterns. Let  $[x_1 \ x_2 \dots \ x_M]^T$  be a input vector applied in the input layer, where  $x_j=0$  for  $j \in [1, M]$ . The activation of the  $i$ -th unit in the competitive learning layer is given by

$$y_i = \sum_{j=1}^M w_{ij} x_j \quad (10.43)$$

where  $w_{ij}$  is the  $(i, j)$ -th element in the weight matrix corresponding to the connectivity from the  $j$ -th neuron of the input layer to the  $i$ -th neuron in the competitive layer.

Let

$$y_k = \max_i(y_i) \quad (10.44)$$

$$\text{i.e., } \mathbf{W}_k^T \mathbf{X} \geq \mathbf{W}_i^T \mathbf{X} \text{ for all } i, \quad (10.45)$$

where  $\mathbf{W}_k$  and  $\mathbf{W}_i$  are the weight vector of the instars  $k$  and  $i$  respectively. Let us further assume that the weight vectors  $\mathbf{W}_i$  to all the neurons are normalized, i.e.,  $\|\mathbf{W}_i\|=1$  for all  $i$ . Geometrically this means

$$\|\mathbf{X} - \mathbf{W}_k\| \leq \|\mathbf{X} - \mathbf{W}_i\|, \text{ for all } i \quad (10.46)$$

i.e., the input vector  $\mathbf{X}$  is closest to the weight vector  $\mathbf{W}_k$  among all  $\mathbf{W}_i$ .

Given a set of input vectors, applied one by one to the input layer. For each input, the winner neuron  $k$  in the feedback layer is identified and its weights are adjusted using (10.47) and (10.48).

$$\mathbf{W}_k(m+1) = \mathbf{W}_k(m) + \Delta \mathbf{W}_k(m) \quad (10.47)$$

$$\text{where, } \Delta \mathbf{W}_k(m) = \eta (\mathbf{X} - \mathbf{W}_k(m)) \quad (10.48)$$

Competitively learning laws like (10.47) and (10.48) and their variations are used for learning vector quantization (LVQ) [8]. LVQ is of two basic types. In LVQ<sub>1</sub>, the following learning laws are employed:

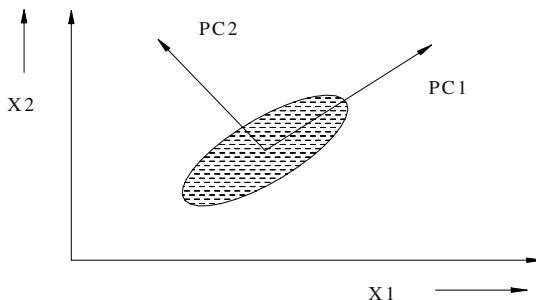
$$\left. \begin{array}{l} \Delta \mathbf{W}_k = \eta (\mathbf{X} - \mathbf{W}_k), \text{ if the winning class } k \text{ is correct,} \\ = -\eta (\mathbf{X} - \mathbf{W}_k), \text{ if the winning class } k \text{ is incorrect.} \end{array} \right\} \quad (10.49)$$

In LVQ<sub>2</sub> the following poling is adopted. If the input vector  $\mathbf{X}$  is in classified by the winning unit  $k$ , and if the nearest neighboring neuron  $i$  has the correct class, then

$$\left. \begin{array}{l} \Delta \mathbf{W}_k = -\eta (\mathbf{X} - \mathbf{W}_k) \text{ for incorrect winner} \\ \Delta \mathbf{W}_i = -\eta (\mathbf{X} - \mathbf{W}_i) \text{ for correct neighboring neuron.} \end{array} \right\} \quad (10.50)$$

## 10.6 Principal Component Analysis

Principal component analysis (PCA) is a method of representing data points in a compact form [5] by reducing their dimensions. Let us, for instance, consider a data set  $D = \{x | x \in R^N\}$ , where each data  $x$  is a point in  $N$ -dimensional space. Suppose, we are interested to find the first two principal components of  $D$ . What do they geometrically mean? This can be made clear with Fig. 10.6



**Fig.10.6:** Direction of the first two principal components of the 2–dimensional data points.

The first principal component  $PC_1$  is the direction along which the points in  $D$  have a maximum variance. The second principal component  $PC_2$  is the directional orthogonal to  $PC_1$  along which the variance is maximum. The principle stated above can be extended in this manner to determine the third, fourth and higher principal components.

It is possible to transform the data points  $x$  to  $y$  by an effective transformation  $x \rightarrow y$ , where  $x \in R^N$  and  $y \in R^P$ ,  $P < N$ . This can be accomplished by projecting the data points onto principal subspace formed by the first  $p$  principal components. This is the basis of *dimensionality reduction*, and the technique of representing data is referred to as *subspace decomposition*.

Let  $\hat{X}$  be the reconstructed data point obtained from the projections  $y$  onto the  $p$  largest principal component directions  $q_i$ s:

$$\hat{X} = \sum_{i=1}^p y_i q_i \quad (10.51)$$

The error vector  $\mathbf{e} = \mathbf{X} - \hat{\mathbf{X}}$  is orthogonal to the approximate data vector  $\hat{\mathbf{X}}$ . This is called the *principal of orthogonality*

Principal components can be easily extracted using the covariance matrix

$\mathbf{C} = E(\mathbf{X} - \bar{\mathbf{X}})(\mathbf{X} - \bar{\mathbf{X}})^T$ , where  $\bar{\mathbf{X}} = E(\mathbf{X})$  is the mean value of the data set  $\mathbf{X}$ . The principal components are the eigenvectors of the covariance matrix  $\mathbf{C}$  arranged in the descending order of the eigenvalues [2].

## 10.7 Self –Organizing Feature Map

Kohonen [7] proposed a new technique for mapping a given input pattern onto a 2-dimensional spatial organization of neurons. In fact, Kohonen considered a set of weights, connected between the components of the input pattern and a given neuron, located at position  $(i, j)$  in a two dimensional plane. Let us call the weight vector for neuron  $N_{ij}$  be  $\mathbf{W}_{ij}$ . Thus, for a set of  $(n \times n)$  neurons (points) on the 2-D plane, we would have  $n^2$  such weight vectors, denoted by,  $\mathbf{W}_{ij} \quad 1 \leq i, j \leq n$ . In Kohonen's model, the neuron having a minimum distance between its weight vector  $\mathbf{W}_{ij}$  and a given input pattern vector  $\mathbf{X}$  is first identified using the following criterion.

Find the  $(k, l)^{th}$  neuron, where

$$\|\mathbf{X} - \mathbf{W}_{kl}\| = \min_{1 \leq j \leq n} [\min_{1 \leq i \leq n} \|\mathbf{X} - \mathbf{W}_{ij}\|] \quad (10.52)$$

After the  $(k, l)^{th}$  neuron in the 2-D plane is located, the weights of its neighboring neurons are adjusted by using

$$\mathbf{W}_{ij}(t+1) = \mathbf{W}_{ij}(t) + \alpha \|\mathbf{X} - \mathbf{W}_{ij}\| \quad (10.53)$$

until the weight vector reaches equilibrium

$$\text{i.e., } \mathbf{W}_{ij}(t+1) = \mathbf{W}_{ij}(t). \quad (10.54)$$

A question, which now may be raised, is how to select the neighborhood neuron  $N_{ij}$ . In fact, this is done by selecting a square or circular zone around the neuron  $N_{ij}$ , where the dimension of the radius of the circle or the diagonal of the square with respect to  $N_{ij}$  is chosen arbitrarily.

After the weights in the neighborhood of the selected neuron are adapted, the neighborhood is gradually reduced, and the process of selection of neuron and weight adaptation of its neighboring neurons are continued until the

equilibrium condition is reached. By gradually reducing the neighborhood of the neuron, selected for weight adaptation, ultimately, a steady-state neighborhood of neurons in the 2-D plane is obtained. The region of neurons thus obtained represent a spatial mapping of the neurons, corresponding to the input pattern.

Further, if we have a number of input patterns representing various problem instances, each pattern can be mapped onto the 2-D space in a random order by the above method.

The algorithm for self-organizing neural adaptation for a input vector  $\mathbf{X}$  and the corresponding weight vector  $\mathbf{W}_{ij}$  is presented below for rectangular neighborhood of neurons.

**Procedure Self-organization ( $\mathbf{X}$ ,  $\mathbf{W}_{ij}$ )**

**Begin**

**Repeat**

**For**  $i := 1$  to  $n$  **do**

**Begin**

**For**  $j := 1$  to  $n$  **do**

**Begin**

**If**  $\|\mathbf{X} - \mathbf{W}_{k,l}\| = \text{Min} [\text{Min} \|\mathbf{X} - \mathbf{W}_{ij}\|]$   
 $1 \leq j \leq n \quad 1 \leq i \leq n$

**Then** adjust weights of neighboring neurons of  $N_{k,l}$   
by the following rule:

**For**  $i' := (k - \delta)$  to  $(k + \delta)$  **do**

**Begin**

**For**  $j' := (l - \delta)$  to  $(l + \delta)$  **do**  
**Begin**  
 $\mathbf{W}'_{i'j'}(t+1) = \mathbf{W}'_{ij}(t) + \alpha \|\mathbf{X} - \mathbf{W}'_{ij}\|$

**End For;**

**End For;**

$\delta := \delta - \epsilon; // \text{Space-organization} //$

**End For;**

**End For;**

**Until**  $\delta \leq$  pre-assigned quantity;

**End.**

The above algorithm should be repeated for all input patterns. Consequently, all the input patterns will be mapped on the 2D plane as  $n$ -dimensional points, each having a  $\mathbf{W}_{ij}$ .

During a recall phase, a input vector representing an unknown pattern is compared with the trained weights of all the neurons. An Euclidean norm is used to measure the similarity of the input pattern with the weight vectors of each

neuron. The neuron whose weight vector is found to be closest to the given input pattern is regarded as the recalled message.

## 10.8 Application in Face Recognition

This section describes two approaches for face recognition. The first is accomplished by principal component analysis, and the latter by self-organizing neural nets. There feature extraction part for both the methods are common, as evident from Fig. 10.7.

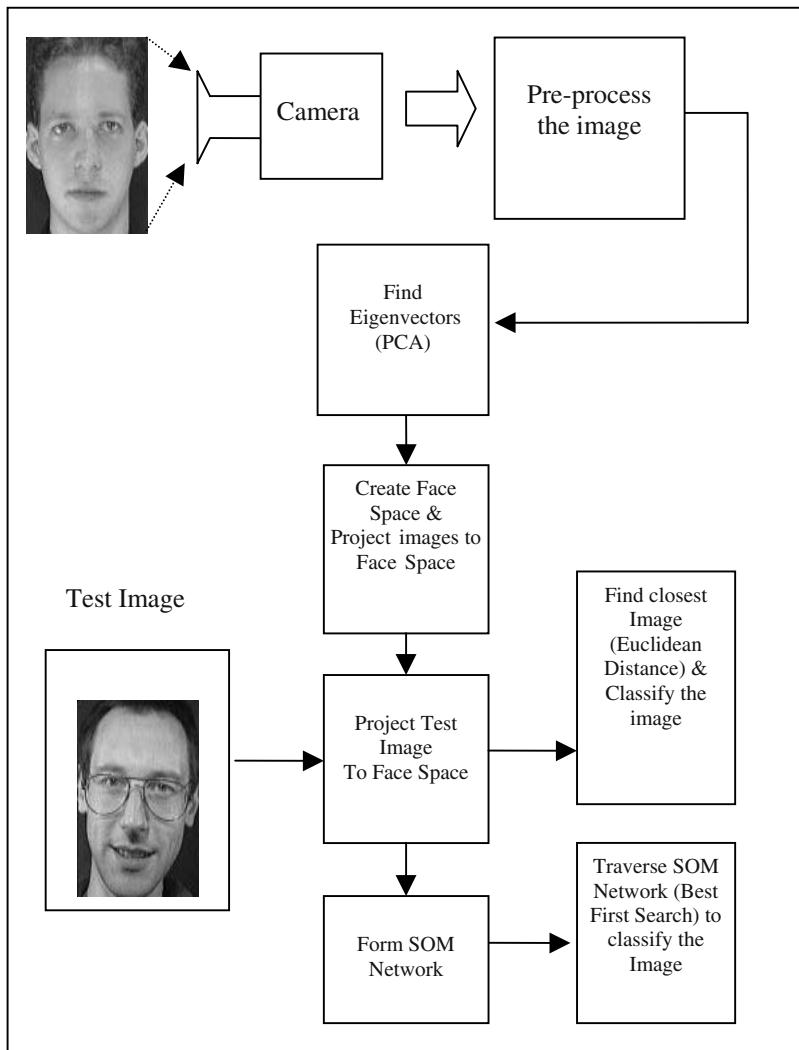
**Face Recognition Using Principal Component Analysis:** In this study, we considered 9 facial images of  $(32 \times 32)$  size for 40 individuals. Thus we have 360 images each of  $(32 \times 32)$  size. The average gray value of each image matrix is then computed and subtracted from the corresponding image matrix elements. One such image after subtraction of the average value is presented in Fig. 10.8. This is some form of a normalization that keeps the image free from illumination bias of the light source. Now, we construct a matrix  $\mathbf{X}$  of  $((32 \times 32) \times 360) = (1024 \times 360)$  dimension with the above data points. We also construct a covariance matrix by taking the product  $\mathbf{X} \mathbf{X}^T$  and evaluate the eigenvectors of  $\mathbf{X} \mathbf{X}^T$ . Since  $\mathbf{X}$  is of dimension  $(1024 \times 360)$ ,  $\mathbf{X} \mathbf{X}^T$  will have a dimension of  $(1024 \times 1024)$ .  $\mathbf{X} \mathbf{X}^T$  thus will have 1024 eigenvalues, out of which we select the first 12, on experimental basis. The eigenvectors corresponding to these 12 eigenvalues are called the first 12 **principal components**. Since the dimension of each principal component is  $(1024 \times 1)$ , grouping these 12 principal components in column-wise fashion, we construct a matrix of dimension  $(1024 \times 12)$ . We denote this matrix by  $\mathbf{EV}$  (for eigenvector).

To represent an image in the eigen space, we first represent that image in  $(1 \times 1024)$  format and then project it onto the face space by taking the dot product of the image matrix  $\mathbf{IM}$ , represented in  $(1 \times 1024)$  format and the  $\mathbf{EV}$  and call it a point  $(\mathbf{PT})$  in the face space. Thus

$$(\mathbf{PT})_{1 \times 12} = (\mathbf{IM})_{1 \times 1024} \cdot (\mathbf{EV})_{1024 \times 12}. \quad (10.55)$$

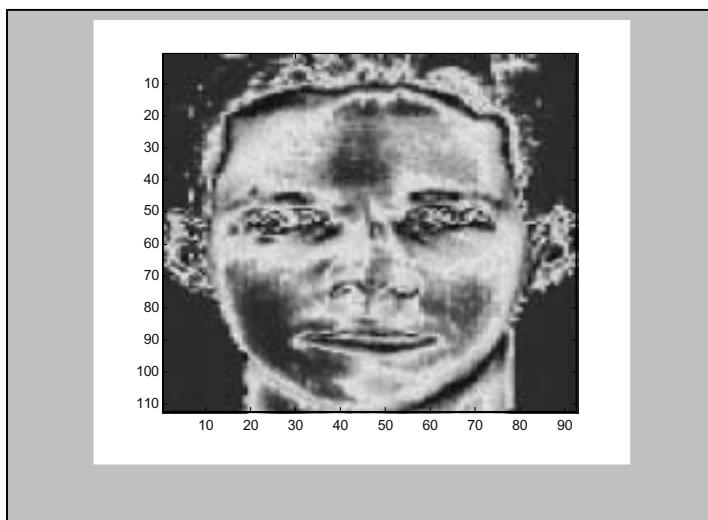
The projection of images onto face space as 12-dimensional points is presented in Fig. 10.9. Representing all 360 images by the above expression, we thus get 360 points in 12-dimensional image space.

Now, suppose given a test image and we want to classify it to one of the 40 persons. One simple way to solve this problem is to determine the corresponding image point  $(\mathbf{PT})$  in 12-dimensional space and then determine the image point (out of 360 points) that has the least Euclidean distance w.r.t the test image point. The test image thus can be classified to one of 360 images.



**Fig. 10.7:** A schematic architecture of the overall system.

The principal component analysis (PCA) thus reduces the dimension of matching from  $(32 \times 32)$  to  $(1 \times 12)$  but requires computing the distance of a test point with all image points. An alternative scheme that reduces less computing time is by a self-organizing neural net. The self-organizing scheme inputs the  $(1 \times 12)$  points for all of the 360 images, constructs a network and searches a test point by the best first search paradigm in the search space. A schematic diagram, briefly outlining the two approaches, is presented in Fig. 10.7 above.

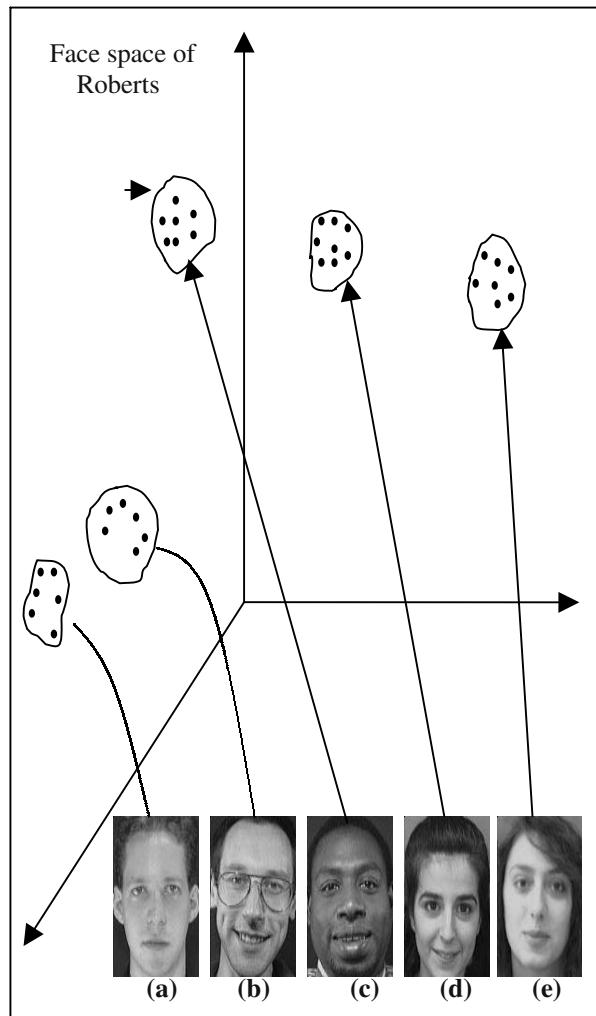


**Fig. 10.8:** A facial image after subtracting the average value from the pixels of the image.

**Face Recognition Using Self-organizing neural nets:** Let us assume that 12-dimensional vectors, one each for the 360 images, are already available by principal component analysis. We now consider each 12-dimensional point as input and adapt the weights of the neurons on a two-dimensional plane. Remember that for mapping a feature vector (here the 12-dimensional vector), we need to find the neuron on the 2-D plane, where the error  $\| \mathbf{IP} - \mathbf{W} \|$  is minimum for some weight vector  $\mathbf{W}$  of a neuron. We then have a selected small square region centering that neuron and adjust the weights of all neurons within that encirclement. Now, we find a next point, where again the error  $\| \mathbf{IP} - \mathbf{W} \|$  is minimum. We now consider a smaller square around the point on the 2-D space, where this error too is minimum. The process of adaptation of weights of neurons is thus continued, until we reach a situation, when the weights of a node do not change further. Thus the 12-dimensional point is mapped to one point on the 2-D neuronal plane. Sometimes instead of a single point, a small cluster of points is found to have almost no change in weights. In that case the 12-dimensional point is mapped onto the cluster. In the same manner, we map all the 360 points onto a 2-dimensional surface.

It may so happen that more than one projected image point corresponds to the same cluster on the 2-D surface. In that case the weight vectors of those neurons are again mapped hierarchically to another 2-D surface. Repeating the same process for all clusters, we got a hierarchical organization of self-organizing

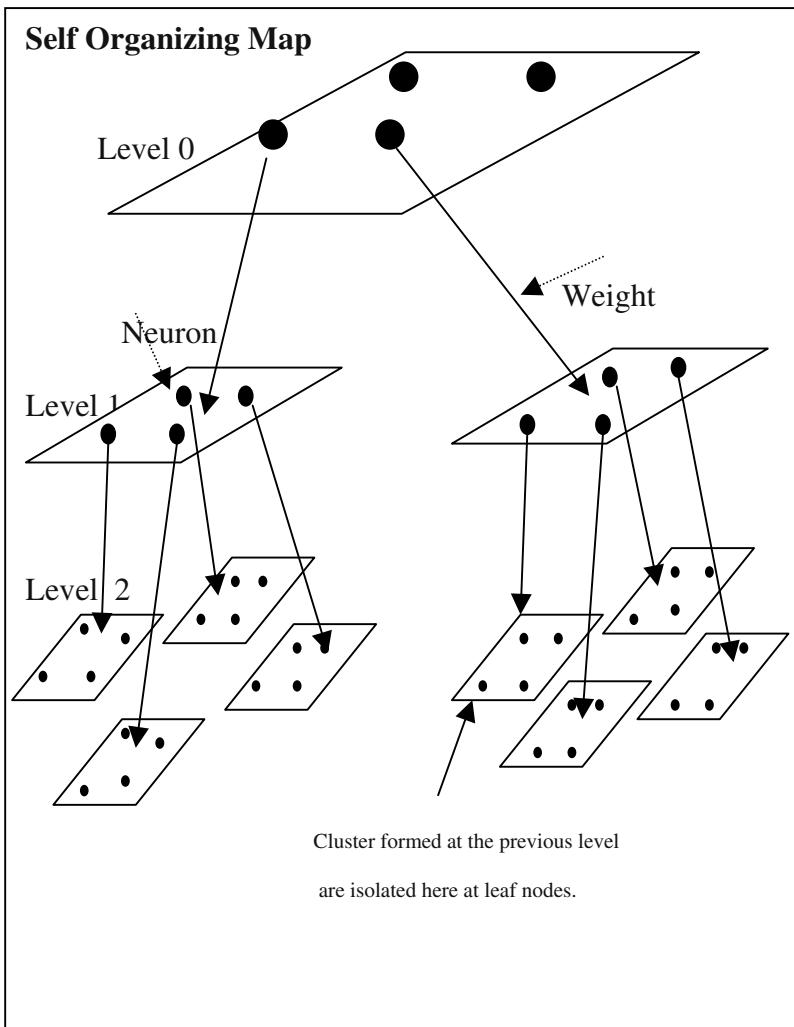
map (SOM) in Fig. 10.10. The main advantage of this structure is that it can always map a new projected image point onto the leave surfaces of the structure. Here is the significance of SOM over the back-propagation learning. For a new input-output pattern the back-propagation net is to be re-trained, whereas for a new input pattern the incremental classification of SOM just maps the new point onto one of the existing leaves of the 2-D surfaces.



(a) Jim, (b) Jack, (c) Robert, (d) Lucy and (e) Margaret

**Fig. 10.9:** Projection of different facial poses of persons mapped on 12- dimensional space as points; we considered 9 poses per person that together form a cluster; all 9 points in a cluster, however, have not been shown for clarity.

After the structural organization of the SOM is over, we can classify a test image by the structure. Let  $\mathbf{IP}$  be the projected point corresponding to the test image. Let  $\mathbf{W}$  be the weight vector of a neuron. We first compute the  $\|\mathbf{IP} - \mathbf{W}\|$  for all neurons mapped on the surface at level 0 and identify the neuron where the measure is minimum. We then compute  $\|\mathbf{IP} - \mathbf{W}\|$  for all neurons of a surface in level 1, which is pointed to by the winning neuron at level 0. The process is thus continued until one of the leaves of the structure is reached. The search procedure in a SOM thus is analogous to best first search.



**Fig. 10.10:** Hierarchical organization of SOM: Faces clustered on a plane at closely mapped points are re-mapped onto lower planes for further classification.

## 10.9 Conclusions

The chapter introduced three different types of competitive learning models including PCA, SOFM and two-layered competitive learning. The last model can be configured for three different applications such as short-term memory, clustering and feature mapping. For reducing the entry of noise in a competitive learning network, a specialized shunting activation model is used prior to data submission to the input layer. The instability of Hebbian learning in the feedback layer of the competitive learning network has been demonstrated, and Oja's correction to handle the problem has been discussed. The principles of PCA and SOFM have been illustrated using face recognition problems.

## Exercise

- Find the eigenvalues and eigenvectors of the following matrix:

$$A = \begin{pmatrix} 8 & -6 & 2 \\ -6 & 7 & -4 \\ 2 & -4 & 3 \end{pmatrix}.$$

Also identify the principal eigenvector.

**[Hints:** (a) The characteristic equation for the above matrix is given by

$$\text{Det}(A - \lambda I) = 0, \text{ which on simplification yields}$$

$$\lambda(\lambda - 3)(\lambda - 15) = 0.$$

Thus the eigenvalues are 0, 3 and 15.

If  $x$ ,  $y$  and  $z$  are the components of an eigenvector corresponding to an eigenvalue  $\lambda$ , then

$$[A - \lambda I] \begin{bmatrix} x \\ y \\ z \end{bmatrix} = 0.$$

Substituting  $\lambda=0$  in the above equation, we have  $8x-6y+2z=0$ ,  $6x+7y-4z=0$  and  $2x-4y+3z=0$ . These equations determine a single linearly

independent solution, which may be taken as (1, 2, 2) so that every non-zero multiple of this vector is an eigenvector corresponding to  $\lambda=0$ . Similarly, for  $\lambda=3$  and 15 we obtain the eigenvectors (2, 1, -2) and (2, -2, 1) respectively.

(b) The eigenvalue  $\lambda=15$  is the largest among the three eigenvalues, and consequently its corresponding eigenvector is the principal eigenvector.]

2. Linsker used a modified Hebbian rule [3] given by

$$\Delta \mathbf{w} = \rho(y\mathbf{x} + a\mathbf{x} + \mathbf{b}y + \mathbf{d}).$$

It is also given that  $y = \mathbf{x}^T \mathbf{x}$  is the output of a neuron,  $\mathbf{C} = \langle \mathbf{x} \mathbf{x}^T \rangle$ ,  $\boldsymbol{\mu} = \langle \mathbf{x} \rangle$ . When  $\mathbf{d}=\mathbf{0}$  and  $b_i = -a$  for  $i = 1, 2, \dots, n$  show that

$$\langle \Delta \mathbf{w}_i \rangle = \rho [ \mathbf{C}_i \mathbf{w} + a (\mu_i - \sum_{j=1}^n \mu_j w_j) ]$$

where  $\langle \mathbf{x} \rangle$  denotes expectation of  $\mathbf{x}$ .

[Hints:  $\langle \Delta \mathbf{w} \rangle = \langle \rho(y\mathbf{x} + a\mathbf{x} + \mathbf{b}y + \mathbf{d}) \rangle$

$$= \langle \rho \{ (\mathbf{x} \mathbf{x}^T) \mathbf{w} + a\mathbf{x} + b(\mathbf{x}^T \mathbf{w}) + \mathbf{d} \} \rangle$$

With  $\mathbf{d}=\mathbf{0}$  and  $b_i = -a$  for  $i=1, 2, \dots, n$  and  $\langle \mathbf{x} \rangle = \boldsymbol{\mu}$ , the above expression reduces to

$$\rho [\mathbf{C}\mathbf{w} + a(\boldsymbol{\mu} - \mathbf{w}^T \boldsymbol{\mu} \mathbf{1})],$$

where  $\mathbf{1}$  denotes a column vector with all elements equal to 1.

Defining  $\mathbf{C}_i$  as the i-th row of  $\mathbf{C}$  matrix in the last expression, we finally obtain:

$$\langle \Delta \mathbf{w}_i \rangle = \rho [\mathbf{C}_i \mathbf{w} + a(\mu_i - \sum_{j=1}^n \mu_j w_j)]. ]$$

3. Show that Linsker's weight adaptation rule [3] can be obtained by computing average of the gradient descent learning rule given below:

$$\langle \Delta \mathbf{w} \rangle = -\rho \frac{\partial E}{\partial \mathbf{w}},$$

on the cost function  $E = - (1/2) \mathbf{w}^T \mathbf{C} \mathbf{w} + (\lambda/2) \left( 1 - \sum_j w_j \right)^2$

where  $\mathbf{C} = [C_{ij}]$  and  $\lambda = a\mu$ .

**[Hints:** Given  $E = - (1/2) \mathbf{w}^T \mathbf{C} \mathbf{w} + (\lambda/2) \left( 1 - \sum_j w_j \right)^2$

$$\langle \Delta \mathbf{w} \rangle = -\rho \frac{\partial E}{\partial \mathbf{w}}$$

$$= \rho [\mathbf{C}\mathbf{w} - ((\lambda/2) .2 (1 - \sum_j w_j) (-1))]$$

With  $\lambda = a\mu$ , the above expression reduces to

$$\langle \Delta \mathbf{w} \rangle = \rho [\mathbf{C}\mathbf{w} + a(\mu - \mathbf{w}^T \mu \mathbf{1})],$$

which is the weight adaptation rule of Linsker.]

4. The output  $y$  of a non-linear PCA [3] is given by

$$y = \sum_{i=1}^n \left( w_i + \sum_{j=1}^n w_{ij} x_j \right) x_i.$$

Show that this can be interpreted as the output of a linear unit with the input vector as

$$\mathbf{z} = [x_1 \ x_2 \ \dots \ x_n \ x_1^2 \ x_1 x_2 \ \dots \ x_1 x_n \ x_2^2 \ x_2 x_3 \ \dots \ x_n^2]^T,$$

and the weight vector as

$$\mathbf{w} = [w_1 \ w_2 \ \dots \ w_n \ w_{11} \ w_{12} \ \dots \ w_{1n} \ w_{21} \ w_{22} \ \dots \ w_{2n}]^T.$$

**[Hints:** Given:

$$y = \sum_{i=1}^n \left( w_i + \sum_{j=1}^n w_{ij} x_j \right) x_i$$

$$= \sum_{i=1}^n w_i x_i + \sum_{i=1}^n \sum_{j=1}^n w_{ij} x_j x_i$$

$$=[w_1 \ w_2 \ \dots \ w_n \ w_{11} \ w_{12} \ \dots \ w_{1n} \ w_{21} \ w_{22} \ \dots \ w_{2n}]^T [x_1 \ x_2 \ \dots \ x_n]$$

$$x_1^2 \ x_1x_2 \ \dots \ x_1x_n \ x_2^2 \ x_2x_3 \ \dots \ x_n^2]$$

$$= \mathbf{w}^T \mathbf{z}.$$

Thus,  $y$  can be interpreted as the output of a linear unit with input vector  $\mathbf{z}$  and weight vector  $\mathbf{w}$ .]

5. Show that the principal component resulting from non-linear PCA networks described above corresponds to the principal eigenvector of the matrix  $\langle \mathbf{z}\mathbf{z}^T \rangle$ .

**[Hints:** Since  $\mathbf{z}$  is the input vector and  $y$  is the output of the unit, change in weight vector is given by

$$\begin{aligned}\Delta \mathbf{w} &= \rho \mathbf{z} y \\ &= \rho \mathbf{z} (\mathbf{z}^T \mathbf{w}) \\ &= \rho (\mathbf{z} \cdot \mathbf{z}^T) \mathbf{w}\end{aligned}$$

Taking expectation operator on both sides of the last expression we have:

$$\langle \Delta \mathbf{w} \rangle = \rho \langle \mathbf{z}\mathbf{z}^T \rangle \langle \mathbf{w} \rangle$$

Thus, the principal component resulting from non-linear PCA networks correspond to the principal eigenvector of the matrix  $\langle \mathbf{z}\mathbf{z}^T \rangle$ .]

## References

- [1] Grossberg, S., "Some networks that can learn, remember, and reproduce any number of complicated space-time patterns," *J. Math. Mech.*, vol. 1, no. 19, pp. 53-91, 1969.
- [2] Haykin, S., *Neural Networks: A Comprehensive Foundation*, Prentice-Hall, NJ, 2000.
- [3] Hassoun, M. H., *Fundamentals of Artificial Neural Networks*, Prentice-Hall, NJ, 1995.
- [4] Hertz, J. A., Computing with Attractors, In *The Handbook of Brain Theory and Neural Networks*, Arbib, M. A. (Ed.), Cambridge, MIT Press, MA, pp. 230-234, 1991.
- [5] Jolliffe, I. T., *Principal Component Analysis*, Springer-Verlag, NY, 1986.

- [6] Kosko, B., *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*, Prentice-Hall, Englewood Cliffs, NJ, 1992.
- [7] Kohonen, T., "Analysis of simple self-organizing process," *Biological Cybernetics*, vol. 44, pp. 135-140, 1982.
- [8] Nasrabadi, N. M. and King, R. A., "Image Coding Using Vector Quantization: a Review," *IEEE Trans. Communications*, vol. 36, pp. 957-971, 1988.
- [9] Oja, E., "Neural networks, principal components and subspaces," *Int. J. Neural Systems*, vol. 1, no. 1, pp. 61-68, 1989.
- [10] Yegnanarayana, B., *Artificial Neural Networks*, Prentice-Hall of India, 1999.

# 11

## Neuro-dynamic Programming by Reinforcement Learning

*The chapter introduces the principles of reinforcement learning that rests on the foundation of the penalty-reward mechanism of our natural learning process. It begins with Q-learning and its variants and discusses the scope of realization of Q-learning on neural networks. Two distinct models of neural topologies have been considered for on-line adaptation of weights in the neural networks, following the dynamics of the Q-learning law. The principles of the Q-learning algorithm have been illustrated with the well-known grid-world problem of mobile robots. The convergence analysis of the Q-learning algorithm is presented, and the scope of extension of the Q-learning algorithm in multi-agent learning systems has been addressed at the end of the chapter.*

### 11.1 Introduction

The last three chapters covered three different paradigms of machine learning using neural nets. This chapter introduces a completely different approach to

learning, hereafter referred to as reinforcement learning. This learning paradigm is neither supervised nor purely unsupervised. In fact, an external expert provides an immediate reward of each action of the learner on the environment. These rewards are cumulatively used by specialized learning algorithm to gain knowledge about the environment of the learner. The learning models to be employed in this chapter are based on the classical theories of dynamic programming.

In dynamic programming [1] decisions are made in stages, with the outcome of each decision being predictable to some extent before the next decision is made. The fundamental aspects of dynamic programming are that decisions cannot be taken in isolation. Rather, the desire for a low cost at the present must be balanced against the undesirability of high costs in the future. This is well known as the *credit assignment problem* because a credit or blame needs to be assigned to each one of a set of interacting decisions. Because of the similarity of the learning models to be presented shortly with classical dynamic programming, we call them neurodynamic-programming models.

To illustrate the principle of reinforcement learning, let us consider the problem of a mobile robot that has a fixed goal of docking onto its battery charger whenever its power level is low. An expert, suppose, assigns a positive reward value of 100 to state transitions of the robot that immediately provides a connection to the charger and a reward of zero to all other state transitions.

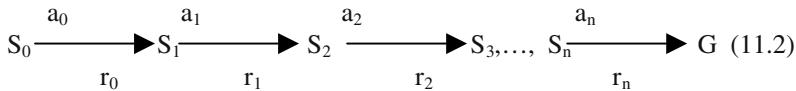
The task of the robot is to perform sequences of actions such as move forward, turn right or left etc., observe their consequences, and learn a control policy. The desired control policy in the present context is to determine the actions in sequence that together maximize the reward accumulated over time by the robot

The principle of learning a control policy by an agent is presented below using a state transition model (11.1). The model describes an agent and its environment. The agent acts on its environment and by doing so it undergoes a state transition. Given a particular goal G for the agent. An expert external to the agent computes a real-valued reward for each state transition. To formalize suppose the agent at state  $S_i$  does an action  $a_i$  on its environment, and receives a reward  $r_i$  for its transition to state  $S_j$ . This can be denoted by

$$S_i \xrightarrow{a_i} S_j \quad (11.1)$$

$r_i$

If the agent goes on acting on its environment until it reaches a goal state G, we can describe the state transitions as follows:



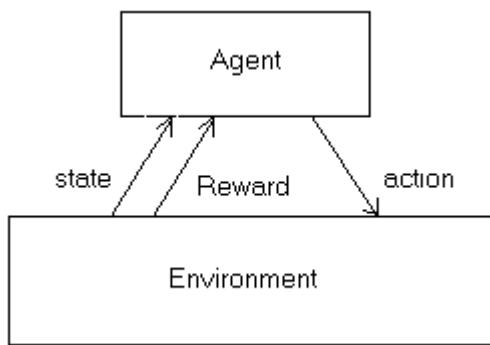
where  $a_0, a_1, a_2, \dots, a_n$  are the sequence of actions on states  $S_0, S_1, S_2, \dots, S_n$  with respective rewards  $r_0, r_1, r_2, \dots, r_n$ . The objective of the agent is to learn to choose sequence of actions:  $a_0, a_1, a_2, \dots, a_n$  that maximize the cumulative reward

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \dots + \gamma^n r_n$$

$$= \sum_{i=0}^n \gamma^i r_i \quad (11.3)$$

where  $\gamma$  is discounting factor and its typical value lies in  $[0,1]$ .

It may be mentioned here that the scope of the proposed principle of learning is not only restricted to mobile robots, but it also has extensive applications in manufacturing optimization problems. The problem under reference is concerned with determining a sequence of manufacturing actions to maximize the reward of values of goods produced minus the costs involved. The other application of reinforcement learning includes learning the moves to win a game. In fact, Tesauro [10] presented a TD-GAMMON program, capable of employing reinforcement learning to become a world-class backgammon player.



**Fig. 11.1:** An agent acting on its environment changes state and receives a reward for its action with respect to a given goal G.

The fundamental differences of reinforcement learning with other function approximation supervised learning are presented below:

- ◆ **Delayed reward:** Let  $S$  denote a state and  $P$  denote a policy that maps state  $S$  to action  $A$ , i.e.,  $P: S \rightarrow A$ . In supervised learning training examples  $\langle S, P(S) \rangle$  are given. In reinforcement learning  $P(S)$  is not known, but as immediate reward of action  $A$  on state  $S$  is available, the rewards the agent receives are delayed and the cumulative reward,

$$\sum_{i=0}^n \gamma^i r_i,$$

thus cannot be evaluated until the agent reaches the goal state.

- ◆ **Tradeoff in selection of old or new states:** In reinforcement learning the learner faces a tradeoff between selection of known actions (& states) and exploration of new actions (& states).
- ◆ **Partially observable states:** On occasions, the agent perceives only partially observable states. For example, a mobile robot with only front sonar transducer is unable to determine the range of obstacles around it in all directions. Consequently, it has to consider its old sensory data along with its current sensor readings to determine its action. The best policy may correspond to the choice of actions specifically to improve the observability of the environment.
- ◆ **Exploiting past experience for future learning:** A mobile robot has several goals such as planning a path through a narrow corridor, docking onto the battery charger and coordination among the robots for complex tasks like box-pushing or carrying a large stick to a goal position. Suppose the robot has already earned an experience in path planning. It may then use this knowledge to dock onto the battery charger. Thus past experience may be used to reduce sample complexity in learning new tasks.

## 11.2 Formulation of the Reinforcement Learning Paradigm

In this section we present the formulation of the reinforcement learning paradigm [2-8]. We remember that in reinforcement learning, an agent acts on the environment to cause a transition of environmental state, and receives a immediate reward for its action. The change of state due to agent's action on the environment may be either deterministic or non-deterministic depending on the nature of the problem. The general formulation of the reinforcement learning problem can be recasted as a Markov Decision Process (MDP)

Let

- S be a set of distinct states that an agent can perceive,
- A be a set of actions that the agent can perform on states in S,
- $s_t$  be a state in S that the agent perceives at time t,
- $a_t$  be an action that the agent performs at time t on perceiving state  $s_t$ ,
- $r_t = r(s_t, a_t)$  be an immediate reward that the agent receives at time t due to its action  $a_t$  at state  $s_t$ .
- $s_{t+1} = \delta(s_t, a_t)$  is the succeeding state of  $s_t$  obtained by an action  $a_t$ .

$\delta(s_t, a_t)$  and  $r(s_t, a_t)$  are in general non-deterministic function. But to start with the simplest formulation of reinforcement learning, we consider them to be deterministic.

The agent in the present context is to learn a *policy*  $P: S \rightarrow A$  for selecting its next action  $a_t$  based on its current state  $s_t$ , i.e.,  $P(s_t) = a_t$ . One way of determining the policy P is to attempt to maximize the cumulative rewards the agent receives by its actions on the environment. Given an arbitrary initial state  $s_t$ , we define a cumulative reward  $V^P(s_t)$  achieved by following a policy P starting with an initial state  $S_t$  as follows :

$$\begin{aligned} V^P(s_t) &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \text{upto } \infty \\ &= \sum_{i=0}^{\infty} \gamma^i r_{t+i}, \quad 0 \leq \gamma \leq 1. \end{aligned} \tag{11.4}$$

The sequence of rewards  $r_{t+i}$  is generated at state  $s_{t+i}$ , beginning at state  $s_t$  by repeatedly using the policy P. The immediate reward  $r_t$  at state  $s_t$ , is given the highest weight 1 in expression (11.4), whereas each successive reward is discounted by an additional factor of  $\gamma$ . It is to be noted that the smaller the value of  $\gamma$  in  $[0, 1]$ , the less is the contribution of the successive rewards in (11.4). Increasing the immediate reward and discounting the future rewards has a logical significance, as we prefer to enhance the cumulative reward in the shortest span of time, however small.

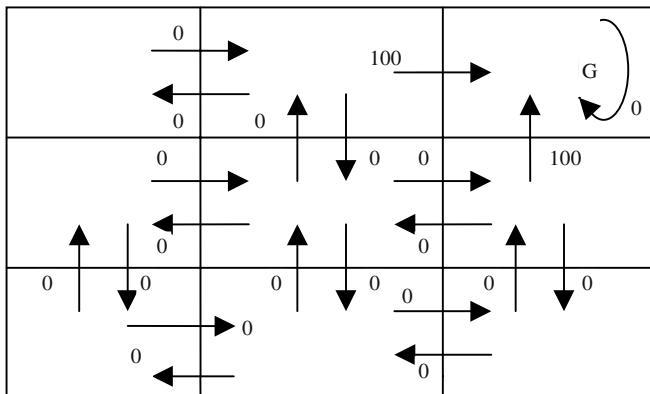
The agent's learning task now can be precisely presented. The agent should learn a policy P that maximizes  $V^P(s)$  for all state s. Such policy is called an optimal policy and is denoted by

$$P^* \equiv \operatorname{argmax} V^P(s), \quad \text{for all } s \tag{11.5}$$

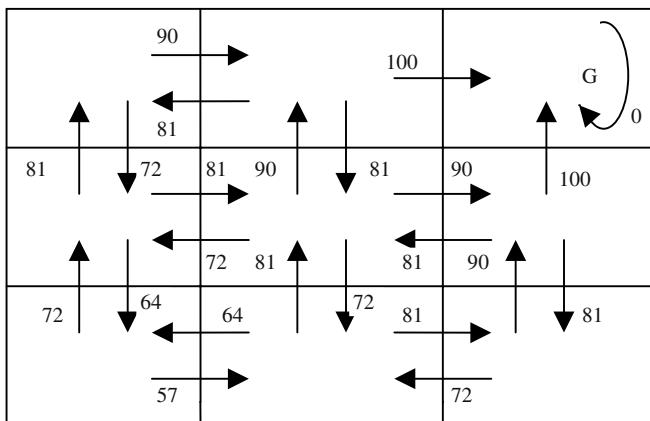
$V^P(s)$  for  $P = P^*$  is hereafter denoted by  $V^*(s)$  for simplicity in notation.  $V^P(s)$  in general is called a *value function* and  $V^*(s)$  is hereafter referred to as the *optimal value function*.

What does  $V^*(s)$  actually mean? It denotes the maximum discounted cumulative reward that an agent receives starting from state  $s$  following an optimal policy  $P^*$ .

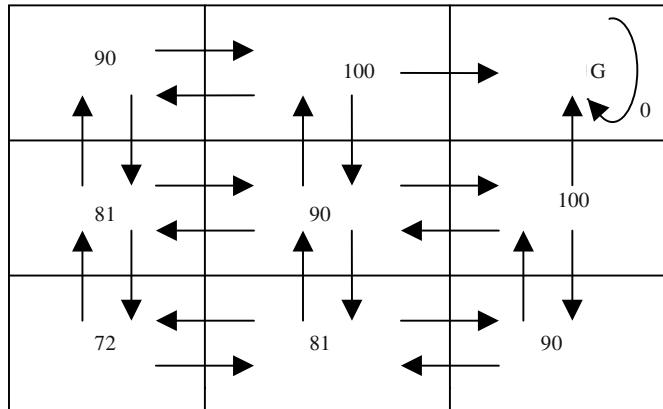
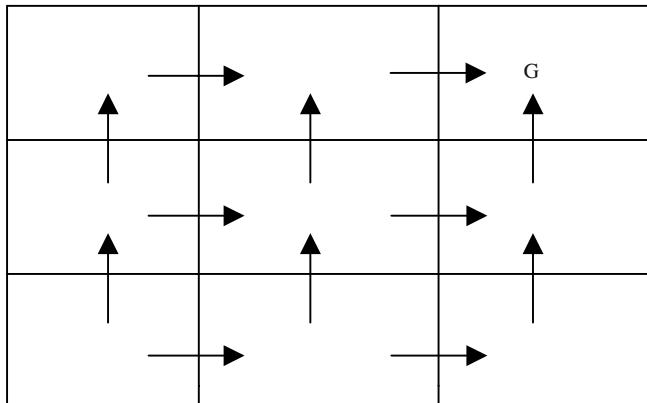
**Example 11.1:** Consider a robot's world represented by a 6-cell rectangular grid. A robot can move from its current position to its neighboring cell. Diagonal movements are not allowed. There is a fixed goal position, labeled as G, in the structure (Fig. 11.2). The arrows in the diagram denote a possible action that the agent performs for transition of states following the direction of the arrowhead. The numbers labeled against the arrows denote the immediate reward  $r(s, a)$  obtained at state  $s$  due to action  $a$  (Fig. 11.2(a)).



(a) Immediate reward values labeled against the arrows.



(b)  $Q(s, a)$  values with reference to Fig.(a).

(c) The  $v^*(s)$  values indicated in the cells.

(d) One optimal policy described by the arrows.

**Fig. 11.2:** Computing Q-values and  $V^*(s)$  values from immediate reward with  $\gamma=0.9$ , and hence finding the optimal policy.

It is clear from Fig. 11.2(a) that immediate reward is 100 when the agent reaches the goal G and else it is zero. In Fig. 11.2(b), we computed the Q-values to be defined shortly by adding immediate reward with subsequent discounted rewards. For example, the Q-value of the center is

$$\begin{aligned} 0 + 100\gamma + 0\gamma^2 + 0\gamma^3 & \dots \\ = 90, \end{aligned}$$

and the Q-value of the center at the last row is

$$\begin{aligned} 0 + 0\gamma + 100\gamma^2 + 0\gamma^3 & \dots \\ = 81. \end{aligned}$$

The  $V^*(s)$  value presented in Fig. 11.2(c) is computed by putting the maximum Q-value labeled at the outgoing arcs of a cell. Thus,  $V^*$ -value of the cell at the center-bottommost row is  $\max \{81, 64, 81\}=81$ . The  $V^*(s)$  of other cells have been computed analogously.

In Fig. 11.2(d), we have shown one optimal policy derived by following the directions of maximum Q-values marked against the outgoing arcs of each cell.

### 11.3 Q-Learning

It is clear from example 11.1 that given the next states  $\delta(s_t, a_t)$  due to an action  $a_t$  at state  $s_t$  and the immediate rewards  $r(s_t, a_t)$ , we can determine an optimal policy. Now, given a state  $s$ , what action should the agent prefer? It should prefer that action that offers the agent maximum cumulative reward. This action is called optimal action. The cumulative reward for such *optimal action* is given by

$$P^*(s) = \operatorname{argmax} [ r(s, a) + \gamma V^*(\delta(s, a)) ] \quad (11.6)$$

where  $\delta(s, a)$  is the next state due to action  $a$  at state  $s$ .

Thus for known  $r$  and  $\delta$ , determining optimal policy  $P^*$  is straight-forward.

The function that computes the current reward  $r(s, a)$  and the discounted rewards by an optimal policy is called a Q-evaluation function. Formally, it is given by

$$Q(s, a) = r(s, a) + \gamma V^*(\delta(s, a)) \quad (11.7)$$

Now from (11.6) and (11.7), we can easily write

$$P^*(s) = \operatorname{argmax} Q(s, a) \quad (11.8)$$

What is the advantage of rewriting (11.6) in the form of (11.8)?

This is because of the fact that the agent even in absence of  $r$  and  $\delta$  can find optimal actions using the Q-function. In fact, the agent determines the optimal action at a state  $s$  by identifying the action with the largest Q-value.

The Q-evaluation function can easily be constructed in a recursive manner as follows. We can write:

$$V^*(s) = \max_{a'} Q(s, a') \quad (11.9)$$

Now, combining (11.7) and (11.9) we have

$$Q(s, a) = r(s, a) + \gamma \max_{a'} Q(\delta(s, a), a') \quad (11.10)$$

The recursive definition of  $Q$  provides a basis of the Q-learning algorithm [11] to be presented shortly. Since the learner guesses the approximate value of  $Q(s, a)$ , we denote it by a separate notation  $Q'(s, a)$ . The learner in the algorithm thus maintains a table of  $Q'(s, a)$  for table indices  $(s, a)$ . Initially,  $Q'$  for given  $(s, a)$  is unknown. So, initial entries of the table are zero.

The learner then updates its  $Q'$  values using the following rules with known reward  $r(s, a)$ .

$$Q'(s, a) \leftarrow r(s, a) + \gamma \max Q'(s', a') \quad (11.11)$$

Where  $s' = \delta(s, a)$ .

The algorithm for Q-learning is outlined below precisely.

**Procedure Q-learning ( $r(s,a), Q'(s,a)$ )**

**For** state  $s=s_1$  to  $s_n$

**For** action  $a=a_1$  to  $a_m$

        Initialize  $Q'(s, a) \leftarrow 0$

**End for;**

**End for;**

Observe the current state  $s$ ;

**Repeat**

    Select an action  $a \in \{a_1, a_2, \dots, a_m\}$  and execute it;

    Receive an immediate reward  $r(s, a)$ ;

    Observe the new state  $s' \leftarrow \delta(s, a)$ ;

    Update :  $Q'(s, a) \leftarrow r(s, a) + \gamma \max_{a'} Q'(s', a')$ ;

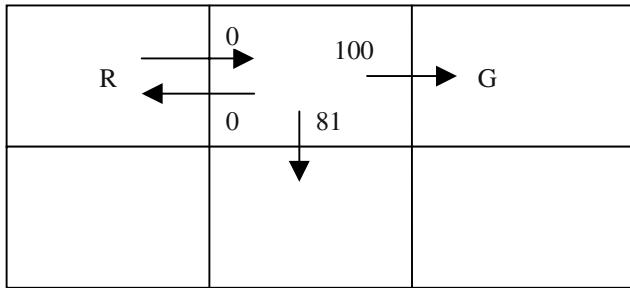
$s \leftarrow s'$

**For ever.**

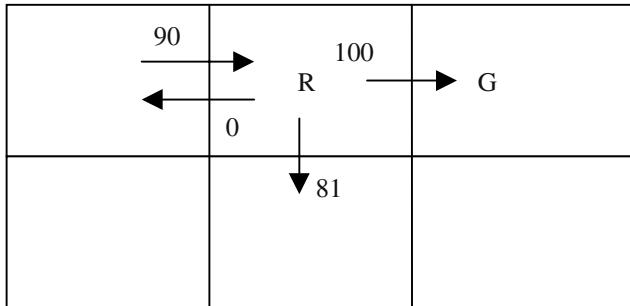
The first part of the algorithm initializes  $Q'(s, a)$  for  $s \in \{s_1, s_2, \dots, s_n\}$  and  $a \in \{a_1, a_2, \dots, a_m\}$  to zero. Then depending on the observed state  $s$ , the algorithm selects an action  $a$  from the set  $\{a_1, a_2, \dots, a_m\}$  and executes it. A reward  $r(s, a)$  is

received and a new state  $\delta(s, a)$  is determined. The algorithm then updates  $Q'$ -values and redefine  $s=s'$ . The algorithm continues looping for ever. In practice, the algorithm is terminated when  $Q'(s, a)$  for all  $(s, a)$  reach steady values.

**Example 11.2:** To illustrate the Q-learning algorithm, let us consider a 6-cell rectangular grids as the workspace of a mobile robot (Fig 11.3). Let  $Q'(s, a)$  be labeled against the directed arcs representing actions by the agent



(a) Initial state  $S_1$  in an illustrative Q-learning problem.



(b) Next state  $S_2$  after movement of the robot R to the right.

**Fig. 11.3:** Two consecutive states of a robot's workspace with  $Q'$  values labeled against the arrows.

Assuming  $\gamma=0.9$ , we now update the  $Q'$ -value for the arrow from the left topmost cell to its next central cell due to the movement of the robot R to the right. It is to be noted that immediate reward for movement of the robot to the

right is zero, as it is a non-goal position. The Q'-value of the transition from the left topmost cell to its next cell is now computed by

$$\begin{aligned} Q'(s_1, \text{right-movement}) &\leftarrow r + \gamma \max_{a'} Q'(s_2, a') \\ &= 0 + 0.9 * \max\{81, 0, 100\} \\ &= 90. \end{aligned}$$

Consequently, the Q'-value of the incoming arrow towards the top central cell from its left cell is changed from 0 to 90 in Fig. 11.3(b)

What do we learn from this example? The Q'-value is estimated backward from the new state to the old state.

In example 11.2 we just demonstrated how the learning equation (11.11) is used for adaptation of Q'-values. The example 11.3 below illustrates how the agents gradually update Q'-values from zero values.

**Example 11.3:** Consider the same robot's world as illustrated in example 11.2. Suppose, the Q'-values are all zero and the immediate rewards for all actions except to reaching the goal is zero (Fig. 11.3(a)). The immediate reward for reaching the goal in a single step is 100. Also assume that  $\gamma=0.9$ . In Fig. 11.4 we demonstrate the backward movement of the Q'-values. Since the initial Q'-values are all zero, there will be no change in Q'-values until the robot accidentally reaches the goal and contributes its current  $\gamma$ -value to Q'-value. The Q'-value of the actions for transition to the neighboring cells of the goal thus can be updated to non-zero values.

In this example, we show three steps of updating of Q'-values for the 2-step of motion of the robot from the left topmost cell to the goal through the central cell.

**Step 1:** Suppose the robot makes a movement to the central cell from its left cell by a moveright action (Fig. 11.4(b)). Here,  $r=0$ , Q'-values of outgoing arcs of the top central cell too are zero. Thus

$$Q'(s_1, \text{moveright})$$

$$= r + \gamma \max_{a'} Q'(s_2, a')$$

$$a'$$

$$= 0 + 0.9 * \max(0, 0)$$

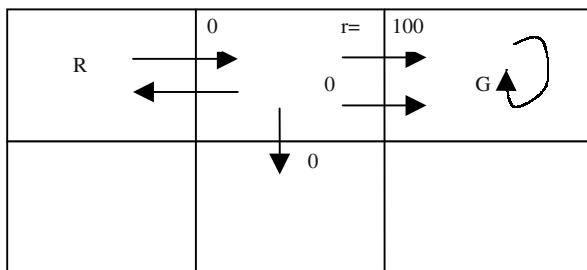
$$= 0.$$

**Step 2:** The robot now moves one step right further and reaches the goal position (Fig. 11.4(c)). Here,  $r=100$ , but  $Q'$  for all outgoing arcs of the top rightmost cell are zero. Thus

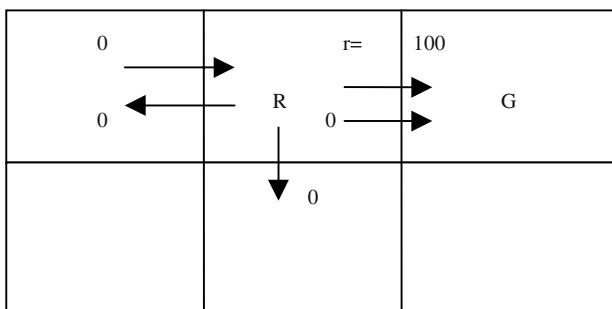
$$\begin{aligned} Q'(s_2, \text{moveright}) &= r + \gamma \max_{a'} Q'(s_3, a') \\ &= 100 + 0.9 * \{0, 0\} \\ &= 100. \end{aligned}$$

**Step 3:** Suppose, we again compute the  $Q'$ -value for the rightmovement of the robot from top leftmost cell to the top central cell (Fig. 11.4(d)). Here,  $Q'(s_2, \text{moveright})=100$ ,  $r=0$ . Thus

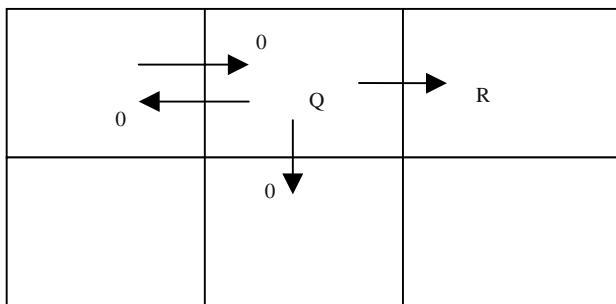
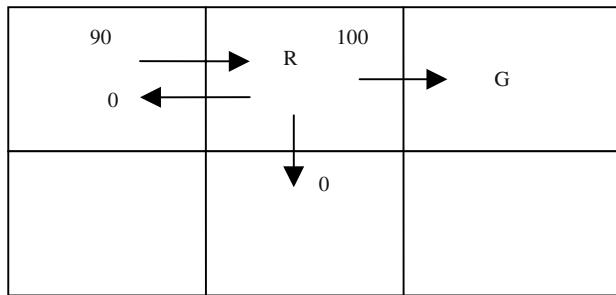
$$\begin{aligned} Q'(s_1, \text{moveright}) &= r + \gamma \max_{a'} Q'(s_2, a') \\ &= 0 + 0.9 * \max\{0, 100, 0\} \\ &= 90. \end{aligned}$$



(a) States  $s_1$  with  $Q'$ -values.



(b) State  $s_2$  with  $Q'$ -values.

(c) State  $s_3$  with  $Q'$ -values.(d) Re-computing  $W'$ -values for transition to state  $s_2$ .**Fig. 11.4:** Illustrating the principle of updating  $Q'$  in a robot's world.

## 11.4 Convergence of the Q-Learning Algorithm

A question that naturally arises: whether the  $Q'$ -values converges towards the exact  $Q$ -values of the function. The answer is in the affirmative. To prove the convergence of  $Q'$  towards  $Q$ , we make the following assumptions:

- The system is a deterministic Markov decision process (MDP).
- For all  $s$  and  $a$ ,  $r(s, a)$  should be bounded, i.e.,  $|r(s, a)| < c$ .
- The agent should visit every state action pair  $(s, a)$  infinitely often.

The key idea underlying the proof of convergence is that the table entry  $Q'(s, a)$  with the largest error must have its error reduced by a factor of  $\gamma$  whenever the  $Q'$  is updated [6]. To prove the convergence of the Q-learning algorithm, we first define a list of parameters.

Let

$Q_n'$  be the agent's table of estimated Q-value after n updates,  
 $\Delta_n$  be the maximum error in  $Q_n'$ ,  $|r(s, a)| < c$ , a constant.

Then

$$\Delta_n \equiv \text{Max } |Q_n'(s, a) - Q(s, a)| \quad (11.12)$$

Now, suppose any table entry  $Q_n'(s, a)$  is updated on iteration (n+1) and the magnitude of error in the revised estimate is

$$\begin{aligned} & |Q_{n+1}'(s, a) - Q(s, a)| \\ &= |(r + \gamma \max_{a'} Q_n'(s', a')) - (r + \gamma \max_{a'} Q(s', a'))| \\ &= \gamma |\max_{a'} Q_n'(s', a') - \max_{a'} Q(s', a')| \\ &\leq \gamma \max_{a'} |Q_n'(s, a') - Q(s', a')| \\ &\leq \gamma \max_{s'', a'} |Q_n'(s'', a') - Q(s'', a')| \\ &= \gamma \Delta_n \end{aligned}$$

$$\text{Thus } |Q_{n+1}'(s, a) - Q(s, a)| \leq \gamma \Delta_n \quad (11.13)$$

Since the largest error in the initial table is  $\gamma \Delta_0$  is bounded as  $Q_0'(s, a)$  and  $Q(s, a)$  are bounded, thus, the largest error is  $\leq \gamma \Delta_0$ . After k such intervals, the error is  $\gamma^k \Delta_0$ . Now, as  $0 \leq \gamma \leq 1$ , as  $k \rightarrow \infty$ ,  $\gamma^k \Delta_0 \rightarrow 0$ . Hence, the Q-learning converges with zero error.

## 11.5 Non-deterministic Rewards and Actions

Until now we discussed deterministic Markov process to model reinforcement learning. The Q-learning, for instance, is one such deterministic learning algorithm. A question that naturally arises: how should we classify reinforcement learning into deterministic and nondeterministic types? The classification in fact, is based on the reward function  $r(s, a)$  and state transition  $\delta(s, a)$ . In deterministic learning  $r(s, a)$  and  $\delta(s, a)$  are fixed, but in nondeterministic learning these two parameters are probabilistic. To be precise,

suppose the environment is nondeterministic, i.e., an action in state  $s$  may cause transitions to one of many possible states  $s_1, s_2, \dots, s_k$ . Thus transition to a particular state  $s'$  from a given state  $s$  by action  $a$  has a probability  $P(s' | s, a)$ . Further, since an action  $a$  at state  $s$  may cause a transition to a state  $s' \in \{s_1, s_2, \dots, s_k\}$ , the reward function  $r(s, a)$  also is probabilistic. When the probability distributions  $r$  and  $\delta$  are functions of  $s$  and  $a$  only (i.e. they do not depend on previous states or actions), we call the system a nondeterministic Markov decision process.

Because of multiplicity in transitions from one state, it is difficult to generalize the learning behavior of the agent. One simple way to handle the problem is to consider the expected value of the nondeterministic outcomes of the discounted cumulative reward received by applying this policy:

$$V^P(s_t) \equiv E \left[ \sum_{i=0}^{\infty} \gamma^i r_{t+i} \right] \quad (11.14)$$

where the sequence of rewards  $r_{t+i}$  is generated by following policy  $P$  beginning at state  $s$ .

The deterministic Q-learning equation (11.7) introduced earlier now can be generalized as follows

$$\begin{aligned} Q(s, a) &\equiv E [ r(s, a) + \gamma V^*(\delta(s, a)) ] \\ &\equiv E [ r(s, a) ] + \gamma E[V^*(\delta(s, a))] \\ &\equiv E [ r(s, a) ] + \gamma \sum_{s'} P(s' | s, a) V^*(s') \end{aligned} \quad (11.15)$$

We can also express (11.15) in a recursive manner as presented below:

$$Q(s, a) \equiv E [ r(s, a) ] + \gamma \sum_{s'} P(s' | s, a) \max_{a'} Q(s', a') \quad (11.16)$$

which is generalization of (11.11).

Since the reward function  $r(s, a)$  returns different rewards each time the transition  $\langle s, a \rangle$  is repeated, the training rule (11.16) will repeatedly alter the  $Q'(s, a)$  values, even if they are initialized correctly in the  $Q'$  table. To overcome this problem, an alternative training rule is suggested. The new rule presented in (11.17) below takes a decaying weighted average of the current  $Q'$ -value and the revised estimate.

$$Q_n'(s, a) \leftarrow (1-\alpha_n)Q_{n-1}'(s, a) + \alpha_n [ r + \max Q_{n-1}'(s', a') ] \quad (11.17)$$

where

$Q'_n$  denotes the agent's estimate on the n-th iteration,

$$\alpha_n = \frac{1}{1 + \text{Visits}_n(s, a)},$$

and  $\text{Visits}_n(s, a)$  is the total number of times the state action pair  $(s, a)$  has been visited.

The training rule (11.17) has a decaying weight of  $Q_{n-1}'$ . It may be noted that if  $\alpha_n = 1$ , then learning equation (11.17) becomes similar with the deterministic equation (11.11). Watkins and Dayan [12] have shown that the value of  $\alpha_n$  is an important parameter among many others that satisfy the condition for convergence of the nondeterministic learning equation (11.17)

## 11.6 Temporal Difference Learning

The Q-learning algorithm gradually reduces the difference in Q-value estimates of successive states. Temporal difference learning may be considered as a further generalization of Q-learning. In temporal difference learning, the learning process attempts to reduce the discrepancies between estimates made by the agent at different times. The generalization of Q-learning to temporal difference learning lies in reducing discrepancies between a state and its more distant descendants or ancestors rather than between successive states of Q-learning.

It may be remembered that the Q-learning training rule discussed earlier calculates a training value for  $Q'(s_t, a_t)$  in terms of the values for  $Q'(s_{t+1}, a_{t+1})$  where the state  $s_{t+1}$  is generated by applying an action  $a_t$  to the state  $s_t$ . Let  $Q^{(1)}(s_t, a_t)$  be the training value calculated by one step lookahead function as follows:

$$Q^{(1)}(s_t, a_t) \equiv r_t + \gamma \max_a Q'(s_{t+1}, a) \quad (11.18)$$

Alternatively, we can compute a training value for  $Q^{(2)}(s_t, a_t)$  based on the observed rewards in two steps, given by

$$Q^{(2)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 \max_a Q'(s_{t+2}, a) \quad (11.19)$$

Similarly, for n-steps observed rewards, we can write:

$$Q^{(n)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \dots + \gamma^{(n-1)} r_{t+n-1} + \gamma^n \max_a Q'(s_{t+n}, a) \quad (11.20)$$

In late eighties Sutton [9] proposed a general method for blending the above alternative formulation of training estimates. The method is popularly called TD( $\lambda$ ). A constant  $\lambda$  in [0,1] is used in TD ( $\lambda$ ) to combine the estimates obtained from different lookahead distances. The learning equation that implements the above issue is presented below:

$$Q^\lambda(s_t, a_t) \equiv (1-\lambda)[Q^{(1)}(s_t, a_t) + \lambda Q^{(2)}(s_t, a_t) + \lambda^2 Q^{(3)}(s_t, a_t) + \dots] \quad (11.21)$$

A recursive formulation of  $Q^\lambda(s_t, a_t)$ , equivalent to the above definition, is also available in the literature [6]. It is expressed as

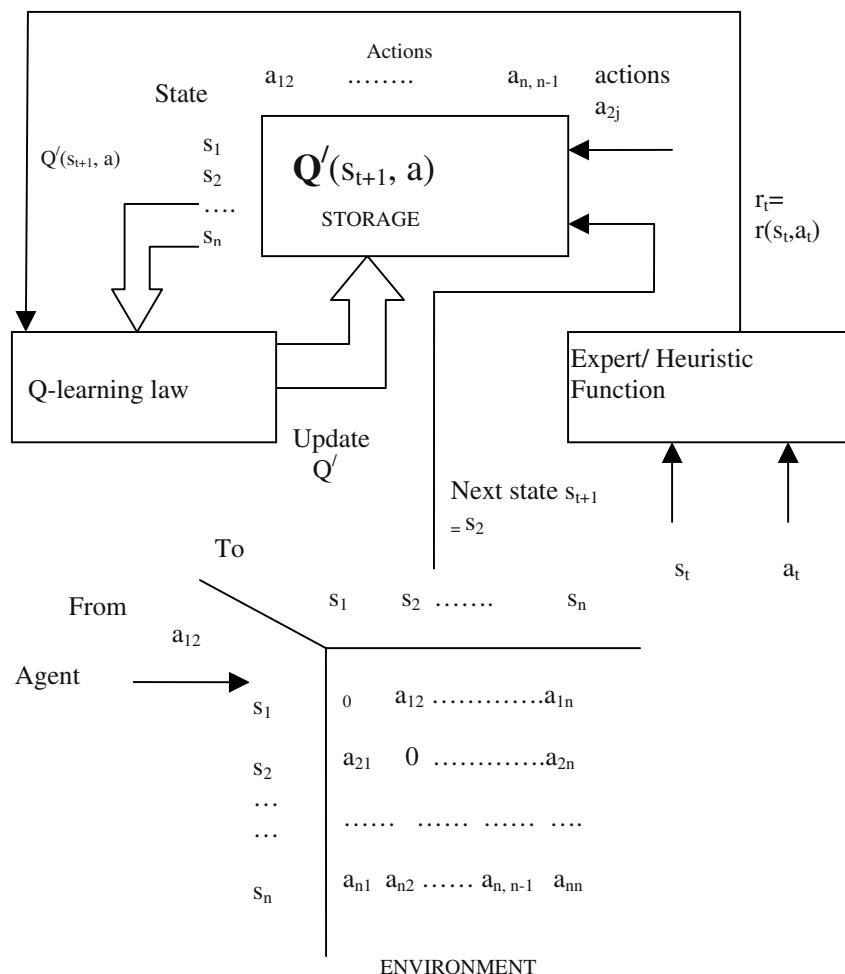
$$Q^\lambda(s_t, a_t) \equiv r_t + \gamma [(1-\lambda) \max Q'(s_t, a_t) + \lambda Q^\lambda(s_{t+1}, a_{t+1})] \quad (11.22)$$

It is to be noted that a choice of  $\lambda=0$  in (11.22) yields  $Q^{(1)}$ , which takes care of only one-step discrepancies in the  $Q'$ -estimates. With increase in  $\lambda$  the algorithm places increasing emphasis on discrepancies based on more distant lookaheads. At  $\lambda=1$ , only the observed  $r_{t+1}$  values are considered with no contribution from the current  $Q'$ -estimate.

## 11.7 Neural Reinforcement Learning

It has been discussed that the Q-values for situation –action pairs  $\langle s, a \rangle$  need to be stored in a 2-dimensional array before the learning process is invoked. Instead of using look-up tables, one simple way to memorize Q-values with its situation-action pairs is to employ neural networks. The initial state is constituted by the weights of the neural networks. The memory requirement for the system to store the knowledge is then defined, *a priori*, by the number of connections in the network. It is irrespective of the number of explored situation-action. A schematic diagram depicting the internal states of the system by a neural net is presented in Fig. 11.5.

In Fig. 11.5, we represented the environment by a matrix  $[s_{ij}]_{n \times n}$ . The  $(i, j)$ -th element of the matrix denotes an action  $a_{ij}$  that causes the environment to have a transition from state  $i$  to state  $j$ . For instance, suppose the agent does an action  $a_{12}$  in state  $s_1$ , causing a transition to state  $s_2$ . The state  $s_2$  and action  $a_{2j}$ ,  $\forall j$  are supplied to the  $Q'$ -storage, and the storage returns a value of  $Q'(s_2, a_{2j})$ . An expert determines an immediate reward  $r_t = r(s_t, a_t)$  and submits the results to the Q-learning unit. The Q-learner updates the values of  $Q'(s_2, a_{2j})$ ,  $\exists j$  in the  $Q'$ -storage.

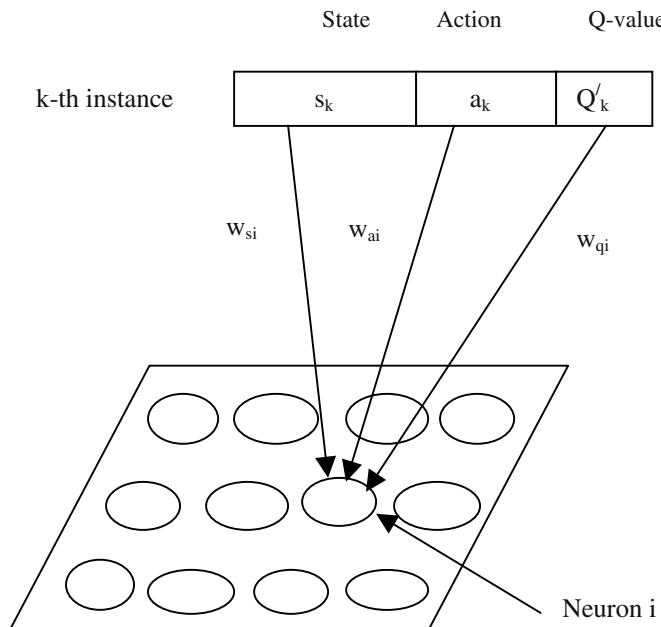


An agent executes an action  $a_{12}$  on state  $s_1$ , causing a transition of the system to state  $s_2$ . An expert/ heuristic function evaluator evaluates the immediate reward  $r_t$  for the said action. The  $Q'$  value stored corresponding to action  $a_{2j}$  and next state  $s_{t+1}$  is used later with immediate reward  $r_t$  to update and re-store the  $Q'$  value in the appropriate location.

**Fig. 11.5:** A schematic diagram of queue learning.

It is indeed important to note that the static  $Q'$ -storage can be realized using a supervised neural learning algorithm, such as the back-propagation algorithm. For training the net, we, however, require a prior database of  $Q'$  against the states and actions.

The static storage  $Q'$  can also be realized by Self-Organizing Feature Map (SOFM). For training the SOFM, we construct a 2-dimensional space of neurons each having a weight vector with 3 components. Let the components for the  $i$ -th neuron be  $w_{si}$ ,  $w_{ai}$  and  $w_{qi}$ , where  $w_{si}$  corresponds to the state,  $w_{ai}$  corresponds to the action and  $w_{qi}$  denotes the measure of the Q-values. All the training instances are mapped onto the SOFM one after another.



**Fig. 11.6:** Mapping of the training instances onto the 2-dimensional array.

Fig. 11.6 explains the mapping of training instances onto the SOFM. Let the  $k$ -th training instance be  $\langle s_k, a_k, Q'_k \rangle$ , where  $s_k$ ,  $a_k$  and  $Q'_k$  respectively denote a state, an action and the Q-value for the action  $a_k$  on the state  $s_k$ . Now, for each neuron in the 2-D neuronal space, we determine the Euclidean distance between the  $\langle s_k, a_k, Q'_k \rangle$  and  $\langle w_{si}, w_{ai}, w_{qi} \rangle$ , and identify the neuron  $j$  where

$$d_j = [(s_k - w_{sj})^2 + (a_k - w_{aj})^2 + (Q'_k - w_{qj})^2]^{1/2} \quad (11.23)$$

is minimum.

After the  $j$ -th neuron is identified on the 2-D space, the Q-value of the  $j$ -th neuron is updated using the Q-learning rule given below:

$$Q' \leftarrow r + \beta \max_{a'} Q'(s, a') \quad [(11.10) \text{ rewritten}]$$

where  $a'$  denotes the best action in the next sensory instant,  $r$  denotes the immediate reward, and  $\beta$  is a constant where  $0 < \beta < 2$ .

To ensure association among the neighborhood neurons around the selected neuron, we use:

$$\left. \begin{array}{l} w_{sn} \leftarrow w_{sn} + (w_{sn} - w_{sj}) \\ w_{an} \leftarrow w_{an} + (w_{an} - w_{aj}) \\ \text{and } w_{qn} \leftarrow w_{qn} + (w_{qn} - w_{aj}), \end{array} \right\} \quad (11.24)$$

where  $w_{sn}$ ,  $w_{an}$  and  $w_{qn}$  denote the weights of neurons in the neighborhood around the selected neuron  $j$ .

## Recall

During the recall phase, the state  $s$  is available, and we need to know the best action and the corresponding Q-value at this state. An exhaustive search is needed to identify the neuron  $k$  in the 2-D space having its weights  $(w_{sk}, w_{qk})$  close enough to  $(s_j, Q_j)$  in the Euclidean space. But how do we know  $Q_j$  for the best action? Since it is not known, we set  $Q_j = 1$ , signifying that we look for the best action with a very large Q-value. In other words, we identify neuron  $k$  such that

$$d_k = [ (s_j - w_{sk})^2 + (1 - w_{qk})^2 ]^{1/2} \quad (11.25)$$

is minimum for all  $k$ .

Retrieve the action  $a_k$  and Q-values of the neuron by the following equations:

$$a_k \leftarrow \alpha_k + w_{aj}, \quad (11.26)$$

where  $\alpha_k$  lying in  $[0, 1]$  is variable parameter that decreases with learning steps.

$$Q'_k = w_{qj}. \quad (11.27)$$

## 11.8 Multi-Agent Reinforcement Learning

Performance of a complex task in a multi-agent system depends largely on cooperative and competitive behavior of the agents. One approach to improve the cooperative/ competitive behavior of the agents is to modify the agents' plans in real time based on the environmental feedback caused by its actions. Since the agents need to learn their environment through interaction with their real world, reinforcement learning is most suitable for multi-agent systems.

Two variants of Q-learning algorithm have been addressed in the current literature of multi-agent learning [13]. They are popularly known as **Action Estimation (ACE)** and **Action Group Estimation (AGE)** algorithms. The algorithms estimate varying degree of involvement and coordination effort on the part of the group members. The underlying assumption of these algorithms is that the agents co-work to optimize a group goal.

### Action Estimation Algorithm (ACE)

Let

$S$  be a current environmental state,

$A_i(S)$  be the set of actions that an agent  $a_i$  can perform at state  $S$ ,

$A_i^j \in A_i(S)$  be the  $j$ -th action by agent  $a_i$ ,

$E_i^j(S)$  be the goal relevance of action  $A_i^j$  in state  $S$ .

Then for all actions, whose estimated goal relevance is above a prescribed threshold, the agent computes and announces a bid proportional to its goal relevance. Let  $B_i^j(S)$  be the bid announced by the agent  $a_i$ . Then

$$B_i^j(S) \propto E_i^j(S), \quad (11.28)$$

$$\text{Or, } B_i^j(S) = (\alpha + \beta) E_i^j(S), \quad (11.29)$$

where

$\beta$  is a noise term introduced to prevent convergence to local minima,  
and  $\alpha$  is a small constant risk factor.

The action with the highest bid is selected for execution, and incompatible actions are eliminated from consideration. The above step is repeated until all actions for which bids were submitted are either selected or eliminated. The selected actions together form a group, called *activity context*. Let  $A$  denote the current activity context. Any external payoff received by the system due to execution of actions in the current activity context is equally divided among the executed actions. The objective of this estimate re-assignment is to enable

successful action sequence to increase in estimate over time and to suppress the estimates of ineffective actions. The net estimate update for any selected action is given by

$$E_i^j(S) \leftarrow E_i^j(S) - B_i(S) + \frac{R}{\sum A} \quad (11.30)$$

where  $R$  denotes the received external reward.

The bid values paid out are summed up and re-distributed equally among all the actions  $A_k^l$  corresponding to the immediately previous activity context  $B$  in state  $S'$ .

$$E_k^l(S') \leftarrow E_k^l(S') + \frac{\sum B_i^j(S)}{|B|} \quad (11.31)$$

### Action Group Estimation Algorithm (AGE)

In this algorithm, the applicable actions from all agents in a given environmental state  $S$  are collected. A set of *activity contexts*  $A(S)$  is calculated from these action sets. It is needless to mention that an activity context consists of any set of mutually compatible actions.

$$A(S) = \{A | \forall A_k^l, A_i^j \text{ such that } A_k^l \text{ and } A_i^j \text{ are compatible}\}.$$

Bids are then collected from each agent for all of its actions in an activity context.

$$B_i^j(S, A) = (\alpha + \beta) E_i^j(S, A) \quad (11.32)$$

where  $E_i^j(S, A)$  is agent  $a_i$ 's estimate of goal relevance of action  $A_i^j$  in state  $S$  and the *activity context*  $A$ .

The activity context with the highest sum of bids for the actions contained is selected, and all the actions contained in it are executed by respective agents.

Let  $A$  be the selected activity context. Then for each action  $A_i^j \in A$ , agent  $a_i$  updates its estimates using the following equation:

$$E_i^j(S, A) \leftarrow E_i^j(S, A) - B_i^j(S, A) + \frac{R}{|A|} \quad (11.33)$$

where  $R$  is the received external rewards.

The total bid paid out in the current activity context is distributed among actions executed in the previous activity context B at state  $S'$ .

$$E_k'(S', B) \leftarrow E_k'(S', B) + \frac{\sum_{A_i^j \in A} B_i^j(S, A)}{|B|} \quad (11.34)$$

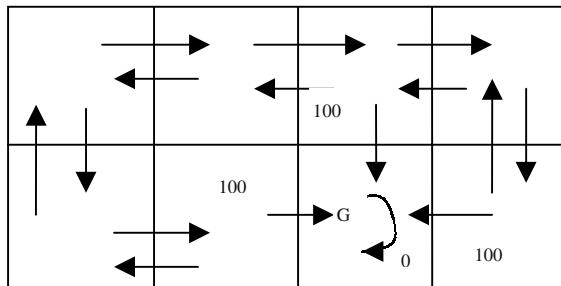
It is indeed important to note that AGE algorithm requires more computational effort, and performance-wise also it supersedes ACE algorithm.

## 11.9 Conclusions

The chapter introduced different types of reinforcement learning. Main emphasis of the chapter is given on Q-learning. The chapter started with a deterministic Q-learning and later extended the ideas for stochastic Q-learning. The principles of Q-learning have been extended to neural realizations for better accuracy and robustness. Both back-propagation algorithm and SOFM can be used for realization of the Q-learning. SOFM, however, has the advantage of on-line tuning of weights following learning. BP can adapt the weights satisfying the Q-learning law, but in offline. Between the multi-agent reinforcement learning algorithms introduced in the chapter, the AGE algorithm is more computation-intensive but performance-wise it supersedes the ACE algorithm [7].

## Exercise

1. Consider the robot's world given in Fig. 11.7 below.



**Fig. 11.7:** A robot's grid world containing the goal position G and immediate rewards. Arrows without any labels are considered to have an immediate reward zero.

- a) Given the goal position G, determine the  $Q(s, a)$  -values for every transition.
  - b) Hence show an optimal policy.
2. a) Assuming that the table of  $Q'$  values is initialized to zero, describe how the  $Q'$ - values are modified as the agent traverses clockwise from the left bottom grid until it reaches the goal.
- b) Repeat the same process as the agent starts moving clockwise from any other grid until it reaches the goal.
3. a) Given the grid-map of a mobile robot (Fig. 11.7). Represent the map by a binary vector.
- b) Define a set of actions such as move forward (MF), turn right (TR), turn left (TL) etc., and hence represent the environment by a matrix.
- c) Suggest a suitable heuristic function to represent that the goal can be reached in one or multiple steps.

**[Hints:** a) We can represent a cell with the robot as 1, and cells without the robot as zero. Thus a grid-map can be represented as a concatenation of rows, where each row is denoted by a bit-string. Rest of the problem can be solved easily with the above representation.]

4. Consider the general form of Q-learning:

$$Q'(s, a) \leftarrow Q'(s, a) + \beta [r(s, a) + \gamma \max_b Q'(s, b) - Q'(s, a)]$$

Show that the above learning law reduces to (11.10) if the table of  $Q'$  matrix is initialized to zero.

[**Hints:** Set  $Q'(s, a) = 0$  for all  $s, a$  in the right hand side of the given expression, and hence obtain expression (11.10).]

5. Consider the problem of obstacle avoidance by a mobile robot. Here, the robot is equipped with ultrasonic transducers that provide range information about the obstacles around the robot.

Let  $x_i(t)$  be the range information obtained from the ultrasonic transducer  $i$ . Suppose we have 8 ultrasonic transducers. Design a heuristic function to determine the immediate reward of the current motion of the robot at time  $t$ .

[**Hints:** Compute:

$$S = \sum_{i=1}^8 x_i(t) - \sum_{i=1}^8 x_i(t-1),$$

which indicates a measure of more/ less occupied by obstacles due to a movement of the robot.

If  $S \geq$  a positive threshold, to be determined experimentally, then the obstacles are away from the robot due to its movement, so  $r = +1$ , else  $r = -1$ .]

6. In path-planning problem of a mobile robot, suppose we need to move the robot towards a fixed goal position without touching any obstacles. Suppose, we want to realize it using Q-learning. Like problem 5, let us assume that the robot has 8 ultrasonic transducers around its cylindrical body to identify the range of obstacles in the direction of flight of the sonar signals. Determine the heuristic function  $r$  for this problem.

[**Hints:** Compute a Manhattan distance between i) the goal position and the next position after one step of movement, and ii) the goal position and the last position. If the difference is greater than a positive threshold, to be determined experimentally, then  $r = -1$ . This means that the robot is moving away from the goal. If the difference is less than the threshold, then  $r = +1$ , i.e. the robot is moving towards the goal.]

7. Consider the problem of realizing Q-learning by using back-propagation algorithm. Given a current state and action, we want to determine the Q-value for the action at the given state. Suppose, the current state and action are applied at the input of the neural net and Q-value is the output of the network. State the weight adaptation policy of the back-propagation neural net.

**Hints:** Let  $s_n$  be a given state, where the agent stochastically selects an action  $a_n$ . With randomly initialized (or pre-trained) weights of the network, we first compute the output of the neural net for known state  $s_n$  and action  $a_n$ . Note that the action  $a_n$  has not been performed yet, we just attempted to compute the response of the network due to  $s_n$  and  $a_n$ . Now, perform  $a_n$  at state  $s_n$ , and get the next state  $s_{n+1}$ . Identify an action that maximizes the Q-value at state  $s_{n+1}$ . Thus compute Q-value using

$$Q(s_n, a_n) \leftarrow r(s_n, a_n) + \gamma \max_b Q(s_{n+1}, b). \quad 0 < \gamma < 1.$$

Compute the error at the output of the neural network by taking the difference of its actual output from the computed Q-value, and back-propagate this error following the steps of back-propagation neural net. A single pass is needed each time, an  $(s_n, a_n)$  pair is given.]

8. In a box-pushing problem two or more mobile robots are employed to push a large box towards a pre-defined goal position. Assume that the robots possess ultrasonic transducers to determine the range of obstacles around them, and are unable to communicate among themselves. How can we employ multi-agent reinforcement learning algorithm for safe transfer of the box to its goal position?

**Hints:** List the possible actions of the individual robots, such as turn  $\pm k(10^\circ)$  clockwise or counterclockwise and apply force in the respective direction of their orientations. The heading direction of the robot should usually be along the direction of the goal position. In case the goal direction is occupied with obstacles, the robots should change their heading direction by minimally rotating clock-wise or counter-clockwise manner to avoid obstacles.

A measure of (heuristic) immediate reward can be obtained by measuring the distance between the current box position and the desired box position. Determine the activity context by identifying the tasks of individual robots that can be performed without conflict. Next explain how you can update  $E_k'(S, B)$ .]

## References

- [1] Bellman, R. E., *Dynamic Programming*, Princeton, NJ: Princeton University Press, 1957.
- [2] Dean, T., Basye, K. and Shewchuk, J., Reinforcement learning for planning and control, In *Machine Learning Methods for Planning*, Minton, S. (Ed.), Morgan-Kaufmann, San Francisco, 1993.
- [3] Lin, L. J., "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, pp. 293-321, 1992.
- [4] Mahadevan, S., "Average reward reinforcement learning: Foundations, algorithms and empirical results," *Machine Learning*, vol. 22, no. 1, pp. 159-195, 1996.
- [5] Mahadevan, S. and Connell, J., "Automatic programming of behavior-based robots using reinforcement learning," *Proc. of the Ninth national Conf. on Artificial Intelligence*, San Francisco, Morgan-Kaufmann, 1991.
- [6] Mitchell, M. M., *Machine Learning*, McGraw-Hill, NY, pp. 384-385, 1997.
- [7] Sen, S. and Weiss, G., *Learning in Multiagent Systems*, In *Multiagent Systems: A Modern Approach to Distributed AI*, Weiss, G. (Ed.), MIT Press, Cambridge, MA, 1999.
- [8] Singh, S. and Sutton, R., "Reinforcement learning with replacing eligibility traces," *Machine Learning*, vol. 22, 1996.
- [9] Sutton, R., "Learning to predict by the methods of temporal difference," *Machine Learning*, vol. 3 pp. 9-44, 1988.
- [10] Tesauro, G., "Temporal difference learning and TD-GAMMON," *Communications of the ACM*, vol. 38, no. 3, pp. 58-68, 1995.
- [11] Watkins, C., *Learning from delayed rewards*, Ph. D. dissertation, King's College, Cambridge, England, 1989.
- [12] Watkins, C. and Dayan, P., "Q-learning," *Machine Learning*, vol. 8, pp. 279-292, 1992.
- [13] Weiss, G., "Learning to coordinate actions in multiagent systems," *Proc. of the 13<sup>th</sup> Int. Conf. on Artificial Intelligence*, pp. 311-316, 1993.

# 12

# Evolutionary Computing Algorithms

*The chapter presents a new kind of classical algorithms that emulates the biological evolutionary process in intelligent search, machine learning and optimization problems. After a brief introduction to this algorithm, the chapter provides a detailed discussion on one such algorithm, called Genetic Algorithm. An analysis of Genetic Algorithm by the well-known Schema theorem and Markov Chains is then presented. The latter part of the chapter is devoted to discuss the possible applications of Genetic Algorithm in machine learning, intelligent search and derivative-free optimization problems. The chapter ends with a discussion on the scope of another evolutionary algorithm, popularly known as Genetic Programming.*

## 12.1 Introduction

During the last two decades, there has been a growing interest in biologically inspired algorithms that rest on the principle of evolution. This class of algorithms is designed based on the natural principle of survival of the fittest, and is usually referred to as evolutionary algorithms.

The main steps of a simple Evolutionary Computing Algorithm are presented below in Procedure Evolutionary-Computation. Here,  $P(t)$  denotes a population

of chromosomes (binary bit strings) at time  $t$ . The procedure initializes a population  $P(t)$  at iteration  $t=0$ . The function: Evaluate  $P(t)$  determines the fitness of the chromosomes by employing a specially constructed fitness measuring function. The *while loop* includes three main steps. First, it increases iteration index by 1. Next it selects a population  $P(t)$  from  $P(t-1)$  based on the results of fitness evaluation. The function: Alter  $P(t)$  evolves  $P(t)$  by some complex non-linear operations. The while loop then re-evaluates  $P(t)$  for the next iteration, and continues evolution until the terminating condition is reached.

**Procedure Evolutionary-Computation**  
**Begin**  
 $t \leftarrow 0;$   
 Initialize  $P(t);$   
 Evaluate  $P(t);$   
**While** (terminating condition not reached) **do**  
**Begin**  
 $t \leftarrow t+1;$   
 Select  $P(t)$  from  $P(t-1);$   
 Alter  $P(t);$   
 Evaluate  $P(t);$   
**End While;**  
**End.**

Depending on the choice of the function: Alter  $P(t)$ , Evolutionary computing algorithms can have different variants. Typical evolutionary computing algorithms include Genetic Algorithms (GAs), Genetic Programming (GP), Particle Swarm Optimization (PSO), Ant Colony Algorithms (ACO), Differential Evolution (DE) algorithm and others. In this chapter, we discuss GAs and GPs in sufficient detail. PSO is introduced in chapter 22. ACO and DE algorithm are outlined in Appendix-B.

The chapter is divided into 10 sections. Section 12.2 provides an introduction to GA and illustrates it with a simple optimization problem. Deterministic explanation of Professor Holland's observation about the algorithm is covered in section 12.3. Stochastic explanation of the algorithm is presented in section 12.4. A convergence analysis of the algorithm using Markov chain is given in section 12.5. Application of GA in optimization problems, machine learning and search are presented in sections 12.6, 12.7 and 12.8 respectively. Principles of Genetic Programming are introduced in section 12.9. Conclusions are listed in section 12.10.

## 12.2 Genetic Algorithm: How Does it Work?

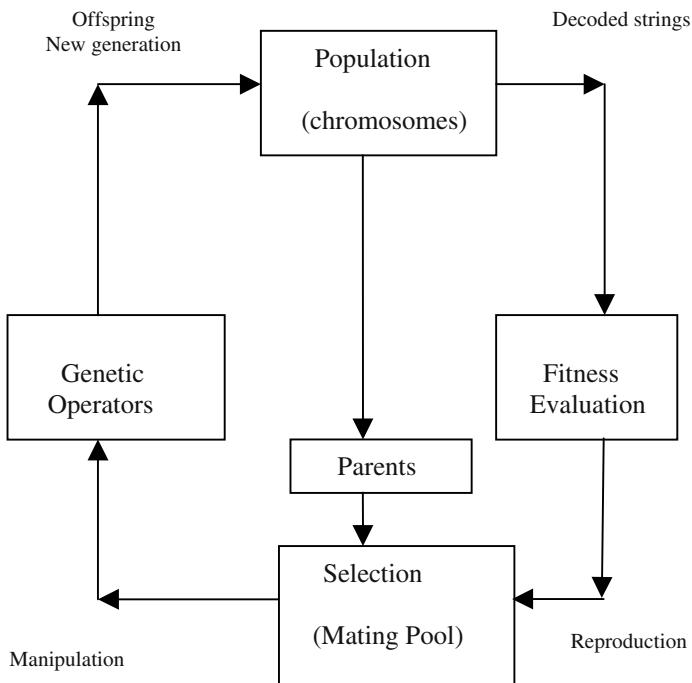
Professor John Holland in 1975 proposed an attractive class of computational models, called Genetic Algorithms (GAs) [1-19], that mimic the biological

evolution process for solving problems in a wide domain. The mechanisms under GA have been analyzed and explained later by Goldberg [10], De Jong [7], Davis [6], Muehlenbein [16], Chakraborti [3-5], Fogel [8], Vose [19] and many others. GA has three major applications, namely, intelligent search, optimization and machine learning.

A GA operates through a simple cycle of stages [9]:

- i) Creation of a “population” of strings,
- ii) Evaluation of each string,
- iii) Selection of best strings and
- iv) Genetic manipulation to create new population of strings.

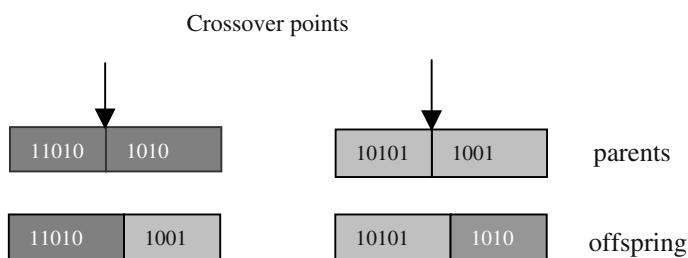
The cycle of a GA is presented below in Fig. 12.1.



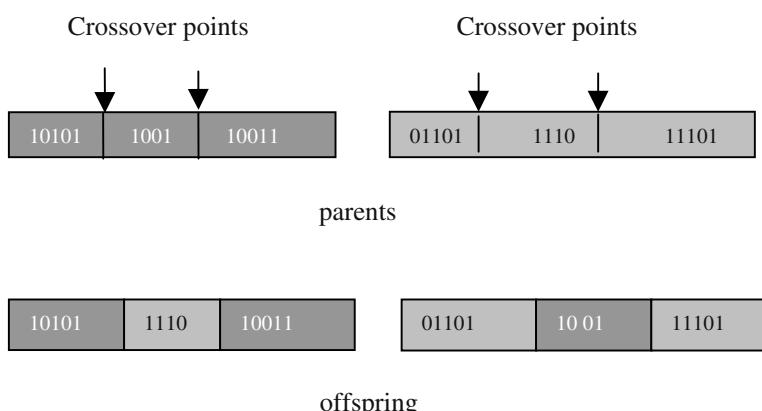
**Fig. 12.1:** The cycle of genetic algorithms.

Each cycle in GA produces a new generation of possible solutions for a given problem. In the first phase, an initial population, describing representatives of the potential solution, is created to initiate the search process. The elements of the population are encoded into bit-strings, called chromosomes. The performance of the strings, often called fitness, is then

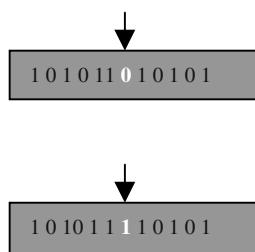
evaluated with the help of some functions, representing the constraints of the problem. Depending on the fitness of the chromosomes, they are selected for a subsequent genetic manipulation process. It should be noted that the **selection** process is mainly responsible for assuring survival of the best-fit individuals. After selection of the population strings is over, the genetic manipulation process consisting of two steps is carried out. In the first step, the **crossover** operation that recombines the bits (genes) of each two selected strings (chromosomes) is executed. Various types of crossover operators are found in the literature. The single point and two points crossover operations are illustrated in Fig. 12.2 and 12.3 respectively. The **crossover points** of any two chromosomes are selected randomly. The second step in the genetic manipulation process is termed **mutation**, where the bits at one or more randomly selected positions of the chromosomes are altered. The mutation process helps to overcome trapping at local maxima. The offspring produced by the genetic manipulation process are the next population to be evaluated.



**Fig 12.2:** A single point crossover after the 3-rd bit position from the L.S.B.



**Fig. 12.3:** Two point crossover: one after the 4<sup>th</sup> and the other after the 8<sup>th</sup> bit positions from the L.S.B.



**Fig. 12.4:** Mutation of a chromosome at the 5<sup>th</sup> bit position.

**Example 12.1:** Consider the problem of determining the solution of a simple algebraic equation:

$$x^2 - 64 = 0,$$

the solution to which is known as  $x = \pm 8$ . Suppose we are interested in the positive solution of the above equation. How can we use GA to handle the problem? In this example, we illustrate the basic steps of GA in solving the above equation.

#### Step 0: Initialization of population

We initialize GA with 10 chromosomes of 5 bit strings. So, population-size (or pop-size) in the present context is 10. The list of 10 chromosomes is given below.

$$\begin{aligned} v_1 &= (00001)_2 = (1)_{10} \\ v_2 &= (00010)_2 = (2)_{10} \\ v_3 &= (00100)_2 = (4)_{10} \\ v_4 &= (01100)_2 = (12)_{10} \\ v_5 &= (10000)_2 = (16)_{10} \\ v_6 &= (10001)_2 = (17)_{10} \\ v_7 &= (11000)_2 = (24)_{10} \\ v_8 &= (10100)_2 = (20)_{10} \\ v_9 &= (11100)_2 = (28)_{10} \\ v_{10} &= (11011)_2 = (27)_{10} \end{aligned}$$

#### Step 1: Evaluation of fitness function

We now evaluate the fitness of the chromosomes by using  $f(x) = |x^2 - 64|$ , where  $x$  is the decimal value of a given chromosome. Naturally, a question

arises: why to select this function? Since we want to find a solution of the equation:  $x^2 - 64 = 0$ , we should select a function that can measure the goodness of the solution. It can indeed be checked that the given function  $f$  will return a value close to zero, when the value of  $x$  is also close to the solution ( $x=8$ ). The results of fitness evaluation for the strings  $v_1$  through  $v_{10}$  are given below.

$$\begin{aligned}f(v_1) &= |1^2 - 64| = 63 \\f(v_2) &= |2^2 - 64| = 60 \\f(v_3) &= |4^2 - 64| = 48 \\f(v_4) &= |12^2 - 64| = 80 \\f(v_5) &= |16^2 - 64| = 192 \\f(v_6) &= |17^2 - 64| = 225 \\f(v_7) &= |24^2 - 64| = 512 \\f(v_8) &= |20^2 - 64| = 336 \\f(v_9) &= |28^2 - 64| = 720 \\f(v_{10}) &= |27^2 - 64| = 665\end{aligned}$$

Now, we determine the probabilities of selection ( $p_i$ ) of chromosome  $i$  by dividing the fitness  $f(v_i)$  of the chromosome by the sum of the fitness of all the chromosomes in the population pool. Thus,

$$p_i = f(v_i) / \sum_{j=1}^{10} f(v_j).$$

Using the above formula, we evaluate the probability of selection of all the 10 chromosomes in the population pool as listed below.

$$\begin{aligned}p_1 &= f(v_1) / \sum f(v_j) = 63 / 2004 = 0.0217 \\p_2 &= f(v_2) / \sum f(v_j) = 60 / 2004 = 0.0206 \\p_3 &= f(v_3) / \sum f(v_j) = 48 / 2004 = 0.0165 \\p_4 &= f(v_4) / \sum f(v_j) = 80 / 2004 = 0.0275 \\p_5 &= f(v_5) / \sum f(v_j) = 192 / 2004 = 0.0661 \\p_6 &= f(v_6) / \sum f(v_j) = 225 / 2004 = 0.0775 \\p_7 &= f(v_7) / \sum f(v_j) = 512 / 2004 = 0.5063 \\p_8 &= f(v_8) / \sum f(v_j) = 336 / 2004 = 0.1158 \\p_9 &= f(v_9) / \sum f(v_j) = 720 / 2004 = 0.2481 \\p_{10} &= f(v_{10}) / \sum f(v_j) = 665 / 2004 = 0.2292\end{aligned}$$

The cumulative probability  $q_j$  for chromosome  $j$  is now computed using the following expression.

$$q_j = \sum_{i=1}^j p_i = (p_1 + p_2 + \dots + p_{j-1}) + p_j = q_{j-1} + p_j, \text{ say.}$$

Thus we evaluate  $q_1$  through  $q_{10}$  as listed below.

$$\begin{aligned}
 q_1 &= p_1 = 0.0217 \\
 q_2 &= q_1 + p_2 = 0.0217 + 0.0206 = 0.0423 \\
 q_3 &= q_2 + p_3 = 0.0423 + 0.0165 = 0.0588 \\
 q_4 &= q_3 + p_4 = 0.0588 + 0.0275 = 0.0863 \\
 q_5 &= q_4 + p_5 = 0.0863 + 0.0661 = 0.2524 \\
 q_6 &= q_5 + p_6 = 0.2524 + 0.0775 = 0.3299 \\
 q_7 &= q_6 + p_7 = 0.3299 + 0.1764 = 0.5063 \\
 q_8 &= q_7 + p_8 = 0.5063 + 0.1158 = 0.6221 \\
 q_9 &= q_8 + p_9 = 0.6221 + 0.2481 = 0.8702 \\
 q_{10} &= q_9 + p_{10} = 0.8702 + 0.2292 = 1.0
 \end{aligned}$$

In order to select a chromosome, we now first generate 10 random numbers in the interval  $[0, 1]$  by spinning a roulette wheel 10 times. The detailed list of 10 random numbers is given below.

$$\begin{aligned}
 &0.5436, 0.7412, 0.8924, 0.2455, 0.6874 \\
 &0.2842, 0.3864, 0.9981, 0.3684, 0.7240
 \end{aligned}$$

The first number in the list is 0.5436. We have to identify the j-th chromosome such that  $q_j > 0.5436$  but  $q_{j-1} < 0.5436$ . We verify that  $q_8 = 0.6221 > 0.5436$ , and  $q_7 = 0.5063 < 0.5436$ . So, chromosome 8 is selected first.

Again, considering the second random number we note that  $q_9 > 0.7412$  but  $q_8 < 0.7412$ ; so, in the second trial the 9-th chromosome is selected. In this manner, we select the chromosomes in the following order.

$$\begin{aligned}
 v'_1 &= v_8 = (10100)_2 \\
 v'_2 &= v_9 = (11100)_2 \\
 v'_3 &= v_{10} = (11011)_2 \\
 v'_4 &= v_{10} = (11011)_2 \\
 v'_5 &= v_9 = (11100)_2 \\
 v'_6 &= v_6 = (10001)_2 \\
 v'_7 &= v_7 = (11000)_2 \\
 v'_8 &= v_{10} = (11011)_2 \\
 v'_9 &= v_7 = (11000)_2 \\
 v'_{10} &= v_9 = (11100)_2
 \end{aligned}$$

It is to be noted that in the Roulette wheel selection,  $v_{10}$  is selected 3-times,  $v_9$  3-times,  $v_8$  once,  $v_7$  2-times and  $v_6$  once.

## Step 2: The Crossover

Now, we select a crossover probability  $p_c = 0.25$  say. To realize the effect of crossover probability, we generate random numbers in  $[0, 1]$  and allow a chromosome to crossover if the random number generated is less than  $p_c$ . Let the random numbers generated for the crossover feasibility testing of the chromosomes in order are

0.2890, 0.1527, 0.2690, 0.1890, 0.2630  
 0.1422, 0.1628, 0.2585, 0.5830, 0.6532.

So, chromosomes  $v_2'$ ,  $v_4'$ ,  $v_6'$  and  $v_7'$  can now be selected for participation in the crossover process. But how should we select the partners of the chromosomes in the crossover game? We can assign the partners arbitrarily. Let us allow  $v_2'$  and  $v_4'$  to form a pair and  $v_6'$  and  $v_7'$  to form a second pair.

Next, the question arises: how to select the cross-site of the chromosomes? To identify the cross-site, we again generate a random number in  $[0, n-1]$  where  $n$  denotes the word-length of the binary string chromosomes. If the random number generated is  $m$ , the  $m$ -th bit is considered as the crossover point for the selected pair. The process is repeated for each pair of selected strings.

As an example, suppose the crossover point of  $v_2'$  and  $v_4'$  is 1, taking L.S.B as the 0<sup>th</sup> bit position. The chromosomes generated after crossover are thus 11111 and 11000. The crossover operation of  $v_2'$  and  $v_4'$  is shown below.

crossover point = 1<sup>st</sup> bit position

$$\left. \begin{array}{l} v_2' = 111|00 \\ v_4' = 110|11 \end{array} \right\} \text{parent}$$

$$\left. \begin{array}{l} v_2'' = 111\ 11 \\ v_4'' = 110\ 00 \end{array} \right\} \text{children}$$

Similarly, suppose the crossover point for the chromosomes  $v_6'$  and  $v_7'$  is the 2<sup>nd</sup> bit position, counted from the L.S.B. Consequently, the resulting offspring are  $v_6'' = 10\ 000$  and  $v_7'' = 11\ 001$ . So, out of the total population pool of 10 chromosomes, namely  $v_1'$  through  $v_{10}'$ , we currently have 6 chromosomes:  $v_1'$ ,  $v_3'$ ,  $v_5'$ ,  $v_8'$ ,  $v_9'$  and  $v_{10}'$ , incapable of participating in the crossover game, and 4 chromosomes:  $v_2''$ ,  $v_4''$ ,  $v_6''$  and  $v_7''$ , which are generated through crossover.

### Step 3: The Mutation

The next operation mutation is performed in a bit-by-bit basis. Let  $p_m = 0.01$ , i.e., we expect on an average 1% bit mutation. There exists altogether  $10 \text{ chromosomes} \times 5 \text{ bits/chromosomes} = 50 \text{ bits}$  in the whole population. 1 % bit mutation thus means  $50 \times 0.01 = 0.5$  bit mutation. Since every bit has an equal chance of mutation, we generate a random number in  $[0, 1]$ , and if the generated number is  $p_m < 0.01$ , we select the chromosome for mutation. Thus for each chromosome, we test the feasibility of the chromosome for mutation.

To identify the bit position of mutation, we generate a random number in  $[0, n-1]$ , where  $n$  is the wordlength of the chromosomes. If the random number generated is  $p$ , the  $p$ -th bit of the selected chromosome will be mutated.

In our proposed application, we found that only  $v_1'$  is selected for mutation at the 2<sup>nd</sup> bit. So, the mutated chromosome  $v_1''' = 10\mathbf{0} 00$ , where the bold 0 is due to mutation. The list of chromosomes in the population pool now is given by

$$\begin{aligned} v_1''' &= (100\ 00)_2 \\ v_2'' &= (111\ 11)_2 \\ v_3' &= (11011)_2 \\ v_4'' &= (110\ 00)_2 \\ v_5' &= (11100)_2 \\ v_6'' &= (100\ 00)_2 \\ v_7'' &= (11001)_2 \\ v_8' &= (11011)_2 \\ v_9' &= (11000)_2 \\ v_{10}' &= (11100)_2 \end{aligned}$$

We thus completed the first pass of the algorithm. We now redefine the above chromosomes by  $v_1$  through  $v_{10}$  and repeat the steps for genetic evolution. The algorithm terminates if the no. of passes crosses a given threshold or the average fitness of chromosomes in each iteration exceeds a threshold.

## 12.3 Deterministic Explanation of Holland's Observation

A **Schema** (or schemata in plural form) / **hyperplane** or **similarity template** [5] is a genetic pattern with fixed values of 1 or 0 at some designated bit positions. For example,  $S = 01?1??1$  is a 7-bit schema with fixed values at 4-bits and don't care values, represented by ?, at the remaining 3 positions. Since 4 positions matter for this schema, we say that the schema contains 4 **genes**.

A basic observation made by Prof. Holland is that “*A schema with an above average fitness tends to increase at an exponential rate until it becomes a significant portion of the population.*”

To explain Holland’s observation in a deterministic manner let us presume the following assumptions [2].

- i) *There are no recombination or alternations to genes.*
- ii) *Initially, a fraction f of the population possesses the schema S and those individuals reproduce at a fixed rate r.*
- iii) *All other individuals lacking schema S reproduce at a rate s < r.*

Thus with an initial population size of  $N$ , after  $t$  generations, we find  $Nf r^t$  individuals possessing schema  $S$  and the population of the rest of the individuals is  $N(1 - f) s^t$ . Therefore, the fraction of the individuals with schema  $S$  is given by

$$\begin{aligned} & (N f r^t) / [N (1-f) s^t + N f r^t] \\ &= f(r/s)^t / [1 + f \{(r/s)^t - 1\}]. \end{aligned} \quad (12.1)$$

For small  $t$  and  $f$ , the above fraction reduces to  $f(r/s)^t$ , which means the population having the schema  $S$  increases exponentially at a rate  $(r/s)$ . A stochastic proof of the above property will be presented shortly, vide a well-known theorem, called the *fundamental theorem of Genetic Algorithm*.

## 12.4 Stochastic Explanation of GA

For presentation of the fundamental theorem of GA, the following terminologies are defined in order.

**Definition 12.1:** *The order of a schema  $H$ , denoted by  $O(H)$ , is the number of fixed positions in the schema. For example, the order of schema  $H = ?001?1?$  is 4, since it contains 4 fixed positions.*

**Definition 12.2:** *The defining length of a schema, denoted by  $d(H)$ , is the difference between the leftmost and rightmost specific (i.e., non-don’t care) string positions.*

For example, the schema  $?1?001$  has a defining length  $d(H) = 4 - 0 = 4$ , while the  $d(H)$  of  $???1??$  is zero.

**Definition 12.3:** The schemas defined over L-bit strings may be geometrically interpreted as **hyperplanes in an L-dimensional hyperspace** (a binary vector space) with each L-bit string representing one corner point in an n-dimensional cube.

### The Fundamental Theorem of Genetic Algorithms (Schema theorem)

Let the population size be N, which contains  $m_H(t)$  samples of schema H at generation t. Among the selection strategies, the most common is the *proportional selection*. In proportional selection, the number of copies of chromosomes selected for mating is proportional to their respective fitness values. Thus, following the principles of proportional selection [17], a string i is selected with probability

$$f_i / \sum_{i=1}^n f_i \quad (12.2)$$

where  $f_i$  is the fitness of string i. Now, the probability that in a single selection, a sample of schema H is chosen is described by

$$\begin{aligned} & m_H(t) / \sum_{i=1}^n f_i \\ & = m_H(t) f_H / \sum_{i=1}^n f_i \quad (12.3) \end{aligned}$$

where  $f_H$  is the average fitness of the  $m_H(t)$  samples of H.

Thus, in a total of N selections with replacement, the expected number of samples of schema H in the next generation is given by

$$\begin{aligned} m_H(t+1) & = N m_H(t) f_H / \sum_{i=1}^n f_i \\ & = m_H(t) f_H / f_{av} \quad (12.4) \end{aligned}$$

$$\text{where } f_{av} = \sum_{i=1}^n f_i / N \quad (12.5)$$

where  $f_{av}$  is the population average fitness in generation  $t$ . The last expression describes that the Genetic algorithm allocates over an increasing number of trials to an above average schema. The effect of crossover and mutation can be incorporated into the analysis by computing the schema survival probabilities of the above average schema. In crossover operation, a schema  $H$  survives if the cross-site falls outside the defining length  $d(H)$ . If  $p_c$  is the probability of crossover and  $L$  is the word-length of the chromosomes, then the disruption probability of schema  $H$  due to crossover is

$$p_c d(H) / (L - 1). \quad (12.6)$$

A schema  $H$ , on the other hand, survives mutation, when none of its fixed positions is mutated. If  $p_m$  is the probability of mutation and  $O(H)$  is the order of schema  $H$ , then the probability that the schema survives is given by

$$(1 - p_m)^{O(H)} = 1 - p_m O(H). \quad (12.7)$$

Therefore, under selection, crossover and mutation, the sample size of schema  $H$  in generation  $(t + 1)$  is given by

$$m_H(t + 1) \geq (m_H(t) f_H / f_{av}) [1 - p_c d(H) / (L - 1) - p_m O(H)]. \quad (12.8)$$

The above theorem is called the **fundamental theorem of GA or the Schema theorem.** It is evident from the Schema theorem that for a given set of values of  $d(H)$ ,  $O(H)$ ,  $L$ ,  $p_c$  and  $p_m$ , the population of schema  $H$  at the subsequent generations increases exponentially when  $f_H > f_{av}$ . This, in fact, directly follows from the difference equation:

$$m_H(t + 1) - m_H(t) \geq (f_H / f_{av} - 1 / K) K m_H(t) \quad (12.9)$$

$$\text{where } K = 1 - p_c d(H) / (L - 1) - p_m O(H). \quad (12.10)$$

$$\Rightarrow \Delta m_H(t) \geq K (f_H / f_{av} - 1 / K) m_H(t). \quad (12.11)$$

Replacing  $\Delta$  by  $(E - 1)$ , where  $E$  is the extended difference operator, we find

$$(E - 1 - K_1) m_H(t) \geq 0 \quad (12.12)$$

$$\text{where } K_1 = K (f_H / f_{av} - 1/K). \quad (12.13)$$

Since  $m_H(t)$  in equation (12.12) is positive,  $E \geq (1 + K_1)$ . Thus, the solution of (12.12) is given by

$$m_H(t) \geq A (1 + K_1)^t \quad (12.14)$$

where  $A$  is a constant. Setting the boundary condition at  $t = 0$ , and substituting the value of  $K_1$  by (14.13) therein, we finally have:

$$m_H(t) \geq m_H(0) (K f_H / f_{av})^t \quad (12.15)$$

Since  $K$  is a positive number, and  $f_H / f_{av} > 1$ ,  $m_H(t)$  grows exponentially with iterations. The process of exponential increase of  $m_H(t)$  continues until some iteration  $r$ , when  $f_H$  approaches  $f_{av}$ . This is all about the proof of the schema theorem.

## 12.5 The Markov Model for Convergence Analysis

To study the convergence of the GA, let us consider an exhaustive set of population states, where ‘state’ means possible members (chromosomes) that evolve at any GA cycle. As an illustration, let us consider 2-bit chromosomes and population size = 2, which means at any GA cycle we select only two chromosomes. Under this circumstance, the possible states that can evolve at any iteration are the members of the set  $S$ , where

$$\begin{aligned} S = \{ &(00, 00), (00, 01), (00, 10), (00, 11), (01, 00), (01, 01), \\ &(01, 10), (01, 11), (10, 00), (10, 01), (10, 10), (10, 11), \\ &(11, 00), (11, 01), (11, 10), (11, 11) \} \end{aligned}$$

For the sake of understanding, let us now consider the population size = 3 and the chromosomes are 2-bit patterns, as presumed earlier. The set  $S$  now takes the following form.

$$\begin{aligned} S = \{ &(00, 00, 00), (00, 00, 01), (00, 00, 10), (00, 00, 11), \\ &(00, 01, 00), (00, 01, 01), (00, 01, 10), (00, 01, 11), \\ &\dots \dots \dots \dots \\ &(11, 11, 00), (11, 11, 01), (11, 11, 10), (11, 11, 11) \}. \end{aligned}$$

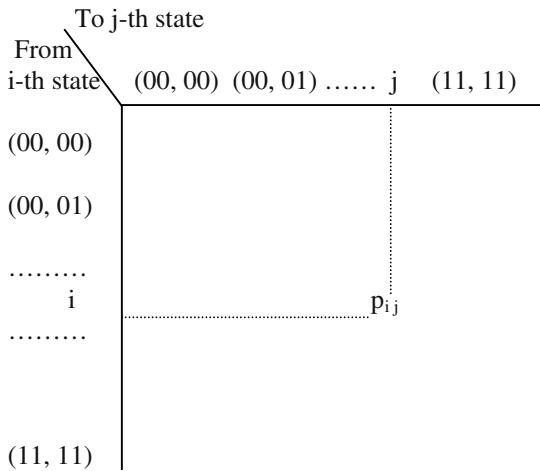
It may be noted that the number of elements of the last set  $S$  is 64. In general, if the chromosomes have the word length of  $m$  bits and the number of chromosomes selected in each GA cycle is  $n$ , then the cardinality of the set  $S$  is  $2^{mn}$ .

The Markov transition probability matrix  $P$  for 2-bit strings of population size 2, thus, will have a dimension of  $(16 \times 16)$ , where the element  $p_{ij}$  of the matrix denotes the probability of transition from  $i$ -th to  $j$ -th state. A clear idea about the states and their transitions can be formed from Fig. 12.5.

It needs mention that since from a given  $i$ -th state, there could be a transition to any 16  $j$ -th states, therefore the row sum of  $P$  matrix must be 1. Formally,

$$\sum_{\forall j} P_{ij} = 1, \quad (12.16)$$

for a given  $i$ .



**Fig. 12.5:** The Markov state-transition matrix  $P$ .

Now, let us assume a row vector  $\pi_t$ , whose  $k$ -th element denotes the probability of occurrence of the  $k$ -th state at a given genetic iteration (cycle)  $t$ ; then  $\pi_{t+1}$ , can be evaluated by

$$\pi_{t+1} = \pi_t \cdot P \quad (12.17)$$

Thus starting with a given initial row vector  $\pi_0$ , one can evaluate the state probability vector after  $n$ -th iteration  $\pi_n$  by

$$\pi_n = \pi_0 \cdot P^n \quad (12.18)$$

where  $P^n$  is evaluated by multiplying  $P$  matrix with itself  $(n-1)$  times.

Identification of a  $P$  matrix for a GA that allows selection, crossover and mutation, undoubtedly, is a complex problem. Goldberg [10], Davis [6], Fogel [8] and Chakraborty [3] have done independent work in this regard. For simplicity of our analysis, let us now consider the GA without mutation.

The behavior of GA without mutation can be of the following three types.

- i) *The GA may converge to one or more absorbing states (i.e., states wherefrom the GA has no transitions to other states).*
- ii) *The GA may have transition to some states, wherefrom it may terminate to one or more absorbing states.*
- iii) *The GA never reaches an absorbing state.*

Taking all the above into account, we thus construct  $P$  as a partitioned matrix of the following form:

$$P = \begin{pmatrix} I & & 0 \\ & \ddots & \\ R & & Q \end{pmatrix}$$

where  $I$  is an identity matrix of dimension  $(a \times a)$  that corresponds to the absorbing states;  $R$  is a  $(t \times a)$  transition sub-matrix describing transition to an absorbing state;  $Q$  is a  $(t \times t)$  transition sub-matrix describing transition to transient states and not to an absorbing state and  $0$  is a null matrix of dimension  $(t \times t)$ . It can be easily shown that  $P^n$  for the above matrix  $P$  can be found to be as follows.

$$P^n = \begin{pmatrix} I & & 0 \\ & N_n R & \\ & & Q^n \end{pmatrix}$$

where the n-step transition matrix  $N_n$  is given by

$$N_n = I + Q + Q^2 + Q^3 + \dots + Q^{n-1} \quad (12.19)$$

As  $n$  approaches infinity,

$$\lim_{n \rightarrow \infty} N_n = (I - Q)^{-1}. \quad (12.20)$$

Consequently, as  $n$  approaches infinity,

$$\lim_{n \rightarrow \infty} P^n = \begin{pmatrix} I & 0 \\ (I - Q)^{-1} R & 0 \end{pmatrix} \quad (12.21)$$

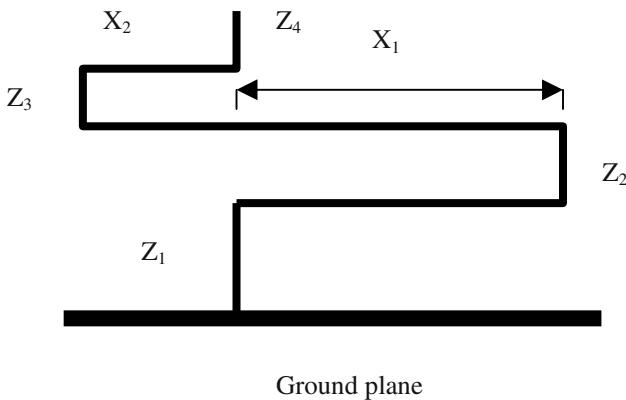
Goodman has shown [8] that the matrix  $(I - Q)^{-1}$  is guaranteed to exist. Thus given an initial probability vector  $\pi_0$ , the chain will have a transition to an absorbing state with probability 1. Further, there exists a non-zero probability that absorbing state will be the globally optimal state [8].

We now explain: why the chain will finally terminate to an absorbing state. Since the first 'a' columns for the matrix  $P^n$ , for  $n \rightarrow \infty$ , are non-zero and the remaining columns are zero, therefore, the chain must have transition to one of the absorbing states. Further, note that the first 'a' columns of the row vector  $\pi_n$  for  $n \rightarrow \infty$  denote the probability of absorption at different states, and the rest of the columns denote that the probability of transition to non-absorbing states is zero. Thus probability of transition to absorbing states is one. Formally,

$$\begin{aligned} & \sum_{i=1}^a \lim_{n \rightarrow \infty} (\pi_n)_i \\ &= \sum_{i=1}^a \lim_{n \rightarrow \infty} (\pi_0 P^n)_i \\ &= \sum_{i=1}^a \lim_{n \rightarrow \infty} (\pi_0 \begin{pmatrix} I \\ (I - Q)^{-1} R \end{pmatrix})_i \\ &= 1. \end{aligned}$$

## 12.6 Application of GA in Optimization Problems

Genetic algorithms have been successfully used in many optimization problems. For instance, the classical travelling salesperson problem, the flowshop and the jobshop scheduling problems and many of the constraint satisfaction problems can be handled with GA. In this section, we illustrate the use of GA in antenna design. There exist various types of antennas. We here illustrate the design of a specialized antenna, called a *monopole loaded with a folded dipole*. The whole antenna has 6 components (Fig. 12.6) , namely  $Z_1$ ,  $Z_2$ ,  $Z_3$ ,  $Z_4$ ,  $X_1$  and  $X_2$ . We want that the electric field vector  $E_\theta$  has to be optimized in one half of the hemisphere. A non-specialist reader can easily understand the meaning of  $E_\theta$  from Fig. 12.7.



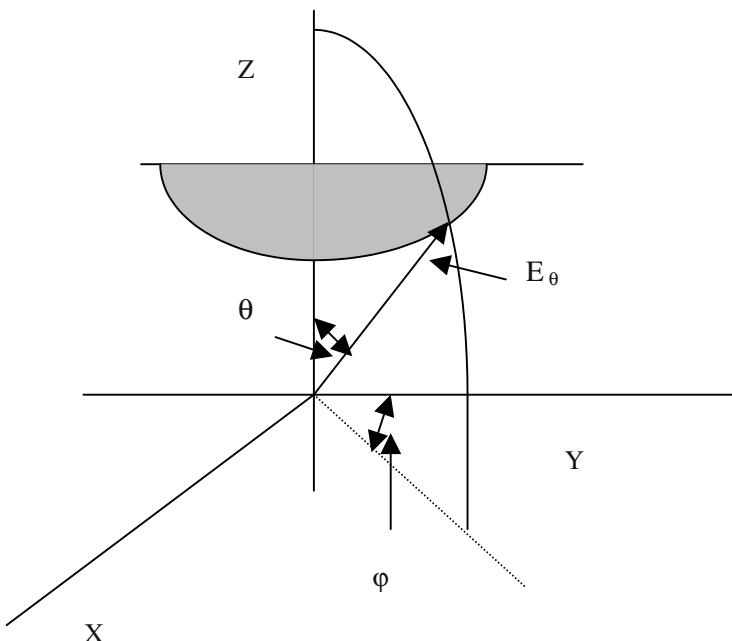
**Fig. 12.6:** A monopole loaded with a folded dipole.

Altshuler and Linden [1] used an NEC<sub>2</sub> package that computes  $E_\theta$  for given arm-lengths of the antenna (Fig. 12.8). The GA program, used by them, on the other hand evolves new chromosomes with 6 fields vide Fig. 12.9. Each field is represented by 5 bits and thus have a possible value in the range of (0 – 31) units. The selection criterion in the present context [1] is given by

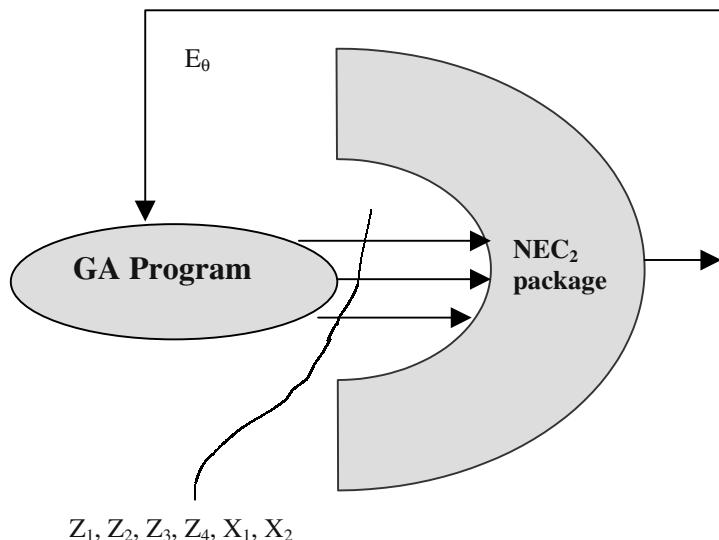
$$\text{Minimize } Z = \sum (E_\theta - E_{\text{desired}})^2. \quad (12.22)$$

$$-180 \leq \forall \varphi \leq 0$$

For realizing this problem, Altshuler et al. considered an initial population of 150 and selected 75 in each iteration. The normal single bit crossover and mutation is used for mating in their scheme. For the best results, the mutation probability is varied between 0 to 0.9.



**Fig. 12.7:** The electric field vector  $E_\theta$  for a given elevation angle  $\theta$  is kept closer to its desired value for the variation of  $\phi$  from (0 – 180) degrees.



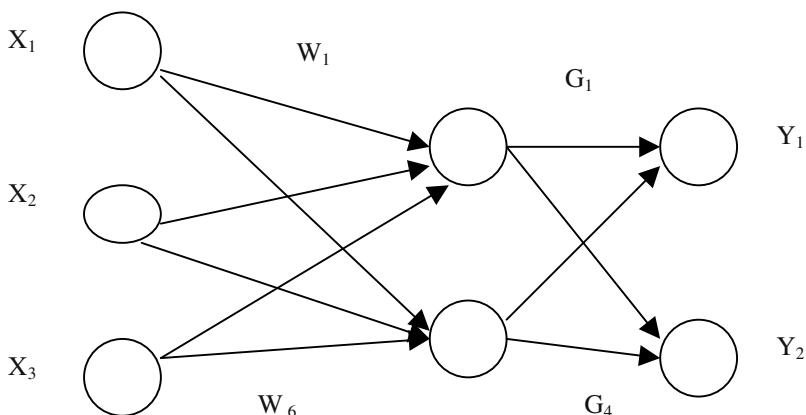
**Fig. 12.8:** The NEC<sub>2</sub> package evaluates  $E_\theta$  from the evolved parameter set:  $Z_1, Z_2, Z_3, Z_4, X_1, X_2$  generated by the GA program.

Z <sub>1</sub>	Z <sub>2</sub>	Z <sub>3</sub>	Z <sub>4</sub>	X <sub>1</sub>	X <sub>2</sub>
----------------	----------------	----------------	----------------	----------------	----------------

**Fig. 12.9:** The field definition of the chromosome, used in antenna design, each field comprising of 5 bits.

## 12.7 Application of GA in Machine Learning

Machine learning is one of the key application fields of Genetic algorithms. A survey of the recent literature [14] on GA reveals that a major area of its applications is concerned with artificial neural nets. It can work with neural nets in three basic ways [15]. First it can adjust the parameters, such as weights and non-linearity of a neural net, when the training instances are supplied. Here it serves the same purpose of the neural learning algorithm. For instance, we can replace the well-known back-propagation algorithm by a GA based scheme. Secondly, GA can be employed to determine the structure of a neural net. Thus when the number of neurons in one or more hidden layer cannot be guessed properly, we may employ GA to solve this problem. Thirdly, GA may be employed to automatically adjust the parameters of a prototype learning equation. This has many useful applications in adaptive control, where the adaptation of the control law is realized with GA.



**Fig.12.10:** Illustrating the use of GA in neural learning.

### 12.7.1 GA as an Alternative to Back-propagation Learning

The back-propagation learning adjusts the weights of a feed-forward neural net by employing the principles of steepest descent learning. One main drawback of this classical algorithm is trapping at local minima. Due to mutation in a GA, it has the characteristics of hill climbing, and thus can overcome the difficulty of trapping at local minima. The principle of using GA for neural learning is presented below.

Here, we considered a three layered neural net with weights (Fig. 12.10). Let the input pattern and the output patterns be  $[ X_1 \ X_2 \ X_3 ]^T$  and  $[ Y_1 \ Y_2 ]^T$  respectively. Let the weights of the first layer and the second layer be  $[ W_1 \ W_2 \ W_3 \ W_4 \ W_5 \ W_6 ]^T$  and  $[ G_1 \ G_2 ]^T$  respectively. Let the non-linearity of each neuron be F. Also, the desired output vector  $[ d_1 \ d_2 ]^T$  is given, where  $d_1$  corresponds to the top output neuron and  $d_2$  corresponds to the bottom output neuron. The selection function in the present context is to minimize Z where

$$Z = [ (d_1 - Y_1)^2 + (d_2 - Y_2)^2 ]^{1/2}. \quad (12.23)$$

The chromosome in the present context comprises of 10 fields, such as  $W_1, W_2, W_3, W_4, W_5, W_6, G_1, G_2, G_3$ , and  $G_4$ . Each field may be represented by a signed real number, expressed in 2's complemented arithmetic. The typical crossover and mutation operators may be used here to evolve the weights. The algorithm terminates when the improvement in Z ceases. Thus for a given input-output training instance, we find a set of neural weights. The idea can be easily extended for training multiple input-output patterns.

### 12.7.2 Adaptation of the Learning Rule/ Control Law by GA

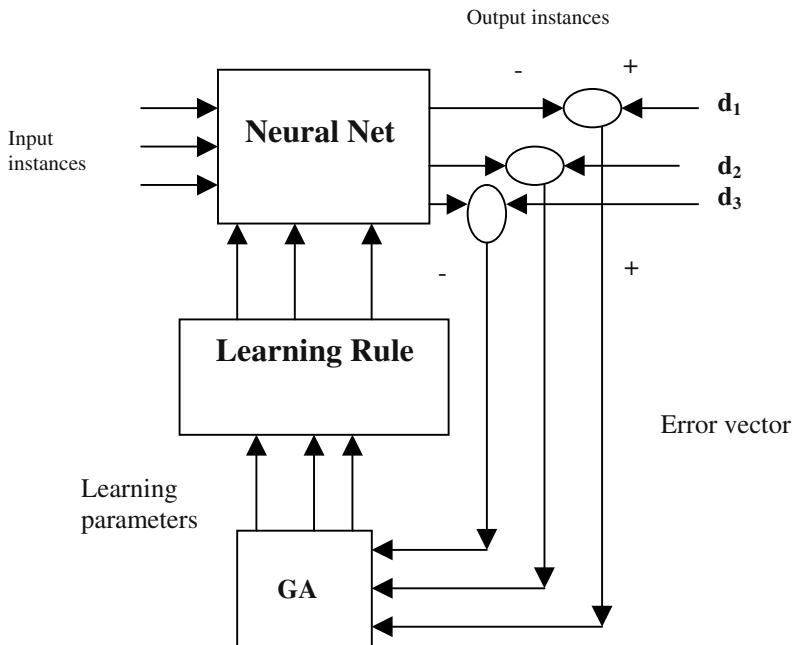
A supervised learning system has to generate a desired output problem instance from a given input problem instance. Let us assume that a multi-layered feed forward neural net is used as the learning agent. Let  $O_k$  be the output node at node  $o_k$  in the output layer,  $I_t$  be the input at the t-th node in the input layer and  $W_{ij}$  be the weight connected from node i to node j. The learning rule in general can be written as

$$\Delta W_{ij} = f(I_t, O_j, W_{ij}) \text{ and} \quad (12.24)$$

$$W_{ij}(t+1) = W_{ij}(t) + \Delta W_{ij}. \quad (12.25)$$

Let  $f(I_t, O_j, W_{ij})$

$$= \sum_t a_t I_t + \sum_j b_j O_j + \sum_t \sum_i c_{ti} I_t O_i + \sum_i \sum_t d_{it} W_{it} I_t + \sum_i \sum_j e_{ij} O_j W_{ij} \quad (12.26)$$



**Fig. 12.11:** Adaptation of the learning rule by using GA.

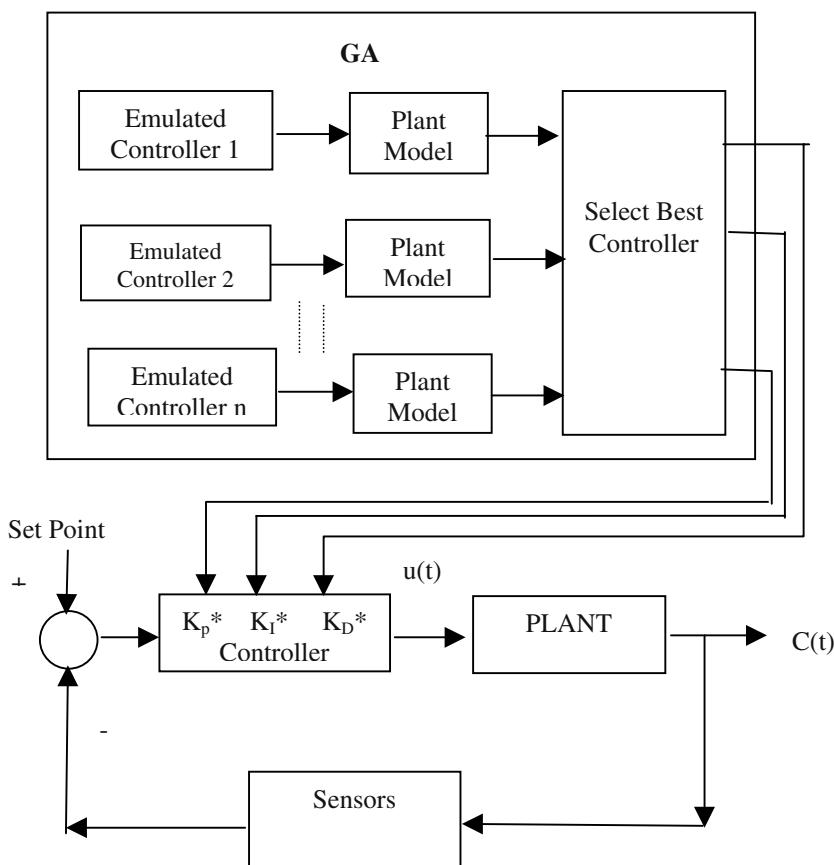
Here, the chromosomes are constructed with the following parameters:  $a_i$ ,  $b_i$ ,  $c_{ti}$ ,  $d_{it}$ ,  $e_{ij}$  as the fields. The fitness of the chromosomes is measured by the square norm of the error signals (target-output) at the output layered nodes. The smaller the norm, the better are the chromosomes. The crossover and mutation operators are comparable with their standard use. After a number of genetic evolutions, GA determines the near optimal values of the parameters. Since the parameters:  $a_i$ ,  $b_i$ ,  $c_{ti}$ ,  $d_{it}$ ,  $e_{ij}$  govern the learning rule, their adaptation constructs new learning rules. Fig. 12.11 describes the adaptation process of the learning rules.

GA can also be used for the adaptation of the control laws in self-tuning adaptive control systems. For instance, consider a self-tuning P-I-D controller. Here, the control law can be expressed as

$$u(t) = K_P e(t) + K_I \int e(t) dt + K_D (de/dt) \quad (12.27)$$

where  $e(t)$  denotes the error (desired output — computed output),  $u(t)$  the control signal and  $K_P$ ,  $K_I$  and  $K_D$  are the proportional, integral and derivative coefficients. The optimization of  $K_P$ ,  $K_I$  and  $K_D$  is required to satisfy some criteria, like minimization of the integral square error:

$$\text{ISE} = \int e^2(t) dt. \quad (12.28)$$

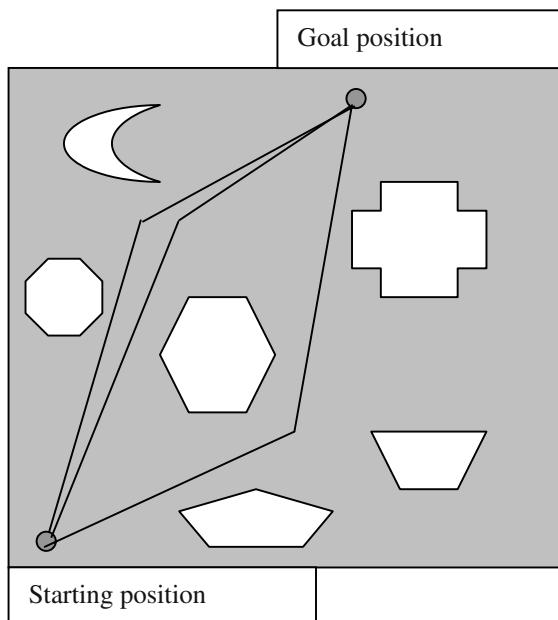


**Fig. 12.12:** The working of GA-based P-I-D tuner.

GA here may be employed to emulate various control laws by randomly selecting different vectors  $[K_p \ K_I \ K_D]^T$ . In other words these vectors represent the population. The ISE, here, has been used as the fitness function. So, GA in each evolution cycle ( iteration ) selects better chromosomes from the population by minimizing the ISE. After each finite interval of time (typically of the order of minutes) the GA submits the control law, described by the optimal vector  $[K_p^* \ K_I^* \ K_D^*]$ . The schematic architecture of the overall system is presented in Fig. 12.12.

## 12.8 Applications of GA in Intelligent Search

Most of the classical AI problems such as n-Queen, the water-jug or the games are search problems, where GA has proved its significance. Other applied search problems include routing in VLSI circuits and navigational planning for robots [17] in a constrained environment. We just introduce the scheme for navigational planning of robots here and will explain the details in chapter 24.



**Fig. 12.13:** Path planning by a robot amidst obstacles.

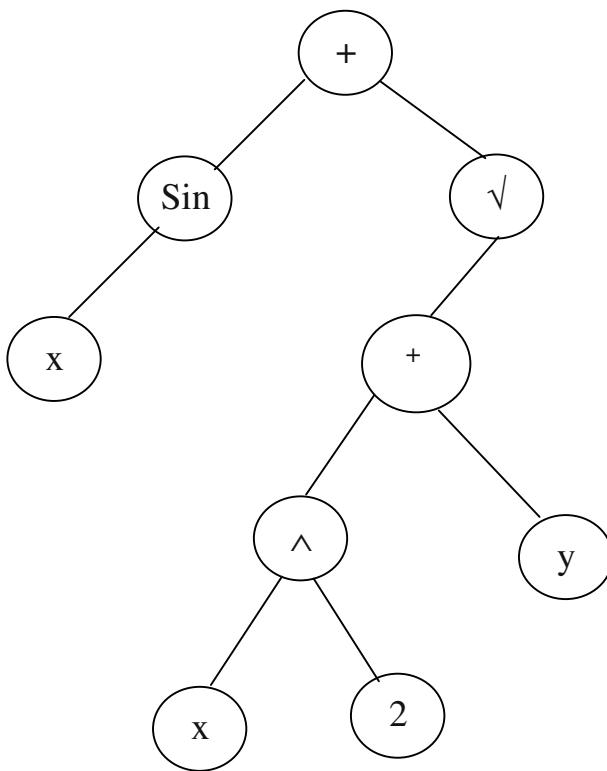
### 12.8.1 Navigational Planning for Robots

The navigational planning for robots is a search problem, where the robot has to plan a path from a given starting position to a goal position. The robot must move without hitting an obstacle in its environment (Fig. 12.13). So, the obstacles in robot's work-space act as constraints to the navigational planning problem. The problem can be solved by GA by choosing an appropriate fitness function that takes into account the distance of the planned path-segments from the obstacles, length of the planned path and the linearity of the paths as practicable.

Michalewicz [17] has formulated the navigational planning problem of robots by GA and simulated it by a new type of crossover and mutation operators. An outline of his scheme is presented in Appendix-A. and the companion CD supplied with the book.

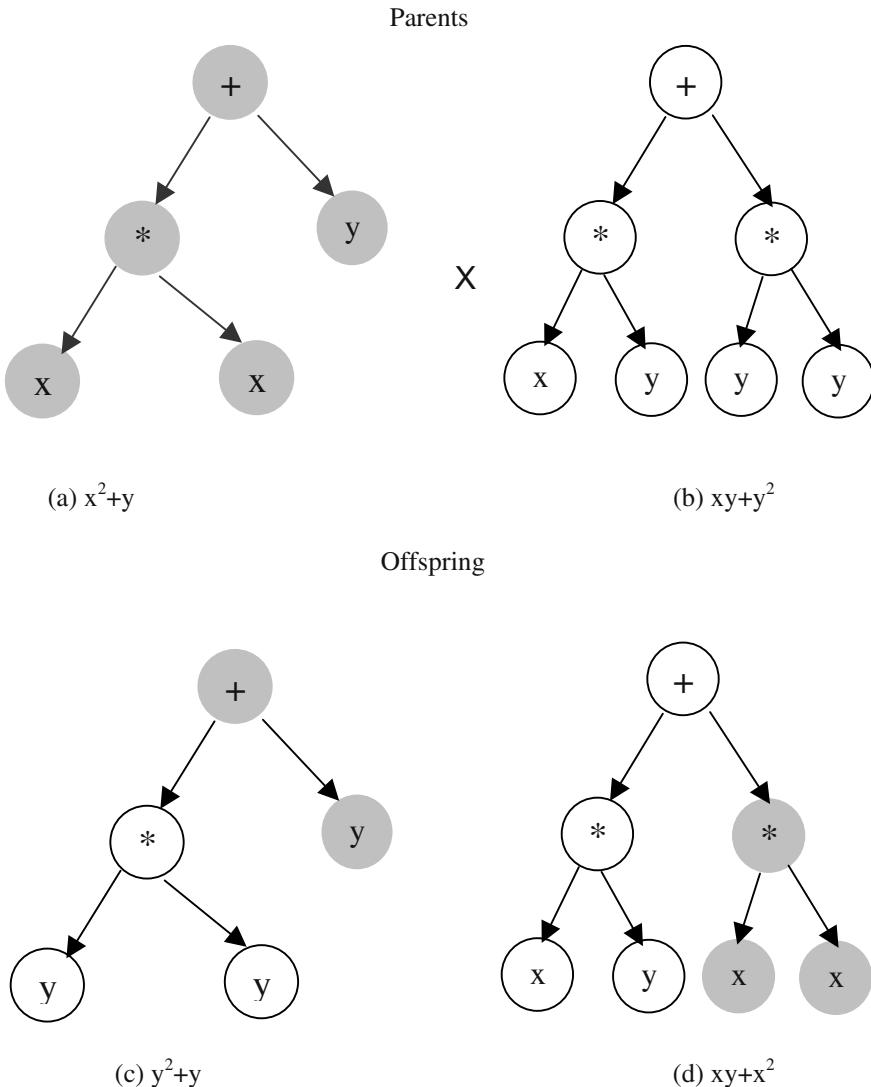
## 12.9 Genetic Programming

Koza [13] applied GA to evolve programs, called Genetic Programming. He represented the program by structure like a parse tree. For instance a function  $f(x) = \text{Sin}(x) + [x * x + y]^{1/2}$  can be represented by a tree, presented in Fig. 12.14.



**Fig. 12.14:** A program tree representing the function  $f(x) = \text{Sin}(x) + \sqrt{x^2 + y}$ .

The crossover operator here is applied on two trees as illustrated in Fig. 12.15. From the parent program for  $x^2 + y$  and  $xy + y^2$  the offspring produced by crossover are  $y^2 + y$  and  $xy + x^2$ .



**Fig. 12.15:** Crossover between 2 genetic programs (a) and (b) yields new programs (c) and (d).

(EQ (DU(MT CS) (NOT CS))  
 (DU(MS NN) (NOT NN)))

where DU (x, y) means Do x until y;

EQ(x, y) returns true when x=y and false otherwise;

MT x means transfer the block from the top of stack to the table;

CS refers to the topmost block in the current stack;

NOT x is true if x is false and vice versa;

MS x means ‘move the block x from the table to the stack top’;

NN refers to the next block required to be placed on the stack top so that the ordering of the letters in UNIVERSAL is maintained.

## **12.10 Conclusions**

GA has proved itself successful in almost every branch of science and engineering. Most of the applications employed GA as a tool for optimization. In fact, when there exist no guidelines to optimize a function of several variables, GA works as a random search and finds an optimal (or at least a near optimal) solution. There exists a massive scope of GA in machine learning, but no significant progress has been observed in this area to date. The next generation intelligent machines are likely to employ GA with neuro-fuzzy models to reason and learn from incomplete data and knowledge bases. Such systems will find immense applications in robotics, knowledge acquisition and image understanding systems.

A common question that is often raised is: can we use GA in ‘discovery problems’? The answer to this is in the affirmative. As GA is a search algorithm, like other search algorithms, it may in association with specialized genetic operators explore some undiscovered search space. Formulation of the problem and selection of the genetic operators, however, play a vital role in this regard.

## Exercise

1. Show the first 3 cycles of genetic evolution for optimizing the function  $y = x^3 - 27$  in the interval  $0 \leq x \leq 12$ . Use set  $P = \{1100, 1010, 1011, 0011\}$  as the initial population.
2. How can you realize crossover probability = 0.7 (say) in the GA program?

[**Hints:** Generate a random number in the interval [0, 1]. If the number generated  $> 0.7$ , then allow crossover between the two selected chromosomes.]

3. List the possible states for population size = 2 and length L of chromosomes = 3. How many states do you obtain? What is the dimension of the Markov state transition matrix P in this context?
4. Suppose in expression (12.8),  $d(H) = L-1$ ,  $p_m \propto (H) \rightarrow 0$ . Under this setting,

$$m_H(t+1) \geq m_H(t) (f_H / f_{av}) (1 - p_c).$$

It is evident from the above inequality that as  $p_c \rightarrow 0$ ,  $m_H(t)$  grows at a faster rate. Can you give a logical justification to this observation? If you feel comfortable to answer this, then try to explain the situation when  $p_c \rightarrow 1$ .

5. Unlike the standard GA, suppose you devise a new type of algorithm, where in each cycle you randomly pick up M chromosomes and select good chromosomes from these by a selection function, and keep copies of the good schemata from the last cycle, the selected population size taking into consideration of the two types = N. Further, assume there is no crossover or mutation in your algorithm. Can you analyze the performance of your algorithm? Can you compare the performance with that of a standard GA?

## References

- [1] Altshuler, E. E. and Linden, D. S., "Wire–Antenna designs using genetic algorithms," *IEEE Antennas and Propagation Magazines*, vol. 39, no. 2, April 1997.
- [2] Bender, E. A., *Mathematical Methods in Artificial Intelligence*, IEEE Computer Society Press, Los Alamitos, pp. 589-593, 1996.
- [3] Chakraborty, U. K., Deb, K. and Chakraborty, M., "Analysis of selection algorithms: a Markov chain approach," *Evolutionary Computation*, vol. 4, no. 2, pp. 133-167, 1996.
- [4] Chakraborty, U. K. and Muehlenbein, H., "Linkage equilibrium and genetic algorithms," *Proc. 4<sup>th</sup> IEEE Int. Conf. on Evolutionary Computation*, Indianapolis, pp. 25-29, 1997.
- [5] Chakraborty, U. K. and Dastidar, D. G., "Using reliability analysis to estimate the number of generations to convergence in genetic algorithm," *Information Processing Letters*, vol. 46, pp. 199-209, 1993.
- [6] Davis, T. E. and Principa, J. C., "A Markov chain framework for the simple Genetic Algorithm," *Evolutionary Computation*, vol. 1, no. 3, pp. 269-288, 1993.
- [7] De Jong, K. A., *An Analysis of Behavior of a Class of Genetic Adaptive Systems*, Doctoral dissertation, University of Michigan, 1975.
- [8] Fogel, D. B., *Evolutionary Computation*, IEEE Press, Piscataway, NJ, 1995.
- [9] Filho, J. L. R. and Treleven, P. C., *Genetic Algorithm Programming Environment*, IEEE Computer Society Press, pp. 28-43, June 1994.
- [10] Goldberg, D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.
- [11] Gupta, B., "Bandwidth enhancement of Microstrip antenna through optimal feed using GA," *Seminar on Seekers and Aerospace Sensors*, Hyderabad, India, 1999.
- [12] Holland, J. H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

- [13] Koza, J. R., *Genetic Programming: on the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.
- [14] Mc Donell, J. R., “Control” In *Handbook of Evolutionary Computation*, Back, T., Fogel, D. B. and Michalewicz, Z. (Eds.), IOP and Oxford University Press, New York, 1998.
- [15] Mitchell, M., *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, MA, 1996.
- [16] Muehlenbein, H. and Chakraborty, U. K., “Gene pool recombination genetic algorithm and the onemax function,” *Journal of Computing and Information Technology*, vol. 5, no. 3, pp. 167-182, 1997.
- [17] Michalewicz, Z, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Berlin, 1992.
- [18] Srinivas, M. and Patnaik, L. M., “Genetic search: Analysis using fitness moments,” *IEEE Trans. on Knowledge and Data Engg.*, vol. 8, no. 1, pp. 120-133, 1996.
- [19] Vose, M. D. and Liepins, G. E., “Punctuated equilibrium in genetic search,” *Complex Systems*, vol. 5, pp. 31-44, 1991.

# 13

# Belief Calculus and Probabilistic Reasoning

*This chapter provides two different techniques for probabilistic reasoning popularly known as Dempster –Shafer theory and Pearl’s evidential model for belief propagation. The former technique is employed to reduce uncertainty in decisions when the relevant information needed to arrive at the decision is obtained from multiple sources with non-uniform levels of authenticity. The latter technique is an extension of classical Bayesian literature. It inputs both causal and evidential information of an event to determine its belief. Pearls’ belief propagation model, to be presented in the chapter, is applied on a causal tree or a graph where nodes denote events and the directed arcs denote cause-effect relationship between each two events. The model has extensive applications in diagnostic systems, where the probabilistic sensory data is fed at the leaves of the causal tree, and the root causes of system failure, which are denoted by non-terminal nodes in the network, are identified through an algorithm for belief propagation.*

## 13.1 Introduction

The last 12 Chapters have been devoted to discussions on three distinctive models of computational intelligence such as fuzzy logic, neural networks and

genetic algorithms. This chapter presented the fourth interesting component of computational intelligence, hereafter referred to as belief calculus. It may be noted that belief calculus does not deal with classical calculus with which we are familiarized, but rather it is concerned with probabilistic models for reasoning under uncertainty. It is called calculus as like classical calculus it deals with small quantities, and a small change in a quantity can change a major decision.

The Chapter begins with an overview on probability theory and Baye's theorem and gradually extends the theorem for constructing an evidential model of reasoning following Judea Pearl's pioneering work. An algorithm for belief propagation on a causal tree has been presented based on Pearl's evidential model of reasoning. Reasoning space for some problems is more complex and a tree may not be sufficient to represent the space. A directed acyclic graph (DAG) therefore is the next choice for representation of the problem space. Consequently, the basic tree-based model for reasoning needs to be extended further to handle the complexity of the problem. The chapter provides a Bayesian model for evidential reasoning on a DAG and an algorithm for reasoning with the model. An example is given to illustrate the computational aspects of the model.

The latter part of the chapter provides a discussion on Dempster-Shafer theory. The theory is centered around an orthogonal summation of beliefs of events. An example has been cited to recognize an object from noisy digital, images of a scene obtained from 2 or more sources.

A sectional classification of the chapter is outlined below. In section 13.2, we provide some text book definitions of probability theory. For most readers this section will be a recapitulation. Section 13.3 presents Pearl's belief propagation algorithm for a causal tree. In section 13.4, we extend the tree-based algorithm for reasoning on a directed acyclic graph. Section 13.5 includes a discussion on Dempster-Shafer theory and its applications. Concluding remarks are listed in section 13.6

## 13.2 Elements of Probability Theory

This provides a few definitions of probability theory, which will be needed to explain Pearl's evidential model of reasoning

**Definition 13.1:** *A priori or unconditional probability of an event A, denoted by  $P(A)$ , represents the probability that the event A has occurred.*

When A denotes a proposition that can only have two values: truth or falsehood,  $P(A)$  represents the probability that proposition A is true [6].

**Definition 13.2:** A **random variable**  $X$  has a domain of possible values,  $\{x_1, x_2, \dots, x_n\}$ , that the variable can assume. The sum of the probabilities of  $x = x_1, x = x_2, \dots, x = x_n$  is always unity. In formal notation,

$$\sum_{x \in \{x_1, x_2, \dots, x_n\}} P(X) = 1 \quad (13.1)$$

**Example 13.1:** Let  $X$  be a random variable that denotes the body-temperature of a patient. If we consider  $X$  to be an integer variable, then  $X$  can take any value in set  $T = \{96, 97, 98, 99, \dots, 106\}$  degree Fahrenheit. As the measurement is inaccurate due to a noisy transducer, we consider  $X$  to be a random variable in  $T$ , satisfying

$$\sum_{x \in T} P(X) = 1,$$

where  $P(X)$  for more than one  $X$  is non-zero. However, if the measurement is precise, then only for one  $X$  we have  $P(X) = 1$ , and for the remaining  $X$  in  $T$ ,  $P(X) = 0$ , but whatever be the case, condition (13.1) is always valid.

**Definition 13.3:** The **conditional probability** of  $A$ , given the prior occurrence of  $B$ , denoted by  $P(A|B)$  is defined as a ratio of the probability of the joint occurrence of  $A$  and  $B$  to the probability of  $B$ , i.e.,

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \quad (13.2)$$

**Example 13.2:** Suppose  $A$  and  $B$  be two binary variables. Thus  $A$  and  $B$  jointly can occur by 4 ways: i)  $A$  and  $\neg B$ , ii)  $\neg A$  and  $B$ , iii)  $\neg A$  and  $\neg B$  and iv ) $A$  and  $B$ . Suppose,

$$P(A \cap \neg B) = 0.05$$

$$P(\neg A \cap B) = 0.55$$

$$P(\neg A \cap \neg B) = 0.03$$

$$P(A \cap B) = 0.37$$

The joint probabilities given above are entered into a 4 – cell board like the one shown in Fig. 13.1.

	$P(A)$	$P(\neg A)$
$P(B)$	0.37	0.55
$P(\neg B)$	0.05	0.03

**Fig 13.1:** Structured representation of the probability of  $A \cap \neg B$ ,  $\neg A \cap B$ ,  $\neg A \cap \neg B$  and  $A \cap B$ .

It is clear from Fig 13.1 that the sum of the joint probabilities must be one. Suppose, event B has already occurred and event A is now occurring. Then

$$\begin{aligned}
 P(A / B) &= \frac{P(A \cap B)}{P(B)} \\
 &= \frac{0.37}{0.37 + 0.35} \quad (\text{See Fig 13.1}) \\
 &= \frac{37}{72}
 \end{aligned}$$

### 13.2.1 Bayes' Law on Conditional Probability

Suppose we are given  $P(A)$  and  $P(B | A)$ . Then

$$P(A \cap B) = P(B | A) \cdot P(A). \quad (13.3)$$

Now, suppose we know  $P(B)$  and  $P(A | B)$  then we can write:

$$P(A \cap B) = P(A | B) \cdot P(B). \quad (13.4)$$

Combining (13.3) and (13.4) we have

$$P(B | A) \cdot P(A) = P(A | B) \cdot P(B) \quad (13.5)$$

$$\text{or, } P(B | A) = \frac{P(A | B) \cdot P(B)}{P(A)} \quad (13.6)$$

This is well known as **Bayes' law on conditional probability.**

Suppose, A has n partitions  $A_1, A_2, \dots, A_n$ , such that  $A = \cup A_i, 1 \leq i \leq n$  and B had m partitions  $B_1, B_2, \dots, B_m$ , such that  $B = \cup B_j, 1 \leq j \leq m$ . Also suppose that we are interested to compute  $P(B_j | A_i)$ .

Then by Baye's law,

$$P(B_j | A_i) = \frac{P(A_i | B_j) \cdot P(B_j)}{\sum_{k=1}^m P(A_i | B_k) \cdot P(B_k)} \quad (13.7)$$

The above result is an extension of Bayes' law on conditional probability.

**Example 13.3:** Suppose, probability of getting infected with chicken pox in a locality is 1/ 50,000. Also given the probability of a patient having rash, when he is infected with chicken pox, is 0.3. if the probability of a patient having rash is 0.2, how can we determine the probability of patient being infected with chicken pox when he has rash on his skin?

Let

R denote rash, and

CP denote Chicken pox.

Then by Bayes' law (13.7) we can write:

$$P(CP | R) = \frac{P(R | CP) \cdot P(CP)}{P(R)}$$

Here,  $P(R | CP) = 0.3$ ,  $P(CP) = 1/ 50,000$  and  $P(R) = 0.2$ . Then by (13.5) we compute:

$$P(CP | R) = \frac{0.3 \times (1/50,000)}{0.2}$$

$$= 3 / 100,000$$

$$= 3 \times 10^{-5}.$$

**Example 13.4:** This example illustrates the application of expression (13.7) in a medical diagnosis problem. The problem is outlined below.

Suppose, it is given that rash (R), fever (F) and high body pain (HBP) be the possible symptoms of two diseases namely Chicken pox (CP) and Herpes (H). Given the following probabilities:

$$P(R | CP) = 0.2, P(F | CP) = 0.4,$$

$$P(HBP | CP) = 0.6, P(R | H) = 0.3,$$

$$P(F | H) = 0.3, P(HBP | H) = 0.7,$$

$$P(CP) = 0.02 \text{ and } P(H) = 0.01.$$

We need to compute  $P(CP | R)$  and  $P(H | R)$ . How should we proceed? By expression (13.7) we can write:

$$\begin{aligned} P(CP | R) &= \frac{P(R | CP) \cdot P(CP)}{P(R | CP) \cdot P(CP) + P(R | H) \cdot P(H)} \\ &= \frac{0.2 \times 0.02}{(0.2 \times 0.02) + (0.3 \times 0.01)} \\ &= \frac{0.004}{0.007} \\ &= 4 / 7. \end{aligned}$$

Similarly we can compute:

$$\begin{aligned} P(H | R) &= \frac{P(R | H) \cdot P(H)}{P(R | H) \cdot P(H) + P(R | CP) \cdot P(CP)} \\ &= \frac{0.3 \times 0.01}{(0.3 \times 0.01) + (0.2 \times 0.02)} \end{aligned}$$

$$= \frac{0.003}{0.007} \\ = 3 / 7.$$

It is interesting to note that for computing  $P(H | R)$  and  $P(CP | R)$  we do not require  $P(F | H)$ ,  $P(HBP | H)$ ,  $P(F | CP)$  and  $P(HBP | CP)$ . This is because of our assumption that F, R and HBP are independent of each other. The above four probabilities, however, are needed for computing  $P(H | F)$ ,  $P(H | HBP)$ ,  $P(CP | F)$  and  $P(CP | HBP)$ .

On occasions we will also have to use a more general version of conditionalized probability in presence of a background evidence E. For instance, suppose we are given with  $P(Y | X, E)$  and we have to simplify it.

Here,

$$P(Y | X, E) \\ = \frac{P(Y \cap (X \cap E))}{P(X \cap E)} \quad (13.8)$$

$$= \frac{P(X \cap (Y \cap E))}{P(X \cap E)} \quad (13.9)$$

$$= \frac{P(X | Y, E) \cdot P(Y \cap E)}{P(X \cap E)} \quad (13.10)$$

$$= \frac{P(X | Y, E) \cdot P(Y | E) \cdot P(E)}{P(X | E) \cdot P(E)} \quad (13.11)$$

$$= \frac{P(X | Y, E) \cdot P(Y | E)}{P(X | E)} \quad (13.12)$$

Thus,

$$P(Y | X, E) = \frac{P(X | Y, E) \cdot P(Y | E)}{P(X | E)} \quad (13.13)$$

is a useful result for applications in probabilistic reasoning in presence of an evidence E. Expression (13.13) is another version of Baye's law. The fundamental Bayes' law, given by (13.6) can be easily obtained from (13.13) when X & Y are independent of E. In that case  $P(X | E) = P(X)$ ,  $P(Y | E) = P(Y)$  and  $P(X | Y, E) = P(X | Y)$ . Consequently, expression (13.6) directly follows from expression (13.6) with the above substitutions.

**Example 13.5:** Show that

$$P(A, B | E) = P(A | B, E) \cdot P(B | E)$$

where B is presumed to be dependent on a background evidence E [6]. Starting with definition (13.2), we can write:

$$\begin{aligned} & P(A, B | E) \\ &= P(A \cap B | E) \\ &= \frac{P(A \cap B \cap E)}{P(E)} \\ &= \frac{P(A | B, E) \cdot P(B \cap E)}{P(E)} \\ &= \frac{P(A | B, E) \cdot P(B | E) \cdot P(E)}{P(E)} \\ &= P(A | B, E) \cdot P(B | E) \quad (\text{proved}). \end{aligned}$$

**Example 13.6:** Show that the statement  $P(A, B | C) = P(A | C) P(B | C)$  is equivalent to saying

$$P(A | B, C) = P(A | C), \text{ and}$$

$$P(B | A, C) = P(B | C).$$

We use the result of example 13.5 to prove the above statement. Thus,

$$\begin{aligned} P(A, B | C) &= P(A | B, C) \cdot P(B | C) \\ &= P(A | C) \cdot P(B | C) \quad (\text{given}) \end{aligned}$$

So,  $P(A | B, C) = P(A | C)$ , which means A and B are independent.

Further,  $P(A, B | C)$

$$\begin{aligned} &= P(B, A | C) \\ &= P(B | A, C) \cdot P(A | C) \\ &= P(B | C) \cdot P(A | C) \quad (\text{given}) \end{aligned}$$

$P(B | A, C) = P(B | C)$ ; i.e., B is independent of A.

Hence, the equivalent sayings follow.

One important aspect of Bayes' conditional probability is that the sum of the probability of an event A and  $\neg A$ , assuming the prior occurrence of B in both the cases is 1. Symbolically,

$$P(A | B) + P(\neg A | B) = 1 \quad (13.14)$$

since  $P(A | B) = \frac{P(B | A) \cdot P(A)}{P(B)}$ , and

$$P(\neg A | B) = \frac{P(B | \neg A) \cdot P(\neg A)}{P(B)}$$

substituting the last two expressions in (13.14) we find

$$P(B | A) \cdot P(A) + P(B | \neg A) \cdot P(\neg A) = P(B) \quad (13.15)$$

In many applications, we denote  $P(B)$  by a normalizing constant ( $1/\infty$ ), the value of which is chosen to satisfy condition (13.14) the process of selecting a constant  $\infty$  to satisfy conditions (13.14) is referred to as *normalization* process. Thus in general we write:

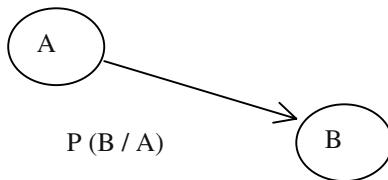
$$P(A | B) = \alpha P(B | A) \cdot P(A) \quad (13.16)$$

and  $P(\neg A | B) = \alpha P(B | \neg A) \cdot P(\neg A)$ . (13.17)

### 13.3 Belief Propagation on a Causal Tree

A Bayesian belief network [2-3] is a directed acyclic graph or tree, where the nodes denote the events and the arcs denote the cause-effect relationship

between the parent and the child nodes. Each node, here, may assume a number of possible values. For instance, a node A may have  $n$  number of possible values, denoted by  $A_1, A_2, \dots, A_n$ . For any two nodes, A and B, when there exists a dependence  $A \rightarrow B$ , we assign a conditional probability matrix  $[P(B/A)]$  to the directed arc from node A to B. The element at the  $j^{\text{th}}$  row and  $i^{\text{th}}$  column of  $P(B/A)$ , denoted by  $P(B_j / A_i)$ , represents the conditional probability of  $B_j$  assuming the prior occurrence of  $A_i$ . This is described in Fig. 13.2.



**Fig. 13.2:** Assigning a conditional probability matrix in the directed arc connected from A to B.

Given the probability distribution of A, denoted by  $[P(A_1) \ P(A_2) \ \dots \ P(A_n)]$ , we can compute the probability distribution of event B by using the following expression:

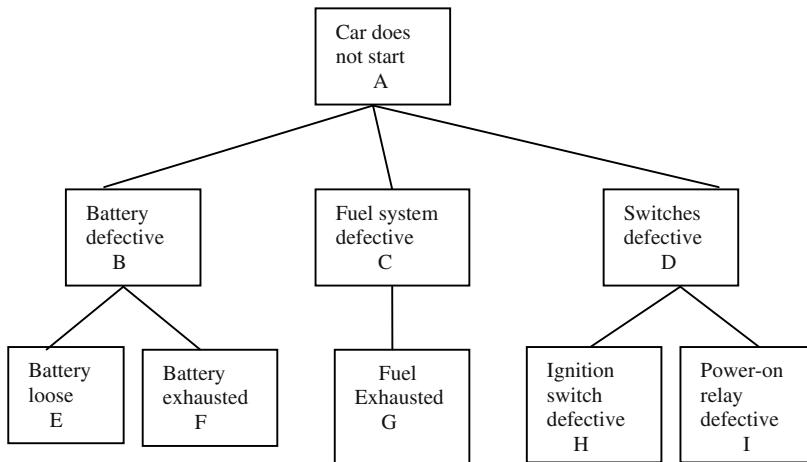
$$\begin{aligned} P(B) &= [P(B_1) \ P(B_2) \ \dots \ P(B_m)]_{1 \times m} \\ &= [P(A_1) \ P(A_2) \ \dots \ P(A_n)]_{1 \times n} \cdot [P(B/A)]_{n \times m} \end{aligned} \quad (13.18)$$

$$= [P(A)]_{1 \times n} \cdot [P(B/A)]_{n \times m} \quad (13.19)$$

We now illustrate the computation of  $P(B)$  with an example.

**Example 13.7:** Consider a Bayesian belief tree describing the possible causes of a defective car.

Here, each event in the tree (Fig. 13.3) can have two possible values: true or false. Thus the matrices associated with the arcs will have dimensions  $(2 \times 2)$ . Now, given  $P(A) = [P(A = \text{true}) \ P(A = \text{false})]$ , we can easily compute  $P(B)$ ,  $P(C)$ ,  $P(D)$ ,  $P(E)$ , ...,  $P(I)$  provided we know the transition probability matrices connected with the links. As an illustrative example, we compute  $P(B)$  with  $P(B/A)$  and  $P(A)$ .



**Fig. 13.3:** A diagnostic tree for a car.

$$\text{Let } P(A) = [P(A = \text{true}) \quad P(A = \text{false})]$$

$$= [0.7 \quad 0.3]$$

	$A_i$	$B_j \rightarrow$	
		$B = \text{true}$	$B = \text{false}$
$P(B/A) =$	$A = \text{true}$	0.8	0.2
	$A = \text{false}$	0.4	0.6

$$\text{So, } P(B) = P(A). \quad P(B / A) = [0.68 \quad 0.32]$$

One interesting property of Bayesian network is that we can compute the probability of the joint occurrence easily with the help of the topology. For instance, the probability of joint occurrence of A, B, C, D, E, F, G, H, I (see Fig. 13.3) is given by

$$\begin{aligned}
 & P(A, B, C, D, E, F, G, H, I) \\
 & = P(A / B).P(A / C).P(A / D).P(B / E, F).P(C / G).P(D / H, I) \quad (13.20)
 \end{aligned}$$

Further, if E and F are independent, and H and I are independent, the above result reduces to

$$P(A/B).P(A/C).P(A/D).P(B/E).P(B/F).P(C/G).P(D/H).P(D/I).$$

Thus, given A, B, C,...,H all true except I, we would substitute the conditional probabilities for  $P(B = \text{true} / A = \text{true})$ ,  $P(A = \text{true} / C = \text{true})$ ....and finally  $P(D = \text{true} / I = \text{false})$  in the last expression to compute  $P(A = \text{true}, B = \text{true}, \dots, H = \text{true}, I = \text{false})$ .

Judea Pearl [2-3] proposed a scheme for propagating beliefs of evidence in a Bayesian network. We shall first demonstrate his scheme with a Bayesian tree like that in Fig. 13.3. It may, however, be noted that like the tree of Fig. 13.3, each variable, say A,B,..., need not have only two possible values. For example, if a node in a tree denotes German Measles (GM), it could have three possible values like severe-GM, little-GM, moderate-GM.

In Pearl's scheme for evidential reasoning, he considered both the causal effect and the diagnostic effect to compute the **belief function** at a given node in the Bayesian belief tree. For computing belief at a node, say V, he partitioned the tree into two parts: i) the subtree rooted at V and ii) the rest of the tree. Let us denote the subset of the evidence, residing at the subtree of V by  $e_v^-$  and the subset of the evidence from the rest of the tree by  $e_v^+$ . We denote the belief function of the node V by  $\text{Bel}(V)$ , where it is defined as

$$\begin{aligned} \text{Bel}(V) &= P(V/e_v^+, e_v^-) \\ &= P(e_v^-/V).P(V/e_v^+)/\alpha \end{aligned} \quad (13.21)$$

$$= \lambda(V) \prod(V)/\alpha \quad (13.22)$$

$$\left. \begin{aligned} \text{where, } \lambda(V) &= P(e_v^-/V), \\ \prod(V) &= P(V/e_v^+), \end{aligned} \right\} \quad (13.23)$$

and  $\alpha$  is a normalizing constant, determined by

$$\alpha = \sum_{v \in \{\text{true, false}\}} P(e_v^-/V). P(V/e_v^+) \quad (13.24)$$

It seems from the last expression that v could assume only two values: true and false. It is just an illustrative notation. In fact, v can have a number of possible values.

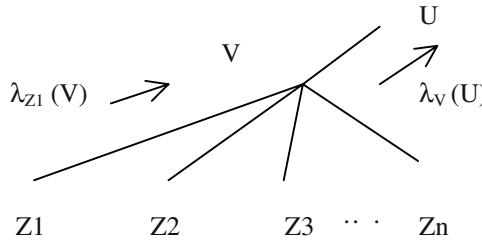
Let node V have n offspring, vide Fig. 13.4. For computing  $\lambda(V)$ , we divide  $e_v^-$  into n disjoint subsets  $e_{Z_i}$ ,  $1 \leq i \leq n$ , where  $Z_i$  is a child of V.

So,  $\lambda(V) = P(e_v^- / V)$ .

$$= P(e_{Z1}^-, e_{Z2}^-, \dots, e_{Zn}^- / V) \quad (13.25)$$

$$= P(e_{Z1}^- / V) \cdot P(e_{Z2}^- / V) \dots P(e_{Zn}^- / V). \quad (13.26)$$

$$= \prod_{i=1}^n \lambda_{Zi}(V) \quad (13.27)$$



**Fig. 13.4:** Propagation of  $\lambda$ s from children to the parent in an illustrative tree.

We now compute  $\Pi(V)$  using the message  $\Pi_V(U) = P(U | e_v^+)$  from the parent  $U$  of  $V$ .

$$\Pi(V) = P(U | e_v^+)$$

$$= \sum_{u \in \{\text{true, false}\}} P(V | e_v^+, U = u) \cdot P(U = u | e_v^+) \quad (13.28)$$

$$= \sum_{u \in \{\text{true, false}\}} P(V | U = u) \cdot P(U = u | e_v^+) \quad (13.29)$$

$$= \sum_{u \in \{\text{true, false}\}} P(V | U = u) \cdot \Pi_v(U = u) \quad (13.30)$$

$$= [P(V|U)]^{T_{2 \times 2}} \times [\Pi_v(0) \quad \Pi_v(1)]^{T_{2 \times 1}} \quad (13.31)$$

We now compute the messages that node  $V$  sends to its parents  $U$  and each of its children  $Z_1, Z_2, \dots, Z_n$  to update their values. Each of these two messages is a conditional probability, given that the condition holds and the probability given that it does not.

Now, the message from  $V$  to parent  $U$ , denoted by  $\lambda_v(U)$ , is computed as

$$\begin{aligned} \lambda_v(U) &= \sum_{v \in \{\text{true, false}\}} P(e_v^- | U, V = v) P(V = v | U) \\ &= \sum_{v \in \{\text{true, false}\}} P(e_v^- | V = v) P(V = v | U) \end{aligned} \quad (13.32)$$

$$= \sum_{v \in \{\text{true}, \text{false}\}} P(V = v | U) \lambda(V = v) \quad (13.33)$$

$$= [P(V | U)]_{2 \times 2} \times [\lambda(0) \quad \lambda(1)]^T_{2 \times 1} \quad (13.34)$$

Lastly, the message from  $V$  to its child  $Z_j$  is given by

$$\begin{aligned} \Pi_{Z_j}(V) \\ = P(V | e_{Z_i}^+) \end{aligned} \quad (13.35)$$

$$= P(V | e_v^+, e_{Z_1}^-, e_{Z_2}^-, \dots, e_{Z_{i-1}}^-, e_{Z_{i+1}}^-, \dots, e_{Z_n}^-) \quad (13.36)$$

$$= \beta \prod_{j \neq i} P(e_{Z_j}^- | V, e_v^+) P(V | e_v^+) \quad (13.37)$$

$$= \beta \prod_{j \neq i} P(e_{Z_j}^- | V) P(V | e_v^+) \quad (13.38)$$

$$= \beta (\prod_{j \neq i} \lambda_{Z_j}(V)) \Pi(V) \quad (13.39)$$

$$= \beta (\lambda(V) / \lambda_{Z_j}(V)) \Pi(V) \quad (13.40)$$

$$= \beta \text{Bel}(V) / \lambda_{Z_j}(V) \quad (13.41)$$

where  $\beta$  is a normalizing constant computed similarly as  $\alpha$ .

The belief updating process at a given node  $B$  (in Fig. 13.3) has been illustrated based on the above expressions for computing the  $\lambda$  and  $\Pi$  messages. We here assumed that at each node and link of the tree (Fig. 13.3) we have one processor [7]. We call these node and link processor respectively. The functions of the node and the link processors are described in Fig. 13.5.

The main steps [8], [12] of the belief-propagation algorithm of Pearl are outlined below.

1. During initialization, we set all  $\lambda$  and  $\Pi$  messages to 1 and set  $\Pi_B(A)$  messages from root to the prior probability  $[P(A_1) P(A_2), \dots, P(A_m)]^T$  and define the conditional probability matrices. Then estimate the prior probabilities at all nodes, starting from the children of the root by taking the product of transpose of the conditional probability matrix at the link and the prior probability vector of the parent. Repeat this for all nodes up to the leaves.
2. Generally the variables at the leaves of the tree are instantiated. Suppose, the variable  $E = E_2$  is instantiated. In that case, we set [7]

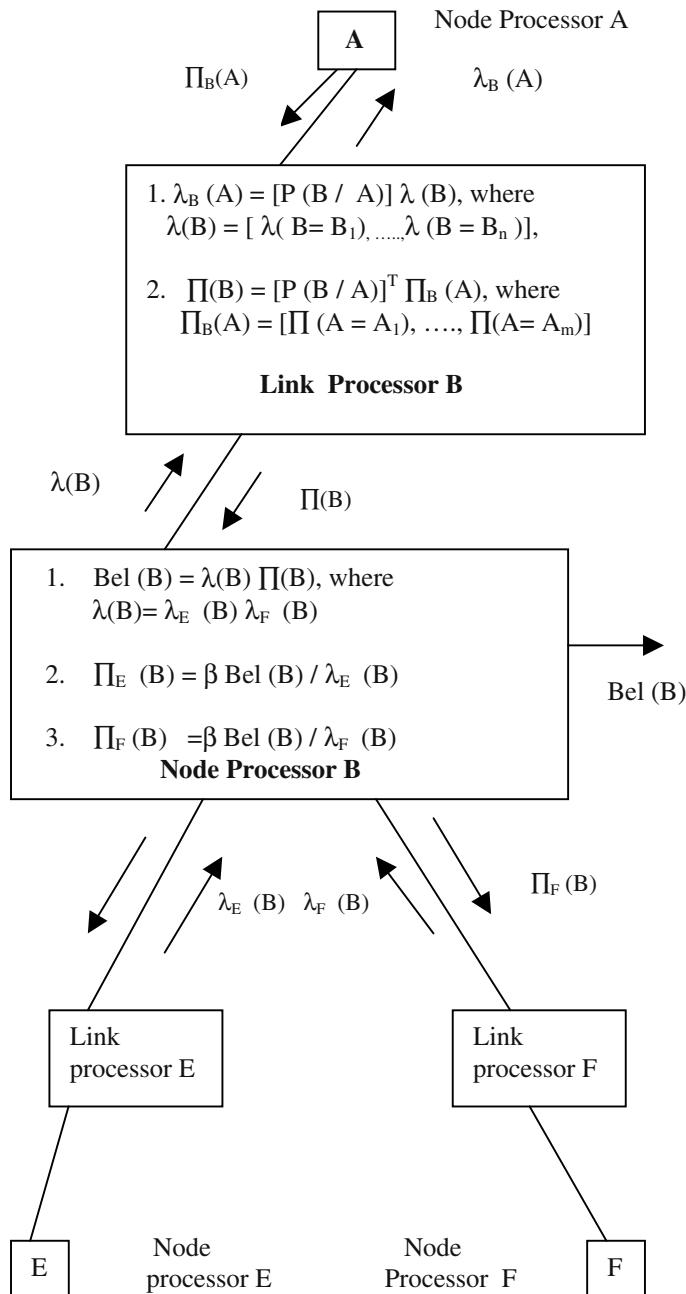
$$\lambda_E(B) = [0 \ 1 \ 0 \ 0 \ 0 \ 0 \dots 0],$$

where the second element corresponds to instantiation of  $E = E_2$ .

3. When a node variable is not instantiated, we calculate its  $\lambda$  values following the formula, outlined in Fig. 13.5.
4. The  $\lambda$  and  $\Pi$  messages are sent to the parents and the children of the instantiated node. For the leaf node there is no need to send the  $\Pi$  message. Similarly, the root node need not send the  $\lambda$  message.
5. The propagation continues from the leaf to its parent, then from the parent to the grandparent, until the root is reached. Then down stream propagation starts from the root to its children, then from the children to grandchildren of the root and so on until the leaves are reached. This is called an equilibrium condition, when the  $\lambda$  and  $\Pi$  messages do not change, unless instantiated further. The belief value at the nodes now reflects the belief of the respective nodes for ‘the car does not start’ (in our example tree).
6. When we want to fuse the beliefs of more than one evidence, we can submit the corresponding  $\lambda$  messages at the respective leaves one after another, and repeat from step 3, otherwise stop.

The resulting beliefs at each node now appear to be the fusion of the joint effect of two or more observed evidences.

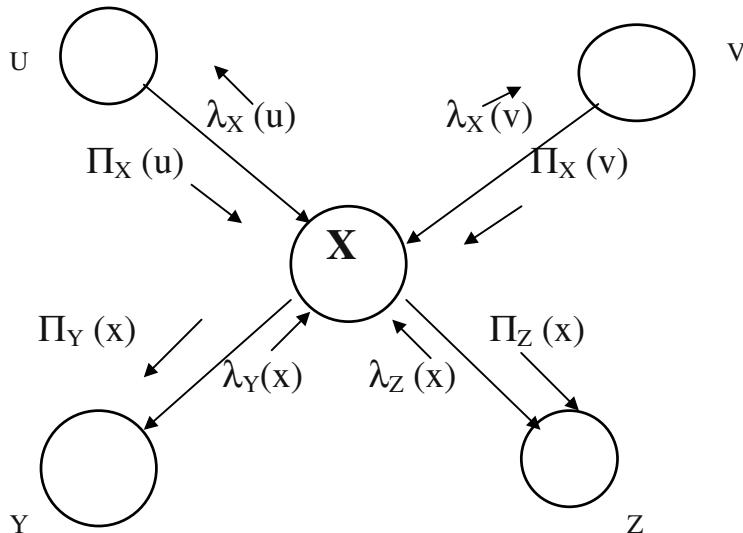
We presented Pearl’s scheme for evidential reasoning for a tree structure only. However, the belief propagation scheme of Pearl can also be extended to **polytrees**, i.e., graphs where the nodes can have more than one parent, but there must be a single arc between each parent to a child and the graph should not have any cycles [12]. We do not derive the formula for belief propagation here, but only state them and illustrate with an example.



**Fig. 13.5:** The computation and propagation of  $\lambda$  and  $\Pi$  messages from and to node B.

### 13.4 Pearl's Belief Propagation Scheme on a Polytree

Let U and V be predecessors of node X, and Y and Z are the successors of node X, as shown in Fig. 13.6. Here, we denote the value of a variable, say V, by lower case notations, say v. Let  $P(x/u, v)$  be the fixed conditional probability matrix that relates the variable to its parents u and v. Let  $\Pi_X(u)$  be the current strength of the causal support, contributed by U to X. Let  $\lambda_Y(x)$  be the current strength of the diagnostic support contributed by Y to X. Causal support represents evidence propagating forward from parents to children, while diagnostic support represents feedback from children to their parents.



**Fig. 13.6:** Propagation of belief through a piece of belief network.

Updating a node X thus involves updating not only its belief function ( $\text{Bel}(x)$ ) but also its  $\lambda$  and  $\Pi$  functions. Belief updating is carried out by the following formula.

$$\text{Bel}(x) = \alpha \lambda_Y(x) \lambda_Z(x) \sum_{u, v} P(x/u, v) \Pi_X(u) \Pi_X(v) \quad (13.42)$$

where  $\alpha$  is a normalizing constant that makes  $\sum \text{Bel}(x) = 1$ .

$$\forall x$$

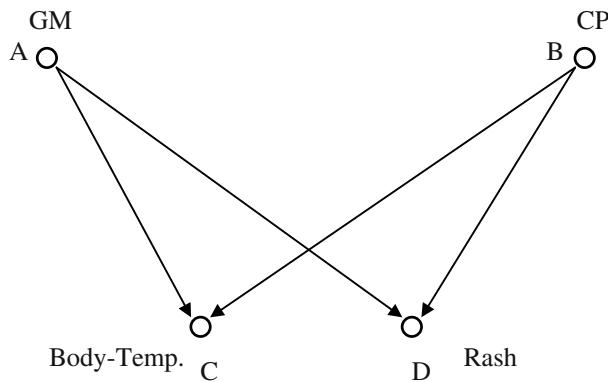
The process of  $\lambda$  and  $\Pi$  updating is now presented below with reference to Fig. 13.6.

$$\lambda_X(u) = \alpha \sum_v [\Pi_X(v) \sum_x [\lambda_Y(x) \lambda_Z(x) P(x/u, v)]] \quad (13.43)$$

$$\Pi_Y(x) = \alpha \lambda_Z(x) [\sum_{u,v} \Pi_X(u) \Pi_X(v) P(x/u, v)] \quad (13.44)$$

For leaves in the network,  $\lambda$ -values are set to one, while for roots (axioms) in the network the  $\Pi$ -values are set equal to their prior probabilities.

**Example 9.5:** To illustrate the process of computing  $\text{Bel}(x)$  at a node  $X$ , let us consider Fig. 13.7.



**Fig. 13.7:** A causal network, representing hypothesis (disease) and evidence (symptom) relationship.

Let the possible value of the hypothesis and evidences be as follows.

$$GM = \{\text{high-GM}, \text{low-GM}\}$$

$$CP = \{\text{high-CP}, \text{little-CP}\}$$

$$\text{Body-Temp} = \{BT \leq 98^\circ F, BT \geq 100^\circ F\}$$

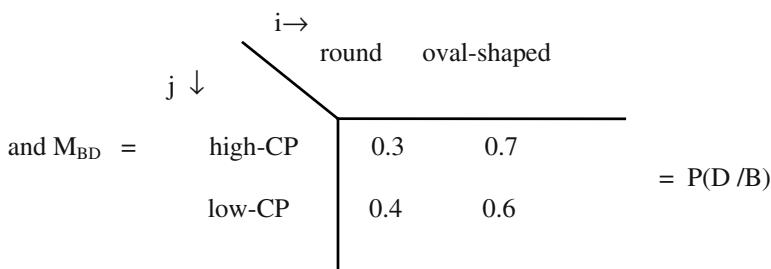
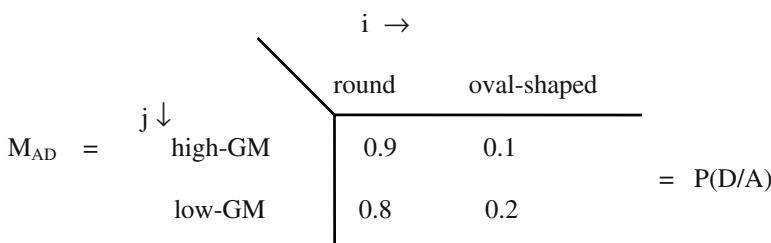
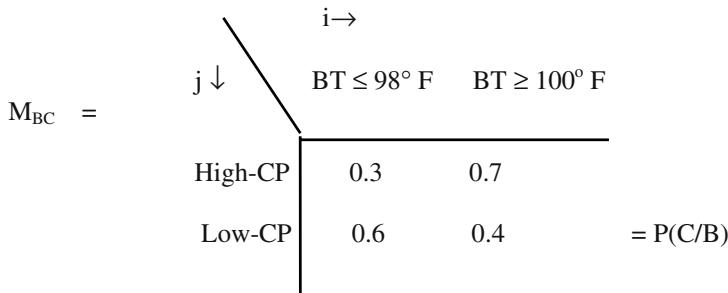
$$\text{Rash} = \{\text{round, oval-shaped}\}$$

The matrices that are associated with the links (arcs) of Fig. 13.7 are presented below.

Let  $M_{AC} =$

		$i \rightarrow$	
		$BT \leq 98^\circ F$	$BT \geq 100^\circ F$
$\downarrow j$	high-GM	0.2	0.8
	low-GM	0.7	0.3

$= P(C/A)$



Suppose, we are interested to compute:

$$\text{Bel}(BT \leq 98^{\circ} F)$$

$$\text{and } \text{Bel}(BT \geq 100^{\circ} F).$$

Now, with reference to Pearl's nomenclature, we assume the following items in Fig. 13.7:

$$\Pi_{BT}(\text{high-GM}) = 0.6$$

$$\Pi_{BT}(\text{low-GM}) = 0.1$$

$\Pi_{BT}$  (high-CP) = 0.25 and

$\Pi_{BT}$  (low-CP) = 0.05

Here,  $\lambda_Y$  (body-temp) = 1.0 and

$\lambda_Z$  (body-temp) = 1.0

Since body-temp (C) is a terminal node in Fig. 13.7, the  $\lambda$  values incoming to it should be unity. Thus by expression (13.42), we find

Unnormalized  $\text{Bel}(BT \leq 98^\circ F)$

$$\begin{aligned}
 &= \Pi_{BT}(\text{high-GM}) \times \Pi_{BT}(\text{high-CP}) \times P(BT \leq 98^\circ F / \text{high-GM, high-CP}) \\
 &+ \Pi_{BT}(\text{high-GM}) \times \Pi_{BT}(\text{low-CP}) \times P(BT \leq 98^\circ F / \text{high-GM, low-CP}) \\
 &+ \Pi_{BT}(\text{low-GM}) \times \Pi_{BT}(\text{high-CP}) \times P(BT \leq 98^\circ F / \text{low-GM, high-CP}) \\
 &+ \Pi_{BT}(\text{low-GM}) \times \Pi_{BT}(\text{low-CP}) \times P(BT \leq 98^\circ F / \text{low-GM, low-CP}) \\
 &= (0.6 \times 0.25 \times 0.2 \times 0.3) + (0.6 \times 0.05 \times 0.2 \times 0.6) + (0.1 \times 0.25 \times 0.7 \times 0.3) + \\
 &\quad (0.1 \times 0.05 \times 0.7 \times 0.6) \\
 &= 0.01995
 \end{aligned}$$

Suppose analogously, we find  $\text{Bel}(BT \geq 100^\circ F) = \beta$  (say)

Then  $\alpha = 1 / (0.01995 + \beta)$ .

So, normalized  $\text{Bel}(BT \leq 98^\circ F) = \alpha \times 0.01995$

and normalized  $\text{Bel}(BT \geq 100^\circ F) = \alpha \times \beta$ .

The  $\lambda$  and  $\Pi$  messages can also be calculated by the formulas supplied. According to Pearl [2], the belief computation in the polytree is done in an asynchronous manner, and at some point of time, the beliefs at all nodes do not change. We call it an equilibrium condition. The belief of the nodes in the polytree at this condition is consistent with the theory of probability.

## 13.5 Dempster-Shafer Theory for Uncertainty Management

The Bayesian formalism assigns a positive belief to a proposition, but it does not take into account of the disbelief of the propositions. Dempster-Shafer (DS) theory, on the other hand, allows information integration by considering both

their belief and disbelief. To illustrate this point, let us consider an example. Suppose that one of the three terrorist groups: A, B and C planted a bomb in an office building in a country. Further, suppose, we have adequate evidence to believe that group C is the guilty one with a measure of belief  $P(C) = 0.8$ , say. On the other hand, without any additional knowledge / fact, we do not like to say that  $P(B) + P(A) = 0.2$ . Unfortunately, we are forced to say so using conventional probability theory as it presumes  $P(\neg C) = 1 - P(C) = P(B) + P(A)$ . This prompted Dempster and his follower Shafer to develop a new theory, well known as the DS theory in the AI community.

In the DS theory, we often use a term, **frame of discernment** (FOD)  $\Theta$ . To illustrate this, let us consider an example of rolling a die. In rolling a die, the set of outcomes could be described by a statement of the form: “the-number-showing-is-i” for  $1 \leq i \leq 6$ . The frame of discernment in the die example is given by

$$\text{FOD } \Theta = \{1, 2, 3, 4, 5, 6\}.$$

Formally, the set of all possible outcomes in a random experiment is called the frame of discernment. Let  $n = |\Theta|$ , the cardinality of  $\Theta$ . Then all the  $2^n$  subsets of  $\Theta$  are called the propositions in the present context. In the die example, the proposition, “the-no-showing-i-is-even” is given by  $\{2, 4, 6\}$ .

In the DS theory, the probability masses are assigned to subsets of  $\Theta$ , unlike Bayesian theory, where probability mass can be assigned to individual elements (singleton subsets). When a knowledge-source of evidence assigns probability masses to the propositions, represented by subsets of  $\Theta$ , the resulting function is called a **basic probability assignment** (BPA).

Formally, a BPA is  $m$

$$\text{where } m : 2^\Theta \rightarrow [0,1]$$

$$\text{where } 0 \leq m(\cdot) \leq 1.0, m(\emptyset) = 0$$

$$\text{and } \sum_{x \subseteq \Theta} m(x) = 1.0 \quad (13.45)$$

**Definition 13.4:** Subsets of  $\Theta$ , which are assigned nonzero probability masses are called **focal elements** of  $\Theta$ .

**Definition 13.5:** A **belief function** [5-6]  $Bel(x)$ , over  $\Theta$ , is defined by

$$Bel(X) = \sum_{Y \subseteq X} m(Y) \quad (13.46)$$

For example, if the frame of discernment  $\Theta$  contains mutually exclusive subsets A, C and D, then

$$\text{Bel} (\{A, C, D\})$$

$$= m(\{A, C, D\}) + m(\{A, C\}) + m(\{A, D\}) + m(\{C, D\}) +$$

$$m(\{a\}) + m(\{c\}) + m(\{d\}).$$

In DS model, belief in a proposition is represented by the belief interval. This is the unit interval [0,1], further demarcated by two points j and k,  $k \geq j$ . Suppose that the belief interval describes proposition A. Then the sub-interval  $[0, j]$  is called Belief (A) and the subinterval  $(k, 1]$  is called the disbelief (A) and the remainder  $[j, k]$  is called Uncertainty (A). **Belief (A)** is the degree to which the current evidence supports A, the **Disbelief (A)** is the degree to which the current evidence supports  $\neg A$  and **Uncertainty (A)** is the degree to which we believe nothing one way or the other about proposition A. As new evidences are collected, the remaining uncertainty will decrease [5], and each piece of length that it loses will be given to Belief (A) or Disbelief (A). The concept can be best described by Fig. 13.8. We denote Belief (A), Disbelief (A) Plausibility (A) and Uncertainty (A) by Bel (A), Disbel (A), Pl (A) and U (A) respectively.

$$\text{Further, } \text{Pl}(A) = \text{Bel}(A) + \text{U}(A) \quad (13.47)$$

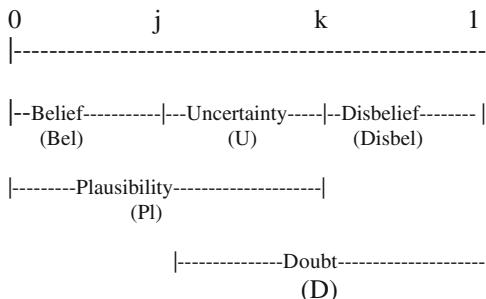
$$\text{and } \text{D}(A) = \text{Disbel}(A) + \text{U}(A). \quad (13.48)$$

It can be easily shown that

$$\text{i) } \text{Pl}(A) \geq \text{Bel}(A) \quad (13.49)$$

$$\text{ii) } \text{Pl}(A) + \text{Pl}(\neg A) \geq 1 \quad (13.50)$$

$$\text{iii) } \text{Bel}(A) + \text{Bel}(\neg A) \leq 1. \quad (13.51)$$



**Fig. 13.8:** The belief interval.

Further, for A being a subset of B,

$$\text{Bel}(A) \leq \text{Bel}(B) \quad (13.52)$$

$$\text{and } \text{Pl}(A) \leq \text{Pl}(B). \quad (13.53)$$

### The Orthogonal Summation of Belief Functions

Assume that two knowledge sources KB1 and KB2 submit two frames of discernments  $\theta_1$  and  $\theta_2$  respectively. Let  $m_1(\cdot)$  and  $m_2(\cdot)$  be the BPA at the subsets of  $\theta_1$  and  $\theta_2$  respectively. The new BPA,  $m(\cdot)$  can be computed based on  $m_1(\cdot)$  and  $m_2(\cdot)$  by using

$$m(X) = K^{-1} \sum_{\substack{X = X_i \cap X_j}} m_1(X_i) \cdot m_2(X_j) \quad (13.54)$$

$$\text{and } K = 1 - \sum_{\substack{X_i \cap X_j = \emptyset}} m_1(X_i) \cdot m_2(X_j) \quad (13.55)$$

where  $X_i$  and  $X_j$  are focal elements of  $\theta_1$  and  $\theta_2$  respectively. We denote the **orthogonal summation operation**, referred to above, by  $m = m_1 \oplus m_2$ .

To illustrate the orthogonal summation process, let us consider the BPAs that are assigned by two knowledge sources through a image recognition process.

Let us assume that knowledge source 1 (KS1) claims that an unknown object in a scene could be

- a chair with  $m_1(\{C\}) = 0.3$ ,
- a table with  $m_1(\{T\}) = 0.1$ ,
- a desk with  $m_1(\{D\}) = 0.1$ ,
- a window with  $m_1(\{W\}) = 0.15$ ,
- a person with  $m_1(\{P\}) = 0.05$ ,
- and the frame  $\theta$ , with  $m_1(\{\theta\}) = 0.3$ .

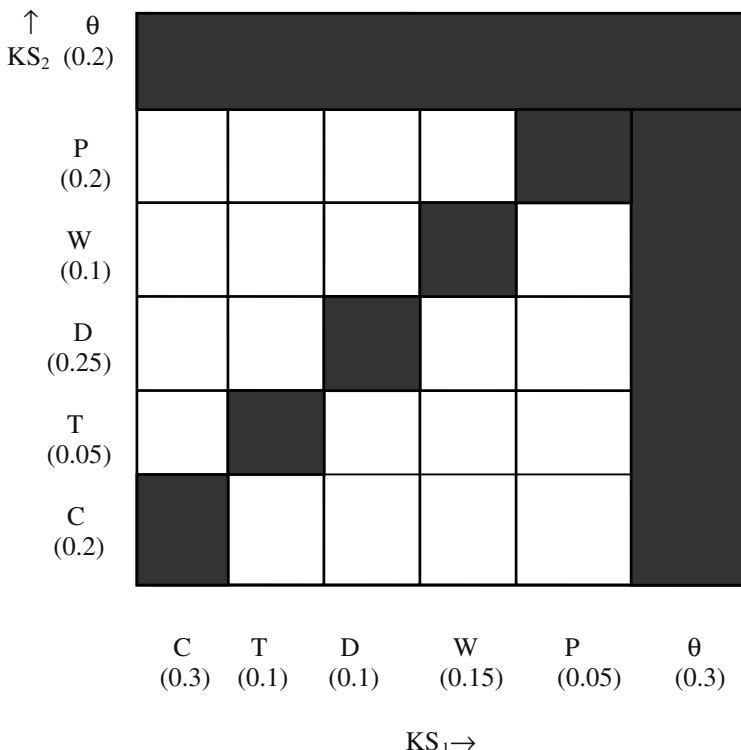
The assignment of  $BPA = 0.3$  to  $\theta$  means that knowledge source 1 knows that something in  $\theta$  has occurred, but it does not know what it exactly is. Analogously, knowledge source 2 (KS2) claims the same object in the scene to be

- a chair with  $m_2(\{C\}) = 0.2$ ,
- a table with  $m_2(\{T\}) = 0.05$ ,
- a desk with  $m_2(\{D\}) = 0.25$ ,
- a window with  $m_2(\{W\}) = 0.1$ ,

a person with  $m_2(\{P\}) = 0.2$ ,  
and the frame  $\theta$  with  $m_2(\{\theta\}) = 0.2$

Now, suppose, we are interested to compute “What is the composite belief of the object to be a chair ?”

To compute this we construct the following table.



**Fig. 13.9:** Illustrating the principle of orthogonal summation.

Now,  $m_{12}(\{C\})$

$$\begin{aligned}
 & m_1(\{C\}). m_2(\{C\}) + m_1(\{\theta\}). m_2(\{C\}) + m_1(\{C\}). m_2(\{\theta\}) \\
 & = \frac{\text{Sum of the area of the shaded blocks}}{=}
 \end{aligned}$$

$$\begin{aligned}
 & 0.2 \times 0.3 + 0.2 \times 0.3 + 0.2 \times 0.3 \\
 = & \frac{0.555}{0.555} \\
 = & 0.32
 \end{aligned}$$

i.e.,  $\text{Bel}_1(C) \oplus \text{Bel}_2(C) = 0.32$ .

The orthogonal summation operations of more than two belief functions can be computed in an analogous manner, by taking two belief functions, one at a time. The major drawback of this technique is high time-complexity, which in the worst case may be as high as  $p_1 \times p_2$ , where  $p_1$  and  $p_2$  represent the hypothesis space of the two sources of evidences. Thus for combining belief from  $n$  sources, the overall time-complexity in the worst case is  $p_1 \times p_2 \times \dots \times p_n$ , where  $p_i$  represents the number of hypothesis in the  $i$ -th knowledge source. Summarizing, the above concept, the worst case time-complexity for  $n$  composition of beliefs from  $n$  sources is  $O(p^n)$ , where  $p_1 = p_2 = \dots = p_n = p$ , say. This exponential time-complexity can be reduced [7], by performing belief combinations on local families, instead of combining beliefs on the entire frames.

## 13.6 Conclusions

The chapter introduced Bayes' classical law on conditional probability and its extension for Bayesian reasoning following Judea Pearl's model. It also outlined the foundation of Dempster-Shafer theory and illustrated the underlying theory with an exemplar problem of object recognition from noisy images. The Bayesian reasoning scheme requires a number of conditional probabilities, the determination of which in practical systems is not always feasible. However, in laboratory-oriented problems, determination of the conditional probabilities does not pose a major problem. Consequently, Bayesian reasoning model has a restricted use in practical reasoning systems. The Dempster-Shafer theory is widely being used as a tool for data fusion. One significant problem in using this theory is its polynomial time-complexity with increasing number of source evidences. In [9] Shenoy and Shafer, however, attempted alternative approach to reduce the complexities of the theory.

## Exercise

- 1.a) There are four terrorist-groups A, B, C, D and three sources of evidences namely source1, source2, source3. Let  $m_i$  be the basic probability assignment by the source<sub>i</sub> for  $i = 1, 2, 3$ .

Given,

$$\begin{aligned}m_1 \{A, C\} &= 0.6 \\m_2 \{A, B, D\} &= 0.7\end{aligned}$$

Determine  $m_{12} \{A\}$ ,  $m_{12} \{A, C\}$ ,  $m_{12} \{A, B, D\}$ ,  $m_{12} \{U\}$ , where  $U$  is the frame of discernment.

- b) Suppose now that new evidences ( $m_3$ ) indicates that organization  $C$  is indeed responsible to a degree of 0.8. Compute  $m_{34} \{C\}$ , where  $m_4 = m_1 \oplus m_2$ . Note that the new intersection tableau for the above computation should include entries for  $m_3 \{C\}$  and  $m_3 \{U\}$  versus  $m_4 \{A\}$ ,  $m_4 \{C, A\}$ ,  $m_4 \{A, B, D\}$ , and  $m_4 \{U\}$ .

[Hints: Part a) Since we are given with  $m_1 \{A, C\}$  and  $m_2 \{A, B, D\}$ , we construct a table with entries  $m_1 \{A, C\}$ ,  $m_1 \{U\}$ ,  $m_2 \{A, B, D\}$  and  $m_2 \{U\}$ .

		$m_2 \rightarrow$	
$m_1 \downarrow$		$\{A, B, D\} (0.7)$	$U (0.3)$
$\{A, C\} (0.6)$		$\{A\} (0.42)$	$\{A, C\} (0.18)$
$U (0.4)$		$\{A, B, D\} (0.28)$	$U (0.12)$

Therefore,

$$\begin{aligned}m_{12} \{A\} &= m_1 \{A, C\} \oplus m_2 \{A, B, D\} \\&= 0.42 / \{0.42 + 0.18 + 0.28 + 0.12\} = 0.42\end{aligned}$$

$$\begin{aligned}m_{12} \{A, C\} &= m_1 \{A, C\} \oplus m_2 \{U\} \\&= 0.18 / \{0.42 + 0.18 + 0.28 + 0.12\} = 0.18 \\m_{12} \{A, B, D\} &= m_1 \{U\} \oplus m_2 \{A, B, D\} \\&= 0.28 / \{0.42 + 0.18 + 0.28 + 0.12\} = 0.28\end{aligned}$$

$$m_{12}(U) = m_1(U) \oplus m_2(U)$$

$$= 0.12 / \{0.42 + 0.18 + 0.28 + 0.12\} = 0.12$$

Part b) Given,  $m_3(C) = 0.8$ ,

so,  $m_3(U) = 0.2$ .

Table of combined values of belief for  $m_1$ ,  $m_2$  and  $m_3$  is given below.

		$m_{12}$		
$m_3$	$\{A\}$ (0.42)	$\{A, C\}$ (0.18)	$\{A, B, D\}$ (0.28).	$U$ (0.12)
$C$ (0.8)	No intersection	$\{C\}$ ( $0.8 \times 0.18$ )	No intersection	$\{C\}$ ( $0.8 \times 0.12$ )
	$\{A\}$ ( $0.42 \times 0.2$ )	$\{A, C\}$ ( $0.2 \times 0.18$ )	$\{A, B, D\}$ ( $0.28 \times 0.2$ )	$\{U\}$ ( $0.12 \times 0.2$ )

Now,

$$m_{34}(C) = \frac{m_3(C) \times m_{12}\{A, C\} + m_3(C) \times m_{12}(U)}{\text{area of all valid intersections}}$$

where

$$m_3(C) \times m_{12}\{A, C\} + m_3(C) \times m_{12}(U)$$

$$= 0.8 \times 0.18 + 0.8 \times 0.12 = 0.24$$

Area of valid intersections

$$= 0.8 \times 0.18 + 0.8 \times 0.12 + 0.42 \times 0.2 + 0.2 \times 0.18 + 0.28 \times 0.2 + 0.12 \times 0.2$$

$$= 0.44.$$

Therefore,  $m_{34}(C) = 0.24 / 0.44$

$$\begin{aligned} &= 6/11 \\ &= 0.545] \end{aligned}$$

2. From basic set theory prove that  $P(\neg A) = 1 - P(A)$ , and that  $P(\neg B / A) = 1 - P(B/A)$ .

To prove  $P(\neg A) = 1 - P(A)$

[**Hints:** Steps.

1.  $P(A \cup \neg A) = P(A) + P(\neg A) - P(A \cap \neg A)$  (Definition)
2.  $P(A \cup \neg A) = 1$  (Universal truth)
3.  $P(A) + P(\neg A) - P(A \cap \neg A) = 1$  (by 1 and 2)
4.  $P(A) + P(\neg A) = 1$  (as  $P(A \cap \neg A) = 0$ )
5.  $P(\neg A) = 1 - P(A)$ . (Proved).

Now to prove:  $P(\neg B / A) = 1 - P(B/A)$ , we by Bayes' theorem write:

$$P(\neg B / A) = \frac{P(\neg B \cap A)}{P(A)}$$

Again,

$$P(B/A) = \frac{P(B \cap A)}{P(A)}$$

Now,  $P(\neg B / A) + P(B/A)$

$$= \frac{P(\neg B \cap A)}{P(A)} + \frac{P(B \cap A)}{P(A)}$$

$$\begin{aligned} &= \frac{P(A)}{P(A)} \\ &= 1 \end{aligned}$$

Therefore,  $P(\neg B / A) = 1 - P(B/A)$  ( proved).]

3. Prove that if A and B are independent,  $P(A/B) = P(A)$ .

[**Hints:**

$$\begin{aligned} P(A/B) &= \frac{P(A \cap B)}{P(B)} \\ &= \frac{P(A) \cdot P(B)}{P(B)} \\ &= P(A) \end{aligned}$$

4. Consider a temporal world describing the seasons in a year. Probability that now is winter is denoted by  $p(\text{winter})$ . Thus for 4 seasons, suppose we are given that

$$\begin{aligned} P(\text{winter}) &= 1/4 \\ P(\text{summer}) &= 1/2 \\ P(\text{spring}) &= 1/8 \\ P(\text{rainy season}) &= 1/8 \end{aligned}$$

Let  $s$  be a fact denoting that it is very cold today. Given that

$$\begin{aligned} P(s/\text{winter}) &= 0.2 \\ P(s/\text{rainy season}) &= 0.6 \\ P(s/\text{summer}) &= 0.1 \\ P(s/\text{spring}) &= 0.1 \end{aligned}$$

Determine the probability that the statement  $s$  is true.

**Hints:**  $P(s) = P(s \cap \text{winter}) + P(s \cap \text{Rainy season}) + P(s \cap \text{summer}) + P(s \cap \text{spring})$

$$\begin{aligned}
 &= P(s/winter) \times P(winter) + P(s/rainy season) \times P(rainy season) + \\
 &\quad P(s/summer) \times P(summer) + P(s/spring) \times P(spring). \\
 \\
 &= 0.2 \times 1/4 + 0.6 \times 1/8 + 0.1 \times 1/2 + 0.1 \times 1/8 \\
 &= \frac{0.4 + 0.6 + 0.4 + 0.1}{8} \\
 &= \frac{1.5}{8} = 0.187 ]
 \end{aligned}$$

5. Is it possible to compute  $P(A/\neg B)$  when you are only given

$P(A)$ ,  $P(B/A)$ , and  $P(B)$ ?

[**Hints:** Yes,  $P(A/\neg B)$  can be computed if  $P(A)$ ,  $P(B/A)$ , and  $P(B)$  are given.]

$$P(A/\neg B) = \frac{P(A \cap \neg B)}{P(\neg B)} \quad (1)$$

$$P(A \cap B) = P(B/A) P(A) \quad (2)$$

$$\text{and } P(\neg B) = P(1 - P(B)) \quad (3)$$

Therefore by substituting the equations (2) and (3) we can compute (1)]

6. Find the probability of the event A when it is known that some event occurred. Suppose, we experimentally obtained:

$$P(B/A) = 0.84, P(A) = 0.2 \text{ and } P(B) = 0.34.$$

[**Hints:**  $P(A|B) = P(B|A) \cdot P(A)/P(B).$ ]

7. Suppose we obtained the following beliefs from 2 sensors in connection with a crime committed by one of 3 members of a team name X, Y and Z. The

clause that X has committed the crime is denoted by X only for brevity. Let  $\theta$  be the frame of discernment.

Source 1 says:

$$\text{Bel}_1(X) = 0.2, \text{Bel}_1(Y) = 0.3,$$

$$\text{Bel}_1(Z) = 0.1, \text{Bel}_1(\theta) = 0.4.$$

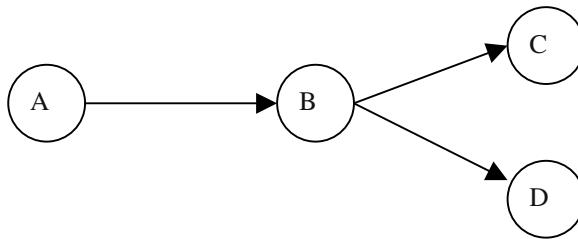
Source 2 says:

$$\text{Bel}_2(X) = 0.1, \text{Bel}_2(Y) = 0.2,$$

$$\text{Bel}_2(Z) = 0.4, \text{Bel}_2(\theta) = 0.3.$$

Determine the composite belief of X committing the crime by Dempster-Shafer theory.

8. Consider the belief network shown in Fig. 13.10. A, B, C and D are Boolean variables. The conditional probability tables involving these variables are given below.



**Fig. 13.10:** A piece of belief network.

$$P(A) = 2/3, P(B/A) = 1/2, P(B/\neg A) = 1/4, P(C/B) = 4/5,$$

$$P(C/\neg B) = 1/5, P(D/B) = 5/6, P(D/\neg B) = 1/3$$

- (i) Compute  $P(A/B)$  in terms of the given probability values.
- (ii) Compute  $P(C/A)$

[**Hints:** Part (1)

$$P(A/B) = \frac{P(B/A) \cdot P(A)}{P(B)}$$

where

$$P(A) = 2/3, P(B/A) = 1/2 \text{ and}$$

$$\begin{aligned} P(B) &= P(B \cap A) + P(B \cap \neg A) \\ &= P(B/A) \cdot P(A) + P(B/\neg A) \cdot P(\neg A) \\ &= \frac{1}{2} \times \frac{2}{3} + \frac{1}{4} \times (1 - \frac{2}{3}) \\ &= \frac{1}{3} + \frac{1}{12} \\ &= 5/12. \end{aligned}$$

Thus,  $P(A/B) = (2/3) (1/2) / (5/12) = 12/15.$

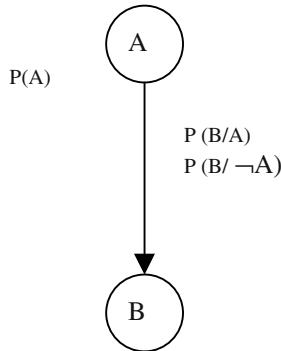
$$\begin{aligned} \text{Part (ii): } P(C/A) &= \frac{P(A/B \cap C) \cdot P(B \cap C)}{P(A)} \\ &= \frac{P(A/B) \cdot P(B \cap C)}{P(A)}, \quad \text{as } A \text{ is independent of } C. \\ &= \frac{P(A/B) \cdot P(C/B) \cdot P(B)}{P(A)} \\ &= \frac{(12/15) (4/5) (5/12)}{(2/3)} \\ &= 6/15 = 0.4. \end{aligned}$$

9. Given the causal dependence of events:  $A \rightarrow B \rightarrow C$ , where  $A \rightarrow B$  means A causes B to happen. Compute  $P(A, B, C)$  when  $P(C/B) = 0.8$ ,  $P(B/A) = 0.4$  and  $P(A) = 0.9$

[**Hints:** According to the definition,

$$\begin{aligned} P(A, B, C) &= P(C/B) \times P(B/A) \times P(A) \\ &= 0.8 \times 0.4 \times 0.9 \\ &= 0.288. \end{aligned}$$

10. Consider the belief network shown in Fig. 13.11.



**Fig. 13.11:** A simple belief network for problem 10.

- i) Derive an expression for the probability of  $A \supset B$  (i.e.  $A \rightarrow B$ ).
- ii) When are  $P(A \supset B)$  and  $P(B/A)$  equal?
- iii) Assume the conditional probability table for the network is not known. Instead all that is known are the values of  $P(A)$  and of  $P(A \supset B)$ . What can be said about the value of  $P(B)$ ?

[**Hints:** Part (1) Consider the following table.

	A	$\neg A$
B		
$\neg B$		

$$\begin{aligned}
 & P(A \rightarrow B) \\
 &= 1 - P(A \wedge \neg B) \\
 &= 1 - \text{unshaded area} \\
 &= 1 - P(A \wedge B) + P(\neg A \wedge B) + P(\neg A \wedge \neg B).
 \end{aligned} \tag{1}$$

where

$$\begin{aligned}
 & P(A \wedge B) \\
 &= P(B \wedge A) \\
 &= P(B/A) \cdot P(A)
 \end{aligned} \tag{2}$$

and  $P(\neg A \wedge B)$

$$\begin{aligned}
 &= P(B \wedge \neg A) \\
 &= (B/\neg A) \cdot P(\neg A) \\
 &= P(B/\neg A) \cdot (1 - P(A))
 \end{aligned} \tag{3}$$

and  $P(\neg A \wedge \neg B)$

$$\begin{aligned}
 &= P(\neg B/\neg A) \cdot P(\neg A) \\
 &= (1 - P(B/\neg A)) \cdot (1 - P(A))
 \end{aligned} \tag{4}$$

Substituting (2), (3) and (4) in (1) we obtain  $P(A \rightarrow B)$ .

Part (ii): See Table below.

		A	$\neg A$
		B	
B	$\neg B$		

$$\begin{aligned}
 P(B/A) &= \frac{P(A \wedge B)}{P(A)} \\
 &= \frac{P(A \wedge B)}{P(A \wedge B) + (A \wedge \neg B)}
 \end{aligned} \tag{5}$$

$$P(A \rightarrow B) = 1 - P(A \wedge \neg B) \tag{6}$$

When  $P(\neg A) = 0$ ,

$$P(A \wedge B) + P(A \wedge \neg B) = 1. \tag{7}$$

Substituting (7) in (5) we obtain:

$$\begin{aligned} P(B/A) &= \frac{P(A \wedge B)}{1} \\ &= P(A \wedge B). \end{aligned} \quad (8)$$

Further, by (6) we have:

$$\begin{aligned} P(A \rightarrow B) &= 1 - P(A \wedge \neg B) \\ &= P(A \wedge B) \quad (\text{by (7)}) \end{aligned} \quad (9)$$

From (9) and (8) we note that when  $P(\neg A) = 0$ ,  $P(A \rightarrow B) = P(B/A)$ .

Part (iii): We are given  $P(A)$  and  $P(A \rightarrow B)$

$$P(B) = P(A \wedge B) + P(\neg A \wedge B) \quad (10)$$

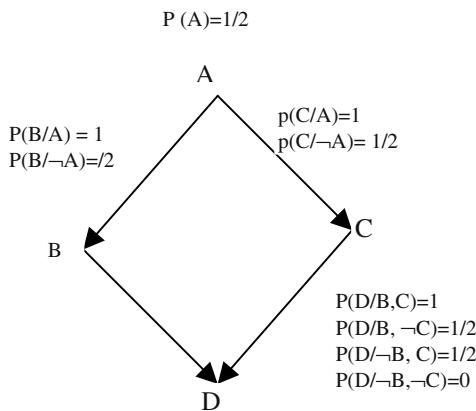
where

$$P(A \wedge B) = P(B/A) \cdot P(A), \quad (11)$$

$$\begin{aligned} P(\neg A \wedge B) \\ &= P(B \wedge \neg A) \\ &= P(B/\neg A) \cdot P(\neg A) \\ &= P(B/\neg A) \cdot (1 - P(A)) \end{aligned} \quad (12)$$

Thus, we obtain  $P(B)$  by substituting (11) and (12) in (9).]

11. An admission committee for a college is trying to determine the probability that an admitted is really qualified. The relevant probabilities are given in the Bayes network shown here. Calculate  $P(A/D)$



A= applicant is qualified

B = applicant has high grade point average

C= applicant has excellent recommendations

D= applicant is admitted

**Fig. 13.12:** A belief network illustrating problem 11.

$$\begin{aligned}
 [\text{Hint: } P(B) &= P(B \cap A) + P(B \cap \neg A) \\
 &= P(B/A) P(A) + P(B/\neg A) P(\neg A) \\
 &= \frac{1}{2} + \frac{1}{4} \\
 &= \frac{3}{4}
 \end{aligned}$$

$$\begin{aligned}
 P(C) &= P(C \cap A) + P(C \cap \neg A) \\
 &= P(C/A) P(A) + P(C/\neg A) P(\neg A) \\
 &= \frac{1}{2} + \frac{1}{4} \\
 &= \frac{3}{4}
 \end{aligned}$$

$$\begin{aligned}
 P(D) &= P(D/B,C) \times P(B,C) + P(D/B, \neg C) \times P(B, \neg C) + \\
 &\quad P(D/\neg B, C) \times P(\neg B, C) + P(D/\neg B, \neg C) \times P(\neg B, \neg C) \\
 &= 1 \times \{P(B) \times P(C)\} + \frac{1}{2} \times \{P(B) \times P(\neg C)\} + \frac{1}{2} \times \{P(\neg B) \times P(C)\} \\
 &= 1 \times (\frac{3}{4})^2 + \frac{1}{2} \times \frac{3}{4} \times \frac{1}{4} + \frac{1}{2} \times \frac{1}{4} \times \frac{3}{4} \\
 &= \frac{3}{4}.
 \end{aligned}$$

For computing  $P(A/D)$  we also need to compute  $P(D/A)$ .

$$\begin{aligned}
 & P(D/A) \\
 &= \frac{P(A/ B \cap C \cap D) \cdot P(B \cap C \cap D)}{P(A)} \\
 &= \frac{P(A) \cdot P(D/ B \cap C) \cdot P(B) \cdot P(C)}{P(A)} \\
 &= 1 \times (3/4)^2 \\
 &= 9/16
 \end{aligned}$$

Therefore,

$$\begin{aligned}
 & P(A/D) \\
 &= \frac{P(D/A) \times P(A)}{P(D)} \\
 &= \frac{(9/16) \times (1/2)}{3/4} \\
 &= 3/8 ]
 \end{aligned}$$

12. Assuming that some of the probabilities involved is zero, show that

$$P(A \cap B/C) = P(A/C) \cdot P(B/C) \text{ if and only if}$$

$$P(A/ B \cap C) = P(A/C).$$

What probabilities can be without losing this result?

[**Hints:**  $P(A \cap B/C)$

$$= \frac{P((A \cap B) \cap C)}{P(C)}$$

$$\begin{aligned}
 &= \frac{P(A \cap (B \cap C))}{P(C)} \\
 &= \frac{P(A/B \cap C) \cdot P(B \cap C)}{P(C)} \\
 &= \frac{P(A/B \cap C) \cdot P(B/C) \cdot P(C)}{P(C)} \\
 &= P(A/B \cap C) \cdot P(B/C)
 \end{aligned}$$

Thus,  $P(A \cap B/C)$

$$= P(A/C) \cdot P(B/C) \text{ when } P(A/B \cap C) = P(A/C).]$$

13. Show that when  $P(B)=0$ ,  $P(B \rightarrow A) = P(\neg B) + P(A \cap B)$  and thus

$$1 - P(B \rightarrow A) = P(B)(1 - P(A/B)).$$

**[Hints:**  $P(B \rightarrow A)$

$$\begin{aligned}
 &= 1 - P(B \cap \neg A) \\
 &= P(\neg B \cap A) + P(\neg B \cap \neg A) + P(A \cap B) \\
 &= P(\neg B) + P(A \cap B) \text{ (the first part is proved)} \\
 &= P(\neg B) + P(A/B) \cdot P(B) \\
 &= 1 - P(B) + P(A/B) \cdot P(B) \\
 &= 1 - P(B)(1 - P(A/B))
 \end{aligned}$$

or,  $1 - P(B \rightarrow A)$

$$\begin{aligned}
 &= 1 - \{ 1 - P(B)(1 - P(A/B)) \} \\
 &= P(B)(1 - P(A/B).]
 \end{aligned}$$

## References

- [1] Castillo, E., Gutierrez, J. M. and Hadi, A. S., *Expert Systems and Probabilistic Network Models*, Springer-Verlag, New York, 1997.
- [2] Patterson, D. W., *Introduction to Artificial Intelligence and Expert Systems*, Prentice-Hall, Englewood Cliffs, NJ, pp. 107-119, 1990.
- [3] Pearl, J., "Fusion, propagation and structuring in belief networks," *Artificial Intelligence*, vol. 29, pp. 241-288, 1986.
- [4] Pearl, J., "Distributed revision of composite beliefs," *Artificial Intelligence*, vol. 33, pp. 173-213, 1987.
- [5] Peng, Y. and Reggia, J. A., "A probabilistic causal model for diagnostic problem solving," *IEEE Trans. on Systems, Man and Cybernetics*, SMC-17, no. 3, pp. 395-408, May-June 1987.
- [6] Russel, S. and Norvig, P., *Artificial Intelligence: A Modern Approach*, Prentice-Hall, NJ, 1995.
- [7] Shafer, G., *A Mathematical Theory of Evidence*, Princeton University Press, Princeton, NJ, 1976.
- [8] Shafer, G. and Logan, R., "Implementing Dempster's rule for hierarchical evidence," *Artificial Intelligence*, vol. 33, pp. 271-298, 1987.
- [9] Shenoy, P. P. and Shafer, G., "Propagating belief functions with local computations," *IEEE Expert*, pp. 43-52, Fall 1986.
- [10] Shoham, Y., *Artificial Intelligence Techniques in PROLOG*, Morgan Kaufmann, San Mateo, CA, pp. 183-185, 1994.
- [11] Stefik, M., *Introduction to Knowledge Systems*, Morgan Kaufmann, San Mateo, CA, pp. 493-499, 1995.

# 14

## Reasoning in Expert Systems Using Fuzzy Petri Nets

*The chapter aims at developing new techniques for uncertainty management in expert systems for two generic class problems using fuzzy Petri net that represents logical connectivity among a set of imprecise propositions. One class of problems addressed in the chapter deals with the computation of fuzzy belief of any proposition from the fuzzy beliefs of a set of independent initiating propositions in a given network. The other class of problems is concerned with the computation of steady state fuzzy beliefs of the propositions embedded in the network, from the initial fuzzy beliefs through a process called belief-revision. During belief-revision, a fuzzy Petri net with cycles may exhibit “limitcycle behavior” of fuzzy beliefs for some propositions in the network. No decisions can be arrived at from a fuzzy Petri net with such behavior. To circumvent this problem, techniques have been developed for the detection and elimination of limitcycles. Further, an algorithm for selecting one evidence from each set of mutually inconsistent evidences, referred to as nonmonotonic reasoning, has also been presented in connection with the problems of belief-revision. Finally the concepts proposed for solving the problems of belief-revision have been*

applied successfully for tackling imprecision, uncertainty, and nonmonotonicity of evidences in an illustrate expert system for criminal investigation.

## 14.1 Introduction

An expert system (ES) generally embodies a knowledge base (KB), a database (DB) of facts, and an inference engine (IE) for interpreting the DB with the help of the KB. DB of real world problems often contains imprecise data; the KB too includes pieces of knowledge, whose certainty is not always guaranteed. Further, due to non-availability of data from authentic sources and incompleteness of data or knowledge, inconsistent data elements often creep into the reasoning space. The chapter aims at developing a unified approach for reasoning in ES in presence of imprecise and inconsistent DB and uncertain KB.

In order to increase the reasoning efficiency [18] of the IE for such an ES, all the above forms for inexactness of DB and KB are required to be mapped onto a structured parallel distributed architecture. Petri nets (PN) [9], which is a directed bipartite graph with a high degree of structural parallelism and pipelining, is an ideal choice for the proposed IE. The input and the output places of each transition in a PN can be used to represent the antecedent-consequent pairs of the pieces of knowledge, represented by production rules (PR). The degree of precision of the propositions can be mapped as markings of the corresponding places representing the propositions. The certainty factor (CF) of the PR is assigned to the appropriate transitions. A threshold, which is also assigned to each transition, represents a lower bound on the degree of propositions that the input places of the transition should possess in order to fire the transition. After a transition fires, the new markings at its output places are evaluated. It may, however, be added that the markings from the input places of the transitions are not removed after transition-firing, as is done in classical PN. The well-known problem of *structural conflict* [13], therefore, does not arise in the present context.

It is to be noted that in the proposed PN model, when one (or more) consequent proposition of an  $i$ th PR and antecedent propositions of a  $j$ th PR is (are) common, the  $i$ th transition always precedes the  $j$ th transition. The pipelining in the execution of the rules based on their dependence relationship is thus established in the PN models. In addition, the highest possible degree of parallelism underlying the IE is fully supported by the PN models, since the transition in the PN, where the enabling conditions are satisfied simultaneously, can fire concurrently. The throughput [18] of the IE could thus be improved significantly by implementing it on a PN.

Fuzzy logic [6], [21] has been used here for its simplicity coupled with its adequate power of humanlike reasoning. The degree of truth of propositions,

CF, and thresholds are, therefore, all mapped into the fuzzy truth scale [0, 1]. Further the operations to be carried out on the proposed PN are simple fuzzy AND and OR operations. The PN thus used for reasoning is called fuzzy PN (FPN), following Looney, who pioneered the concept of imprecision management in decision-making problems using acyclic FPN [11]. Chen et al., too, attempted to develop an algorithm for reasoning [5] in a more generic class of FPN, which, however, is plagued by a number of shortcomings. For example, the cycles were virtually opened up in the algorithm itself. Secondly, the algorithm cannot deal with FPN having intermediate places (defined in Section 14.3) with more than one input arc. Finally, the dependence of the degree of precision of the concluding place on the length of the reasoning paths is undesirable. There exist an extensive works on FPN [1-14], each addressing one or more issues related to this new discipline. In this chapter, we, however, discuss aspects of forward reasoning in an FPN following the author's previous works [8] on the subject.

Two distinct models of FPN constructed for automated reasoning in ES are presented in this chapter. The first model deals with the computation of the degree of precision, hereafter called fuzzy belief of a given concluding proposition, based on the pre-assigned fuzzy beliefs of the independent starting propositions in an acyclic FPN. The algorithm proposed for reasoning in ES using this model is free from the limitations found in [5].

The second model is concerned with the computation of steady state fuzzy beliefs at the places in the network from their pre-assigned (initial) fuzzy beliefs. An algorithm for reasoning in ES based on the model has been developed by extending Looney's algorithm to encompass a generic class of FPN that may include cycles. Belief updating process in the proposed algorithm is carried out for all places synchronously. It is possible that after a finite number of belief revision in the FPN, fuzzy beliefs at each place on the cycle may be repeated, exhibiting periodic temporal oscillation of belief at the places on the cycle. No inference can be derived from an FPN with sustained periodic oscillation (PO), hereafter called limitcycles (LC). A condition for the existence of LC has been derived and the principle used for its elimination has also been discussed. Finally, the principle of inconsistency management has been presented with reference to the proposed model.

The chapter has been subdivided into four sections. The algorithm for reasoning in acyclic FPN have been discussed in Section 14.2, while the algorithms for cyclic FPN have been presented in Section 14.3. The concepts introduced in Section 14.3 have been illustrated in Section 14.4 using a simulated criminology problem. An estimate of time-complexity of the algorithm for the above problem and the justification of the proposed technique with reference to the existing methods has also been summarized in Section 14.4.

## 14.2 Imprecision Management in an Acyclic FPN

Before describing the technique for imprecision management in acyclic FPN, we present a few terminologies first.

### 14.2.1 Formal Definitions and the Proposed Model

**Definition 14.1:** An FPN can be defined as a 9-tuple:

$$FPN = \{P, Tr, D, I, O, cf, th, n, b\}$$

where

- $P = \{p_1, p_2, \dots, p_m\}$  is a finite set of places,
- $Tr = \{tr_1, tr_2, \dots, tr_n\}$  is a finite set of transitions.
- $D = \{d_1, d_2, \dots, d_m\}$  is a finite set of propositions,
- $P \cap Tr \cap D = \emptyset$ ,  $|P| = |D|$ ,
- $I : Tr \rightarrow P^\infty$  is the input function, representing a mapping from transitions to bags of (their input) places,
- $O : Tr \rightarrow P^\infty$  is the output function, representing a mapping from transitions to bags of (their output) places,
- $cf, th : Tr \rightarrow [0, 1]$  are association functions, representing a mapping from transitions to real values between 0 and 1
- $n : P \rightarrow [0, 1]$  is an association function, representing a mapping from places to real values between 0 and 1
- $b : P \rightarrow D$  is an association function, representing a bijective mapping from places to propositions.

In realistic terminology,  $n_i$  represents the fuzzy beliefs of place  $p_i$  i.e.,  $n_i = n(p_i)$ ;  $cf_j = cf(tr_j)$  and  $th_j = th(tr_j)$  represent the CF and threshold of transition  $tr_j$  respectively. Further  $d_i = b(p_i)$ .

**Definition 14.2:** A transition  $tr_i$  is enabled if AND  $\{n_i : p_i \in I(tr_i)\} > th_i$  where  $n_i = n(p_i)$  and  $th_i = th(tr_i)$ . An enabled transition fires, resulting in a fuzzy truth token (FTT) at all its output arcs. The value of the FTT is a function of the CF of the transition and fuzzy beliefs of its input places.

The technique for computing the fuzzy beliefs at a place can be conveniently represented by a model. Based on Definitions 14.1 and 14.2 we propose a model, called the Belief Propagation Model.

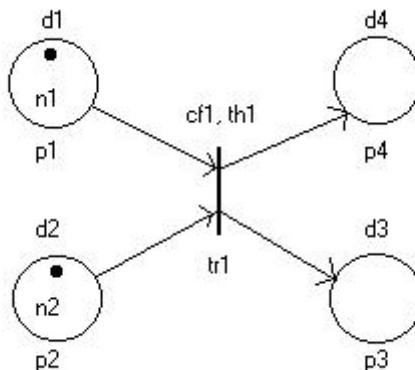
### 14.2.2 Proposed Model for Belief Propagation

Consider a place  $p_i$  which is one of the common output places of transitions  $tr_j$ , where  $1 \leq j \leq m$ . Now, for computing fuzzy beliefs  $n_i$  at place of  $p_i$ , first the condition for transition firing is checked for all  $tr_j$ . If a transition fires then the

fuzzy beliefs of its input places are ANDed and then the result, called FTT is saved. If the transition does not fire, then the FTT corresponding to this transition is set to zero. The fuzzy belief of place  $p_i$  can now be computed by ORing the FTT associated with the  $tr_j$  for  $1 \leq j \leq m$ .

For the sake of brevity, the AND (Minimum of inputs) and OR (maximum of inputs) operators are represented by  $\wedge$  and  $\vee$ , respectively

**Example 14.1:** The parameters defined above are illustrated with reference to Fig.14.1, which describes Production Rule (PR) PR1, where PR1: IF(( $d_1$ ) AND ( $d_2$ )) THEN (( $d_3$ ) OR ( $d_4$ )). Here  $d_1$ =it is hot,  $d_2$ =the sky is cloudy,  $d_3$ =it will rain,  $d_4$ =humidity is high,  $P=\{p_1, p_2, p_3, p_4\}$ ,  $T=\{tr_1\}$ ,  $D=\{d_1, d_2, d_3, d_4\}$ ,  $I(tr_1)=\{p_1, p_2\}$ ,  $O(tr_1)=\{p_3, p_4\}$ . Let  $cf_1=0.9$ ,  $th_1=0.1$ ,  $n_1=0.9$ ,  $n_2=0.5$ , and  $n_3$  and  $n_4$  are to be determined. Now as  $n_1 \wedge n_2 = 0.5 > th_1$ , therefore the transition fires and  $n_3=n_4=[(n_1 \wedge n_2) \wedge cf_1]=0.5$ .



$P=\{p_1, p_2, p_3, p_4\}$ ,  $T=\{tr_1\}$ ,  $D=\{d_1, d_2, d_3, d_4\}$   
 $cf_1=0.9$ ,  $th_1=0.1$ ,  $I(tr_1)=\{p_1, p_2\}$ ,  $O(tr_1)=\{p_3, p_4\}$   
 $n_1=0.9$ ,  $n_2=0.5$   $n_3$  and  $n_4$  are to be determined.

**Fig.14.1:** An FPN representing PR<sub>1</sub>.

**Definition 14.3:** A place with no input arcs is called an *axiom*.

**Definition 14.4:** A path, connected by a set of directed arcs between any two distinct places, in an FPN is called a **reasoning path**. The length of an acyclic single reasoning path between two given places is estimated by counting the number of transitions between the said places on the path. Further, for estimating the length of the reasoning path with cycles, the number of transitions on each cycle is counted only once. If there exists more than one reasoning paths between two places, then the length of the largest path is the measure of the reasoning path length between the said places.

**Definition 14.5:** Any place excluding the starting and terminating place on the reasoning path is called an **intermediate place**.

**Definition 14.6:** If place  $p_i \in I(tr_a)$  and  $p_k \in O(tr_a)$ , then  $p_k$  is called **immediately reachable** from  $p_i$ . The set of places which is immediately reachable from  $p_i$  is called **immediate reachability set** of  $p_i$  and is denoted by  $IRS(p_i)$  [3].

**Definition 14.7:** If  $p_k$  is immediately reachable from  $p_i$  and  $p_j$  is immediately reachable from  $p_k$  then  $p_j$  is **reachable** from  $p_i$ . The reachability relationship is the reflexive and transitive closure of the immediately reachability relationship. The set of places, which is reachable from  $p_i$  is called **the reachability set** of  $p_i$  and is denoted by  $RS(p_i)$  [3].

### 14.2.3 Proposed Algorithm for Belief Propagation

The algorithm of belief propagation to be presented shortly is used for computing fuzzy belief of any propositions, excluding the axioms, in a given acyclic FPN. The proposed algorithm is free from the limitations found in [3], as was pointed out in Section 1. Firstly, it is applicable to acyclic FPN of arbitrary structure. Secondly, the dependence of the fuzzy belief of a proposition on the reasoning path length is eliminated by replacing multiplication operation in [3] by fuzzy AND operation.

One point needs to be emphasized here before the formal presentation of the algorithm. The algorithm for belief propagation cannot be applied to a cyclic FPN because of the requirement of sequential nature of computation from one set of places to their immediate reachable set of places. This statement can be easily proved based on the theorem [6]: “The algorithm of Belief propagation, if applied on a cyclic FPN, does not lead to successful completion because of the nonavailability of fuzzy belief at least one parent of every place selected for belief computation.” However, the problem of belief propagation for a cyclic FPN can be easily tackled by the belief revision algorithm (vide Section 3) with initial zero beliefs at all places excluding the axioms.

The proposed algorithm for belief propagation is comprised of two main procedures, namely “reducenet” and “evaluatenet”. The procedure “reducenet” is required for reducing an FPN of a given structure to a smaller dimension by isolating the reasoning paths, originating at the axioms and terminating at the given conclusion (Goal) place. The procedure “evaluatenet” is used for sequentially computing fuzzy beliefs of all the propagation lying on the reasoning paths of the reduced network.

The procedure “reducenet” works on the following principle. The FPN along with the marked axioms and the concluding proposition (Goal) is submitted to the computer as the input of the procedure “reducenet”. The procedure continues

backtracking, starting from the Goal place until the axioms are reached. The paths thus traced by the procedure are reproduced as the resulting network.

### **Procedure reducenet (FPN, Axioms, Goal, Parents)**

**Begin**

Nonaxioms := Goal ;

**Repeat**

Find-parents (Nonaxioms) ; // Find parents of Nonaxioms. //

Mark the generated parent places, hereafter called Parents and the transitions connected between parents and Nonaxioms ;

Nonaxioms := Parents – Axioms; // Nonaxiom-Parents detection //

**Until** Parents  $\subseteq$  Axioms ;

Trace the marked places and transitions;

**End.**

In the procedure: Evaluanet presented below, “Noncomputed” represents the set of places where fuzzy beliefs have not been computed yet; “Computed” represents the set of places where fuzzy beliefs have been computed, so far; “Unsuccessful-Parents” set represents the set of parents places whose all children do not have fuzzy beliefs yet; “Successful-Children” is the set of places whose all parents do have fuzzy beliefs. The other variables are self-explanatory in the present context.

### **Procedure Evaluatenet (Axioms, Noncomputed)**

**Begin**

Unsuccessful-Parents: =  $\emptyset$  ; Computed: =  $\emptyset$  ;

**For all** q  $\in$  Noncomputed Flag (q): = false ;

B: = Axioms;

**While** Noncomputed  $\neq \emptyset$  **do Begin**

Successful-children: =  $\emptyset$ ;

**For all** p  $\in$  B **do Begin**

Found-Succ.-Children: =  $\emptyset$ ;

Found-Unsucc.-Parents: =  $\emptyset$ ;

**For all** q  $\in$  IRS (p) **do Begin**

**If** (Flag (q)  $\neq$  True)

**Then**

**If** (all parents of q possess fuzzy beliefs)

**Then**

Compute FTT at the transition between p and q for all p  
where IRS (p) = q and compute fuzzy belief at q , following  
Belief propagation model;

Found-Succ.-Children : = Found-Succ.-Children  $\cup$  {q};

Flag (q) := True;

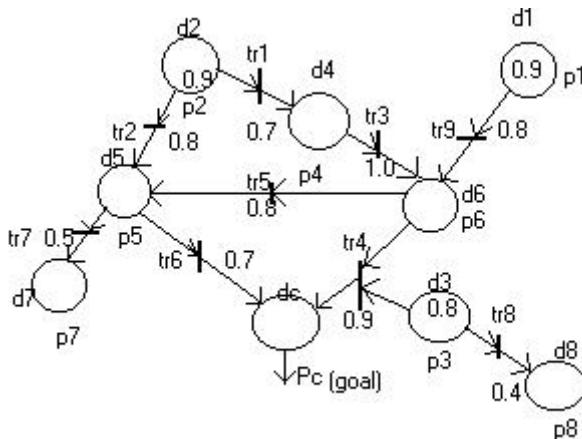
**End If**

**Else** keep record of p in Found-Unsucc.-Parent set;

```

Successful-Children:= Found-Succ.-Children  $\cup$  {q};
Flag (q) := True;
End If
Else keep record of p in Found-Unsucc.-Parent set;
Successful-Children:=Successful-Children  $\cup$  Found-Succ.-Children;
Computed := Computed  $\cup$  Successful-Children;
Noncomputed := Noncomputed - Successful-Children;
Unssuccful-Parents:= Unsuccessful-Parents  $\cup$ 
Found-Unsucc.-Parents;
If (IRS (p)  $\subseteq$  Computed) Then Unsuccessful-Parents:=
Unsuccessful-Parents - {p};
End If;
End If;
End For;
End For;
B:= Successful-Children  $\cup$  Unsuccessful-Parents;
End While;
End.

```



**Fig 14.2:** An FPN used for utilizing evaluetenet procedure, after eliminating places  $p_7$  and  $p_8$  by using reducenet procedure.

Procedure Evaluatenet works on the following principle. For a given parent place  $p \in B$  and a child place  $q \in IRS(p)$ , we need to check whether all the parents of  $q$  possess tokens. If yes, fuzzy belief is computed at  $q$ , and the place  $q$  is declared as a successful child. Similarly, if at least one child of  $p$  is not found to be successful, then  $p$  is called a found-unsuccessful-parent. After generating unsuccessful children from all  $p \in B$ , we re-define  $B$  as the union of Unsuccessful-Parents and Successful-Children. The re-definition of  $B$  ensures completeness of the algorithm. The procedure is terminated when fuzzy beliefs at all possible places in the FPN have been computed.

**Table 14.1:** Results of Execution of Procedure Evaluatenet for FPN of Fig. 14.2

beginning of execution																																																
Unsuccesful-Parents:= $\Phi$ , Computed:= $\Phi$ , B=Axioms={p <sub>1</sub> ,p <sub>2</sub> ,p <sub>3</sub> }, Start of while noncomputed $\neq \Phi$ loop, Successful-Children:= $\Phi$ .																																																
Start of for all p $\in$ loop																																																
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>p</th> <th>Found-Successful Children (FSC)</th> <th>Successful-Computed Children</th> <th>Unsuccesful Parents</th> <th>Noncomputed</th> <th>Fuzzy belief of FSC</th> </tr> </thead> <tbody> <tr> <td>p<sub>3</sub></td> <td>{<math>\Phi</math>}</td> <td>{<math>\Phi</math>}</td> <td>{p<sub>3</sub>}</td> <td>{p<sub>4</sub>, p<sub>5</sub>, p<sub>8</sub>, p<sub>c</sub>}</td> <td>nil</td> </tr> <tr> <td>p<sub>2</sub></td> <td>{p<sub>4</sub>}</td> <td>{p<sub>4</sub>}</td> <td>{p<sub>2</sub>, p<sub>3</sub>}</td> <td>{p<sub>5</sub>, p<sub>6</sub>, p<sub>c</sub>}</td> <td>0.7Λ0.9=0.7</td> </tr> <tr> <td>p<sub>1</sub></td> <td>{p<sub>6</sub>}</td> <td>{p<sub>4</sub>, p<sub>6</sub>}</td> <td>{p<sub>4</sub>, p<sub>6</sub>}</td> <td>{p<sub>2</sub>, p<sub>3</sub>}</td> <td>(0.7Λ1.0)V (0.9Λ0.8)=0.8</td> </tr> </tbody> </table>						p	Found-Successful Children (FSC)	Successful-Computed Children	Unsuccesful Parents	Noncomputed	Fuzzy belief of FSC	p <sub>3</sub>	{ $\Phi$ }	{ $\Phi$ }	{p <sub>3</sub> }	{p <sub>4</sub> , p <sub>5</sub> , p <sub>8</sub> , p <sub>c</sub> }	nil	p <sub>2</sub>	{p <sub>4</sub> }	{p <sub>4</sub> }	{p <sub>2</sub> , p <sub>3</sub> }	{p <sub>5</sub> , p <sub>6</sub> , p <sub>c</sub> }	0.7Λ0.9=0.7	p <sub>1</sub>	{p <sub>6</sub> }	{p <sub>4</sub> , p <sub>6</sub> }	{p <sub>4</sub> , p <sub>6</sub> }	{p <sub>2</sub> , p <sub>3</sub> }	(0.7Λ1.0)V (0.9Λ0.8)=0.8																			
p	Found-Successful Children (FSC)	Successful-Computed Children	Unsuccesful Parents	Noncomputed	Fuzzy belief of FSC																																											
p <sub>3</sub>	{ $\Phi$ }	{ $\Phi$ }	{p <sub>3</sub> }	{p <sub>4</sub> , p <sub>5</sub> , p <sub>8</sub> , p <sub>c</sub> }	nil																																											
p <sub>2</sub>	{p <sub>4</sub> }	{p <sub>4</sub> }	{p <sub>2</sub> , p <sub>3</sub> }	{p <sub>5</sub> , p <sub>6</sub> , p <sub>c</sub> }	0.7Λ0.9=0.7																																											
p <sub>1</sub>	{p <sub>6</sub> }	{p <sub>4</sub> , p <sub>6</sub> }	{p <sub>4</sub> , p <sub>6</sub> }	{p <sub>2</sub> , p <sub>3</sub> }	(0.7Λ1.0)V (0.9Λ0.8)=0.8																																											
end of for all p $\in$ B loop, B= { p <sub>2</sub> ,p <sub>3</sub> ,p <sub>4</sub> ,p <sub>6</sub> }																																																
Noncomputed $\neq \Phi$ , so While Noncomputed $\neq \Phi$ loop is continued																																																
Successful-Children := $\Phi$ . Start of for all p $\in$ loop																																																
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>P</th> <th>Found-Succesful-Children (FSC)</th> <th>Successful-Children</th> <th>Computed</th> <th>Unsuccesful-Parents</th> <th>Noncomputed Fuzzy belief of FSC</th> </tr> </thead> <tbody> <tr> <td>P<sub>2</sub></td> <td>{p<sub>5</sub>}</td> <td>{p<sub>5</sub>}</td> <td>{p<sub>4</sub>, p<sub>5</sub>, p<sub>6</sub>}</td> <td>{p<sub>3</sub>}</td> <td>{p<sub>c</sub>} (0.9Λ0.8)V (0.8Λ0.8)=0.8</td> </tr> <tr> <td>P<sub>3</sub></td> <td>{p<sub>c</sub>}</td> <td>{p<sub>5</sub>, p<sub>c</sub>}</td> <td>{p<sub>4</sub>, p<sub>5</sub>, p<sub>6</sub>, p<sub>c</sub>}</td> <td>{<math>\Phi</math>}</td> <td>{<math>\Phi</math>} (0.8Λ0.7)V {(0.9)Δ(0.8Λ0.8)} = 0.8</td> </tr> <tr> <td colspan="6" style="text-align: center;"><math>p_4 \leftarrow</math> Nil since flag (q) =1 for all q <math>\in</math> IRS(p<sub>4</sub>) <math>\rightarrow</math></td></tr> <tr> <td colspan="6" style="text-align: center;"><math>p_6 \leftarrow</math> Nil since flag (q) =1 for all q <math>\in</math> IRS(p<sub>6</sub>) <math>\rightarrow</math></td></tr> <tr> <td colspan="6" style="text-align: center;">end of for all p <math>\in</math> B loop, B= {p<sub>5</sub>,p<sub>c</sub>}</td></tr> <tr> <td colspan="6" style="text-align: center;">Since Noncomputed = <math>\emptyset</math>, While loop is terminated.</td></tr> <tr> <td colspan="6" style="text-align: center;">End of execution</td></tr> </tbody> </table>	P	Found-Succesful-Children (FSC)	Successful-Children	Computed	Unsuccesful-Parents	Noncomputed Fuzzy belief of FSC	P <sub>2</sub>	{p <sub>5</sub> }	{p <sub>5</sub> }	{p <sub>4</sub> , p <sub>5</sub> , p <sub>6</sub> }	{p <sub>3</sub> }	{p <sub>c</sub> } (0.9Λ0.8)V (0.8Λ0.8)=0.8	P <sub>3</sub>	{p <sub>c</sub> }	{p <sub>5</sub> , p <sub>c</sub> }	{p <sub>4</sub> , p <sub>5</sub> , p <sub>6</sub> , p <sub>c</sub> }	{ $\Phi$ }	{ $\Phi$ } (0.8Λ0.7)V {(0.9)Δ(0.8Λ0.8)} = 0.8	$p_4 \leftarrow$ Nil since flag (q) =1 for all q $\in$ IRS(p <sub>4</sub> ) $\rightarrow$						$p_6 \leftarrow$ Nil since flag (q) =1 for all q $\in$ IRS(p <sub>6</sub> ) $\rightarrow$						end of for all p $\in$ B loop, B= {p <sub>5</sub> ,p <sub>c</sub> }						Since Noncomputed = $\emptyset$ , While loop is terminated.						End of execution					
P	Found-Succesful-Children (FSC)	Successful-Children	Computed	Unsuccesful-Parents	Noncomputed Fuzzy belief of FSC																																											
P <sub>2</sub>	{p <sub>5</sub> }	{p <sub>5</sub> }	{p <sub>4</sub> , p <sub>5</sub> , p <sub>6</sub> }	{p <sub>3</sub> }	{p <sub>c</sub> } (0.9Λ0.8)V (0.8Λ0.8)=0.8																																											
P <sub>3</sub>	{p <sub>c</sub> }	{p <sub>5</sub> , p <sub>c</sub> }	{p <sub>4</sub> , p <sub>5</sub> , p <sub>6</sub> , p <sub>c</sub> }	{ $\Phi$ }	{ $\Phi$ } (0.8Λ0.7)V {(0.9)Δ(0.8Λ0.8)} = 0.8																																											
$p_4 \leftarrow$ Nil since flag (q) =1 for all q $\in$ IRS(p <sub>4</sub> ) $\rightarrow$																																																
$p_6 \leftarrow$ Nil since flag (q) =1 for all q $\in$ IRS(p <sub>6</sub> ) $\rightarrow$																																																
end of for all p $\in$ B loop, B= {p <sub>5</sub> ,p <sub>c</sub> }																																																
Since Noncomputed = $\emptyset$ , While loop is terminated.																																																
End of execution																																																

**Example 14.2:** The procedure evaluenet has been executed for the FPN given in Fig. 14.2, where the fuzzy beliefs of the axioms and CF of transitions are shown in the figure, and thresholds of all transitions are assumed to be 0.2. The results obtained at different steps of the procedure are given in Table 14.1.

**Time-Complexity:** Time-complexity of the reducenet algorithm in the worst case is  $a \times M$ , where  $a$  and  $M$ , respectively, represent the number of axioms and places in the FPN before reduction. The evaluenet algorithm can be implemented on a uniprocessor as well as multiprocessor architecture with  $(M + N)$  number of processors corresponding to  $M$  places and  $N$  transitions. The worst time-complexity of the uniprocessor and multiprocessor-based evaluenet algorithm are  $3M^2$  and  $(2M^2/ pr)$ , respectively, where  $pr$  represents the minimum number of places or transition where computation can take place in parallel and  $M \geq N$ .

**Theorem 14.1:** *In an FPN any place which itself is not an axiom, is always reachable from the set of axioms.*

*Proof:* Let the contradiction of the statement be true. Thus if  $p$  be a place such that  $p \notin$  set of axioms  $A$ , then  $p \notin RS(\alpha)$  where  $\alpha \in A$ . Now, since  $p \notin RS(\alpha)$  then none of the parents of  $p$  belong to  $RS(\alpha)$  i.e.,  $q \notin RS(\alpha)$  where  $IRS(q) = p$ . Analogously,  $r \notin RS(\alpha)$  where  $IRS(r) = q$ . Thus backtracking in the FPN, starting from  $p$  without traversing the same place more than once (for avoiding traversal on the cyclic path more than once), we ultimately find some place  $z$  such that

$$p = IRS(IIRS(IIRS\dots(IIRS(Z))))$$

where  $z$  has no input arcs.

Therefore, by definition  $Z$  is an axiom. So, the initial assumption that  $p \notin RS(Z)$  where  $Z \in A$  is violated. Hence, the theorem is proved.

**Corollary 14.1:** *Any place  $p$  with more than one input arcs is reachable from one or more axioms, through all the reasoning paths terminating at  $p$ .*

**Theorem 14.2:** *If there is no cycle in a FPN, then there should exist at least one place immediately reachable from any of the axioms such that all its parents belong to the set of axioms.*

*Proof:* The theorem will be proved by the method of contradiction. So, let us assume that, there does not exist even a single place, immediately reachable from any of the axioms, such that all its parents belong to the set of axioms in a FPN without cycles.

Let  $S = \{u, v, w, x\}$  be the set of places immediately reachable from the set of axioms, where each element of  $S$  has more than one parents (since with one parent the proof is obvious). Now, according to the initial assumptions, place  $u$  cannot have all its parents in the set of axioms. Moreover, since place  $u$  is reachable from axioms through all the reasoning paths terminating at place  $u$  (vide corollary 14.1) and  $S$  is the exhaustive set of places immediately reachable from the axioms, therefore,  $u$  is reachable from at least one place in  $S$ . In an analogous manner, it can be proved that any place  $p \in S$  is reachable from some other place  $p' \in S$ . This proves the existence of at least one cycle in the network (passing through elements of  $S$ ) which, however, is against the initial assumption. Therefore, the initial assumption is wrong and the statement of the theorem is true.

Two important properties of the belief propagation algorithm are now presented below in Theorems 14.3 and 14.4.

**Theorem 14.3:** *Any place for which fuzzy belief has not been computed yet is reachable from place  $b \in B$ .*

*Proof:* For  $B =$  the set of axioms  $A$ , the statement of the theorem follows directly from theorem 14.1. However, for  $B \neq A$ , let the converse of the statement be true, i.e., for any place  $p$  for which fuzzy belief has not been computed, let  $p \notin RS(b)$  for  $b \in B$ . Now, since  $p$  is reachable from places belonging to  $A$ , vide theorem 14.1, therefore,  $p$  is reachable from set  $X$  where the fuzzy belief of each place  $x \in X$  has already been computed. Now, following, the procedure “evaluenet” it is clear that  $B \subset X$ . So,  $p$  which is not reachable from the places in  $B$  is reachable from places in  $X - B$ . Now, since  $p$  is reachable only from places in  $X - B$  (and not from places in  $B$ ), therefore, there should exist at least one place  $p'$  such that  $p \in RS(p')$  and all the parents of  $p'$  belong to  $X - B$ . Further, since all the parents of  $p'$  possess fuzzy beliefs, therefore fuzzy belief at  $p'$  should be computed currently. However, from the algorithm it is clear that set  $B$  always keeps track of the parents corresponding to the children where fuzzy beliefs have to be computed currently. So, there cannot be any place  $p'$  whose fuzzy belief is yet to be computed with all its parents in the set  $(X - B)$ . Consequently, there cannot be any place  $p$ , which satisfies the initial assumption. Therefore, the initial assumption is wrong and the statement of the theorem is true.

**Definition 14.8:** *The algorithm for belief propagation is said to reach a state of deadlock if for a given set  $B$ , no  $q$  is readily identified for belief computation prior to termination of the algorithm (i.e.,  $Noncomputed \neq \emptyset$ ).*

**Theorem 14.4:** *The computation process in the algorithm for belief propagation never reaches a state of deadlock.*

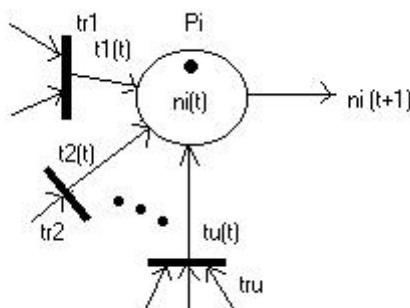
*Proof:* When  $B = A$  = the axioms, we always get a place  $q$  ready for belief computation (vide Theorem 14.2). When  $B = A$ , it is clear from theorem 14.3 that the places having no fuzzy beliefs in the FPN are reachable from places belonging to  $B$ . So, set  $B$  can be treated like set of axioms  $A$ , from the point of view of reachability of the places having no fuzzy beliefs. Therefore, the statement of the theorem follows from the statement of Theorem 14.2. The theorem can be formally proved by the method of induction.

### 14.3 Imprecision and Inconsistency Management in a Cycle FPN

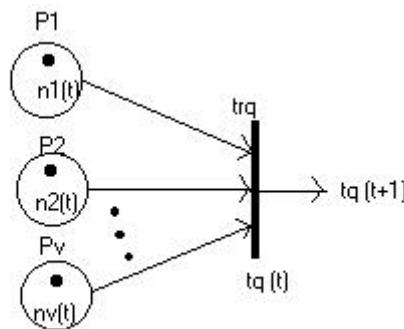
This section presents a new technique for imprecision and inconsistency management in a cyclic FPN using a model for belief-revision, based on the central concept of fuzzy reasoning presented in [11]. The basis of reasoning of this model is humanlike and is different from the existing probabilistic models [12], [17] for belief-revision.

#### 14.3.1 Proposed Model for Belief-Revision

Let  $n_i(t)$  be the fuzzy beliefs of a place  $p_i$  and  $t_q(t)$  be the fuzzy truth token (FTT) associated with the transition  $tr_q$  at time  $t$ . Let  $p_i$  be one of the common output places of transition  $tr_q$  for  $1 \leq k \leq u$  (Fig.14.3) and  $p_j$ ,  $1 \leq j \leq v$  are the input places of transition  $tr_k$  (Fig14.4). The updating of fuzzy values at transition  $tr_q$  and place  $p_i$  can then be represented by (14.1) and (14.2) respectively.



**Fig 14.3:** An FPN exhibiting connectivity from transitions to a place.



**Fig 14.4:** An FPN exhibiting connectivity from places to transitions.

Let  $x = \wedge \{n_j(t)\}$ . Now if  $x >$  threshold marked at  $tr_q$

$$1 \leq j \leq v$$

then the transition fires with the resulting FTT as given by

$$t_q(t+1) = x \wedge cf_q \quad (14.1)$$

where  $cf_q$  is the time invariant CF of the transition, else  $t_q(t+1) = 0$ .

If  $p_i$  is an axiom then

$$n_i(t+1) = n_i(t), \quad (14.2)$$

$$\text{else } n_i(t+1) = \vee t_k(t+1) \quad (14.3)$$

$$1 \leq k \leq u$$

Initial fuzzy beliefs are assigned at all places of a given FPN. Therefore, by definition of enabling of transition, all the transition in the FPN may be enabled simultaneously. The conditions of transition firing are, therefore, checked and the temporal FTT are computed concurrently at all transition using (14.1). Then fuzzy beliefs at all the places in the FPN can be revised in parallel by using (14.2) and (14.3) as is appropriate. This is regarded as one complete belief revision step. For computing fuzzy beliefs at all places up to the  $m$ th step, one should repeat the belief-revision process  $m$  times recursively.

### 14.3.2 Stability Analysis of the Belief-Revision Model

The Definitions of a few more terms, which will be referred to frequently in the rest of the chapter, are in order.

**Definition 14.9:** An arc is called **dominant** if it carries the highest CF among all the arcs incoming to a place or the least fuzzy beliefs (LFB) does not exceed the threshold of the transition.

**Definition 14.10:** An arc is called **permanently dominant** if it remains dominant for an infinite number of belief revision steps.

**Definition 14.11:** An FPN is said to have reached **steady-state** (or equilibrium condition) when the condition:  $n_i(t^*+1)=n_i(t^*)$  is satisfied for each place  $p_i$  in the FPN and the minimum value of  $t = t^*$  is called the **equilibrium time**.

**Definition 14.12:** An FPN is said to have periodic oscillation (PO) if the fuzzy belief of each place on any of the cycles in the FPN is repeated after a definite number of belief-revision steps. In case the PO sustains for an infinite number of belief-revision steps, we say **limitcycles (LC)** exist at each place on the cycle.

The properties of the belief-revision model, which have been obtained analytically are listed in Theorems 14.5 through 14.7.

**Theorem 14.5:** If all the arcs on a cycle with  $n$  transition remain dominant from  $r_1$ th to  $r_2$ th belief-revision steps, then the fuzzy beliefs at each place on the cycle would exhibit i) at least ‘ $a$ ’ number of periodic oscillations where  $a = \lfloor (r_2 - r_1)/n \rfloor$  and ii) limitcycles if  $r_2$  approaches infinity.

*Proof:* If all the arcs on a cycle become dominant at the  $r_1$ th belief-revision step, then the fuzzy beliefs at each place on the cycle will be transferred to its immediate reachable place on the cycle. Now, if all the arcs remain dominant until the  $r_2$ th belief-revision step, then transfer of belief from one place to its immediately reachable place on the cycle would continue for  $(r_2 - r_1)$  times. Let us now assume without any loss of generality, that

$$r_2 - r_1 = a.n + b$$

where  $n$  is the number of transition on the cycle under consideration,  $b \leq n - 1$ , and  $a$  and  $b$  are nonnegative integers such that both of these cannot be zero simultaneously. It is clear from (14.3) that the fuzzy beliefs at each place on the cycle would retrieve the same value after every  $n$  number of belief-revision steps. So, the fuzzy beliefs at each place on the cycle would exhibit at least a number of periodic oscillations, where  $a = \lfloor (r_2 - r_1) / n \rfloor$  as  $b \leq n - 1$ . This proves the first part of the theorem.

Now, to prove the second part, we substitute the limiting value of  $r_2$  to infinity in the expression for  $a$ , which yields the value of  $a$  to be infinity, proving the existence of limitcycles at each place on the cycle.

The following two Definitions are useful to explain Theorem 14.6.

**Definition 14.13:** *The fuzzy multiplication (max-min composition) of an  $(m \times n)$  matrix by an  $(n \times 1)$  column vector is carried out in the same way as in conventional matrix algebra with substitution of multiplication and summation operators by fuzzy AND (Min) and OR (Max) operators, respectively.*

**Definition 14.14:** *The complementation of a fuzzy vector is obtained by replacing elements of the vector by their one's complements (i.e., one minus the value of the element).*

**Definition 14.15:** *The fuzzy AND operations of two vectors is carried out in the same way as in case of addition of two vectors in conventional Matrix Algebra with the substitution of addition by fuzzy AND operation.*

**Definition 14.16:** *A matrix  $P$  is called binary transition to place connectivity (TPC) matrix if its element  $p_{ij}$  is 1, when  $p_i \in O(tr_j)$  and 0, otherwise. A matrix  $Q$  is called binary place to transition connectivity (PTC) matrix if its element  $q_{ij} = 1$  if  $p_j = I(tr_i)$  and 0 otherwise.*

**Definition 14.17:** *A belief vector  $N(t)$  is a column vector, the  $i$ th component of which represents the fuzzy beliefs  $n_i(t)$  of place  $p_i$  at time  $t$ . The belief vector at  $t = t^*$ , the equilibrium time is called the steady-state belief vector (SSBV).*

**Definition 14.18:** *A CF vector represents a time-invariant column vector, the  $i$ th component of which represents the CF of the transition  $tr_i$  in an FPN.*

**Theorem 14.6:** *If the transitions remain enabled at each belief revision step, then the SSBV  $N^*$  satisfies the following expression, where  $P$  and  $Q$  are TPC and PTC matrices respectively, CF is the certainty factor vector,  $c$  over a vector represents its fuzzy complementation operation, and the  $o$  denotes the fuzzy AND-OR composition operator.*

$$N^* = P o [(Q o (N^*)^c)^c \wedge CF]$$

*Proof:* Assume that there exist  $m$  places and  $n$  transition in the FPN. Now, since transitions are always enabled, therefore, by (14.1), FTT of transition  $tr_i$ , at time  $(t + 1)$  is given by

$$t_i(t+1) = \{ \bigwedge_{1 \leq j \leq m} (n_j(t)) \} \wedge cf_i$$

$$= \{ V \bigwedge_{1 \leq j \leq m} (n_j^c(t))^c \wedge cf_i$$

$$= \{ \bigvee_{1 \leq j \leq m} (q_{ij} \wedge n_j^c(t))^c \wedge cf_i \} \quad (14.4)$$

Thus for  $1 \leq i \leq n$ , we get a column vector  $T(t+1)$ , the  $i$ th component of which is  $t_i(t+1)$ . Here,

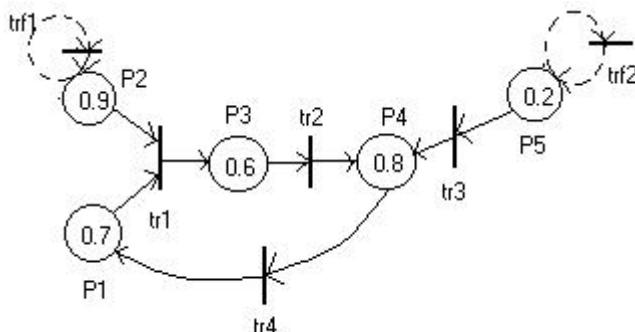
$$T(t+1) = (Q \circ N^c(t))^c \wedge CF. \quad (14.5)$$

Now, in order to represent belief-revision at both axioms and nonaxioms by a single expression like (14.3), we construct, without any loss of generality, a self-loop around each axiom through a transition with threshold = 0 and CF = 1. Thus, one can easily use expression (14.3) for computing fuzzy beliefs at any place at time  $(t + 1)$ . It can now be easily proved that

$$N(t+1) = P \circ T(t+1). \quad (14.6)$$

Now, combining (14.5) and (14.6) and substituting  $t=t^*$ , the equilibrium time, in the derived expression, the theorem can be proved.

**Example 14.3:** Assuming the thresholds and CF associated with all the transition to be zero and one, respectively, the computation of belief vector  $N(1)$  and  $N(2)$  are illustrated in this example with reference to Fig. 14.5. First, virtual transitions are included in the self-loops, represented by the dotted line-segments in Fig. 14.5, constructed around each axiom. Then the matrices  $P$ ,  $Q$ , and vector  $N(0)$  are obtained from Fig. 14.5 as given below.



**Fig.14.5:** An FPN having virtual transitions, constructed around the axioms, for illustrating the formation of  $P$  and  $Q$  matrices.

## From Transitions

$$P = \begin{array}{c|cccccc} \text{To Places} & tr_1 & tr_2 & tr_3 & tr_4 & tr_{f1} & tr_{f2} \\ \hline p_1 & 0 & 0 & 0 & 1 & 0 & 0 \\ p_2 & 0 & 0 & 0 & 0 & 1 & 0 \\ p_3 & 1 & 0 & 0 & 0 & 0 & 0 \\ p_4 & 0 & 1 & 1 & 0 & 0 & 0 \\ p_5 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

## From Places

$$Q = \begin{array}{c|ccccc} \text{To Trans.} & p_1 & p_2 & p_3 & p_4 & p_5 \\ \hline tr_1 & 1 & 1 & 0 & 0 & 0 \\ tr_2 & 0 & 0 & 1 & 0 & 0 \\ tr_3 & 0 & 0 & 0 & 0 & 1 \\ tr_4 & 0 & 0 & 0 & 1 & 0 \\ tr_{f1} & 0 & 1 & 0 & 0 & 0 \\ tr_{f2} & 0 & 0 & 0 & 0 & 1 \end{array}$$

and  $N(0) = [0.7 \ 0.9 \ 0.6 \ 0.8 \ 0.2]^T$ , where  $T$  denotes the transposition operator over the row vector.

Combining (14.5) and (14.6) and substituting  $t = 0$  and  $t = 1$  in the resulting expression, we find

$$N(1) = P \circ (Q \circ N^c(0))^c = [0.8 \ 0.9 \ 0.7 \ 0.6 \ 0.2]^T,$$

$$N(2) = P \circ (Q \circ N^c(1))^c = [0.6 \ 0.9 \ 0.8 \ 0.7 \ 0.2]^T.$$

In case steady state is reached in an FPN, the number of belief-revision steps required to reach steady state is computed below:

Let

$A = \text{set of axioms},$

$\text{Cycle}_i = \text{set of places on cycle } i,$

$N = \text{set of cycles},$

$(l_{ja})_i = \text{minimum reasoning path length from a given axiom } a \text{ to a given place } j \text{ on cycle } i,$

$(l_1)_i = \text{Max} \{ \text{Max} (l_{ja})_i \},$

$a \in A \quad j \in \text{Cycle}_i$

$l_{\max 1} = \text{Max}(l_1)_i,$

$i \in N$

$n = \text{number of transitions on the largest cycle},$

$(l_{jq})_i = \text{minimum reasoning path length from a given place } j \text{ on cycle } i \text{ to a given terminal place } q,$

$(l_2)_i = \text{Max} \{ \text{Max} (l_{jq})_i \}, \text{ where } Y = \text{set of terminals},$

$q \in Y \quad j \in \text{Cycle}_i$

$l_{\max 2} = \text{Max}(l_2)_i$

$i \in N$

$i = \text{any cycle},$

$l_{aq} = \text{minimum reasoning path length from an axiom } a \text{ to a given terminal place } q \text{ without touching any cycle},$

$l_3 = \text{Max} \{ \text{Max} (l_{aq}) \}$

$a \in A \quad q \in Y$

**Theorem 14.7:** In case steady-state condition can be reached in an FPN, then the number of belief revisions required to reach steady-state in the worst case ( $T_{SS}$ ) is given by

$$T_{SS} = \text{Max} \{ l_3, (l_{\max 1} + l_{\max 2} + n - 1) \}.$$

*Proof:* It can be easily shown [8] that  $T_{SS}$  is equal to the belief revision steps needed for the transfer of fuzzy beliefs from the set of axioms to all the terminal places of the FPN. So,  $T_{SS}$  is equal to the number of belief revision required to traverse the largest reasoning path length between the axioms and the terminal places. In case the largest reasoning path does not pass through any cycle, then  $T_{SS} = l_3$ . On the other hand, if it passes through cycles, then we need to compute three terms:

- 1) belief revision steps required for the fuzzy beliefs of the axioms to reach all possible places on all the cycles,

- 2) belief revision steps required for the transfer of fuzzy beliefs of axioms from a few places on the largest cycle to all places on the same cycle, and
- 3) the maximum number of belief revision required for the propagation of beliefs from the places on the cycles to all the terminal places.
- 1) and 3) have already been estimated. Now to compute 2, we use one property, which follows from Theorem 14.5, that for reaching steady-state in an FPN, at least one arc on each cycle should be permanently nondominant. Thus for the largest cycle with  $n$  transitions, belief revision required in 2) in the worst case =  $(n - 1)$ . So adding 1), 2) and 3), we get  $T_{SS}$  for the largest reasoning path that passes through cycle. Keeping option for the largest reasoning path either to pass or bypass cycles, we get the result presented in the theorem.

**Corollary 14.2:** *The number of belief revisions required to reach steady-state is always  $\leq$  number of transitions in the FPN .*

### 14.3.3 Detection and Elimination of Limitcycles

It follows from Corollary 14.2 that, in case limitcycles sustain in an FPN, then the steady-state condition will not be reached even after  $z$  number of belief revisions, where  $z$  denotes the number transition in the FPN. Procedure Belief revision has been developed based on this concept

- 1) either to detect existence of limitcycles or
- 2) to determine the steady state values of fuzzy beliefs in an FPN in absence of LC .

**Procedure Belief-revision (FPN, Initial beliefs, thresholds, total-no-of-trans)**

**Begin**

Belief-revision-step-count : = 0 ;

**Repeat**

- 1) Update FTT of all transitions in parallel by using (14.1) ;
- 2) Update fuzzy beliefs of all places in parallel by using (14.2) or (14.3), whichever is appropriate;
- 3) Increment Belief-revision-step-count by one;

**Until**

**For** each place in the FPN

(Current-value-of-fuzzy-belief = last-value-of-fuzzy-belief) OR  
(belief-revision-step-count = total-no-of-trans + 1)

**End For;**

**If** (belief-revision-step-count < total-no-of-trans)

**Then** print (current value of fuzzy beliefs of all places)

```

End If
Else print ( “limit cycle exists.”)
End.

```

It may be mentioned here than an alternative belief-revision algorithm in vector-matrix form can be easily developed using (14.5) and (14.6).

Belief-revision algorithm too can be implemented on a uniprocessor as well as multiprocessor architecture with  $(M + N)$  number of processing elements located at  $M$  places and  $N$  transition, respectively. Computational time needed for sequential and parallel belief-revision algorithm in the worst case are of the order of  $2M^3$  and  $2M^2$  respectively.

Now, before describing the method of elimination of LC, we define fuzzy gain.

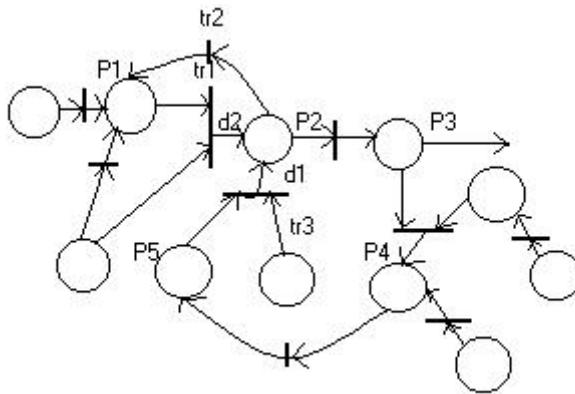
**Definition 14.19:** *Fuzzy gain of an acyclic single reasoning path between two places is computed by taking the minimum of the fuzzy belief of the places and the CF of the transition on that path. In case there exists parallel reasoning paths between two places, then the overall fuzzy gain of the path is defined as the maximum of the individual path gains. Fuzzy gain of a reasoning path, starting from axioms up to a given transition is defined as the maximum of the fuzzy gains from each axiom up to each of the input places of the transition.*

Once the existence of LC is detected in a given FPN, its elimination becomes imperative for arriving at any conclusion. Elimination of LC, however, implies the elimination of permanent dominance of at least one arc of the cycle, containing the places, which exhibit LC behavior. This can be achieved by permanently preventing one of the transitions on the cycle from firing by setting an appropriate threshold at the transition. One possible choice of threshold is the largest fuzzy beliefs on the reasoning path from the axioms to the selected transition.

Questions may, however, be raised: can a user adjust the thresholds? The answer to this is affirmative if the user can ensure that the inferences derived by the system are least affected by threshold adjustment. Again, it is clear from the discussions in [7-8] that the reasoning path with the least fuzzy gain is ignored in most cases, if there exists multiple reasoning paths leading to the concluding place. It is, therefore, apparent that in order to eliminate LC, one should select a transition that lies on the path with the least fuzzy gain.

However, adjustment of threshold at a transition on the cycle with least fuzzy gain may sometimes induce LC into a neighborhood cycle. This is due to the fact that a transition selected for threshold adjustment may cause permanent dominance of the output arc of another transition, both of which have a common

output place, shared by two neighborhood cycles. For example, selecting transition  $tr_1$  in cycle  $p_1 p_2 p_1$  (Fig.14.6) causes permanent dominance of the arc  $d_1$  on cycle  $p_2 p_3 p_4 p_5 p_2$  where  $p_2$  is the common output place of  $tr_1$  and  $tr_3$ . In case all the arcs excluding  $d_1$  on the cycle  $p_2 p_3 p_4 p_5 p_2$  were permanently dominant, then permanent dominance of  $d_1$  would cause LC to be introduced in the cycle.



**Fig.14.6:** An FPN for illustrating the possibility of induction of LC into neighborhood cycle.

To select the right transition on the cycle, one has to satisfy two criteria jointly:

- 1) The transition to be selected should lie on the reasoning path with the lowest possible fuzzy gain, as far as practicable and
- 2) The selected transition should not induce LC into neighborhood cycles.

To satisfy these two criteria jointly, we compute the threshold of all the transition on the cycle, based on the guideline presented above and construct a set of ordered pairs (of transition and fuzzy gain of the path containing the transition), such that the elements of the set are sorted in ascending order using fuzzy gain as the key field. Thus, the former the position of the transition–fuzzy gain pair, smaller is the fuzzy gain of the path containing the transition. The transition-fuzzy gain pair, which occupies the first position in the set, is picked up, the threshold is assigned at the transition following the method discussed earlier, and the existence of LC is checked in the neighboring cycle. If LC does not exist in the neighborhood cycle, then the LC-elimination process for the proposed cycle is terminated. In case LC is induced in the neighborhood cycle, the transition from the next ordered pair is picked up and the possibility of induction of LC in the neighborhood cycle is examined again. The process is thus continued for each transition in the set according to the position of its corresponding ordered pair in the set, until a suitable one, which does not cause

induction of LC in the neighborhood cycle is detected. If no such transition is found in the set, the LC from the proposed cycle cannot be eliminated and the same is informed to the user.

The above process is repeated for all cycles exhibit in LC behavior.

#### 14.3.4 Nonmonotonic Reasoning in an FPN

Contradictory pairs of propositions, such as a proposition ad its negation, or sets of mutually inconsistent propositions, such as the number of assailants in a criminology case as obtained from different sources, often creep into an FPN by several ways [7]. One simple way of entry of inconsistency into FPN is due to the presence of inconsistent propositions in the DB, which being matched with the antecedent part of PR in the KB, finally appear in the FPN during its formation [8]. For example, consider the rules:  $A \rightarrow B$ ,  $\neg A \rightarrow C$  in a KB and the data  $A$  and  $\neg A$  in the DB. In such a case, both  $A$  and  $\neg A$  appear in the FPN during its formation. An alternative way of entry of inconsistency into an FPN is due to the presence of inconsistent predicates in the antecedent part of one rule and the consequent part of another rule, together with appropriate data elements in the DB. For example, with the rules:  $A \rightarrow B$ ,  $B \text{ AND } C \rightarrow \neg A$ , and the data  $A$ ,  $B$ , and  $C$  in the DB, both  $A$  and  $\neg A$  appear in the FPN. The other ways [7] of entry of inconsistency into an FPN could not be discussed here for lack of space. Irrespective of the ways of entry of inconsistency into an FPN, we present here general algorithm for accepting only one proposition from each set of inconsistent propositions through a global voting in the FPN. The proposition, which receives majority support by all other propositions in the FPN will be selected for subsequent reasoning, while its contradictions are eliminated from the reasoning space. Care should be taken that the inconsistent evidences do not influence the voting process itself. The belief-revision algorithm is used to implement the voting process. The algorithm for inconsistency management below is designed based on the above principles. It consists of 5 main steps.

- 1) Open the output arcs of all the mutually inconsistency sets of propositions, so that the voting process is not influenced by their presence in the FPN.
- 2) Detect LC, when present, and eliminate it, if possible. If elimination is not possible, report the same to the user and exit.
- 3) Revise belief in the FPN until steady state is reached. After steady-state is reached, find for each set of inconsistent propositions the one with the highest steady belief. Ignore the other propositions in each set of inconsistent evidences by setting their belief to zero permanently, hereafter called *grounding*.

- 4) Open the input arc of all the grounded propositions in the FPN, so that their fuzzy beliefs are not changed. Also, reconnect the arcs opened in step 1.
- 5) Return the initial fuzzy beliefs to all, excluding the grounded places. Also return the initial threshold to each transition. Exit.

It may be noted that contradiction management for all sets of inconsistent propositions can be undertaken in parallel in the above algorithm. Further, since the results produced by the above algorithm are highly data-dependent, for better results the initial fuzzy beliefs should be normalized when the characteristics [12], say authenticity level, of the sources are known (Fig.14. 9).

## 14.4 Conclusions

The chapter introduced a new unified method for reasoning in ES in presence of imprecision and inconsistency of data and uncertainty of knowledge by using a structured fuzzy IE called FPN. In the chapter, special emphasis has been given to reasoning in ES with a self-referential [15] KB, such as  $A, B \rightarrow C$ , and  $C \rightarrow A$ , which in the presence of appropriate data elements, leads to introduction of cycles in an FPN. In a self-referential ES with imprecise DB and uncertain KB, the temporal fuzzy beliefs for the propositions, corresponding to the places on the cycle, sometimes exhibit LC behavior. In order to derive stable inferences for such systems, a new technique for detection and elimination of LC has been presented in the chapter. The principles of inconsistency management have also been presented for self-referential expert systems.

The proposed technique was successfully applied [8] in a simulated ES (Fig. 14.7) for criminology problems. Before reasoning is started in such a system, the imprecision and inconsistency of database and the uncertainty of knowledge base together are mapped onto the FPN (Fig. 14.10) [8]. During the reasoning phase, the three main steps, namely

- 1) elimination of inconsistency,
- 2) management of imprecision and uncertainty, and
- 3) resolving conflict among competitive conclusion and tracing of explanation

are performed in order. Time-complexity for reasoning in such self-referential ES in the worst case is  $O(M^3)$ , where  $M$  denotes the total number of places in the FPN .

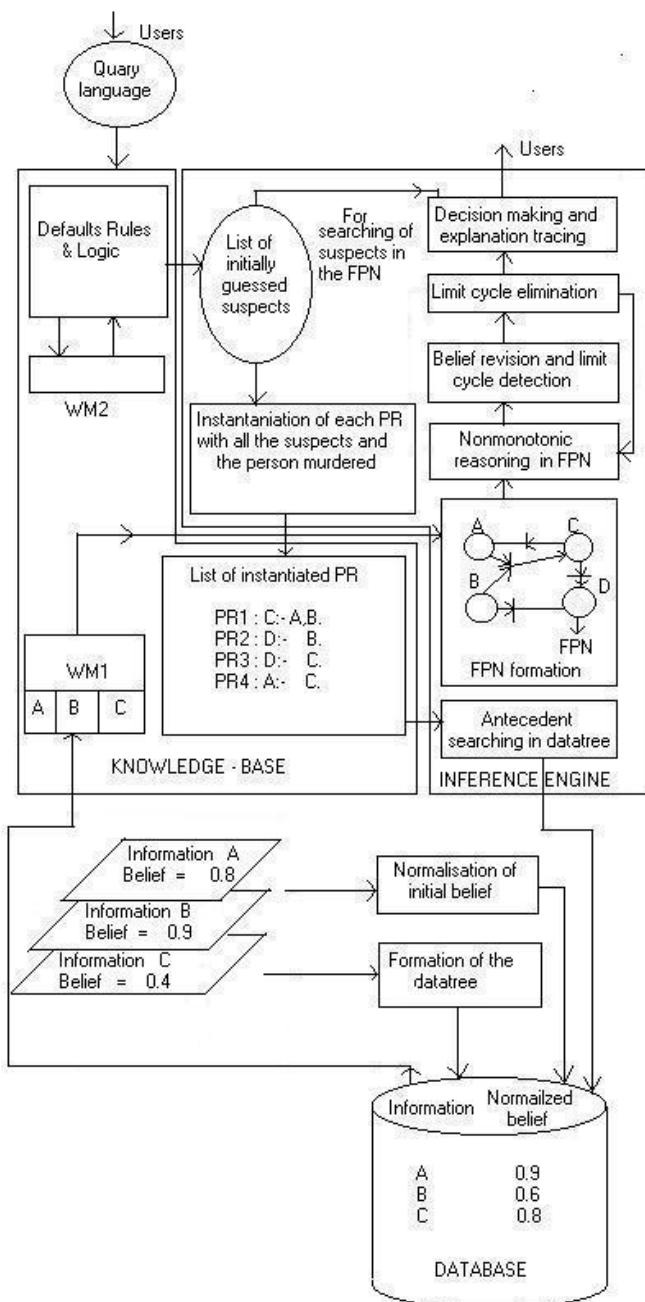
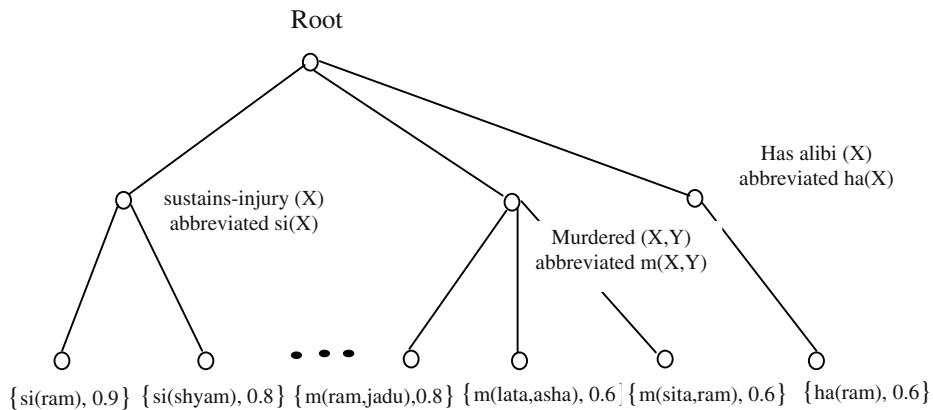
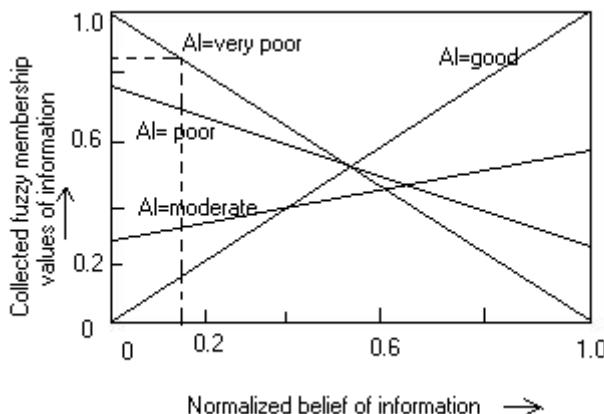


FIG 14.7: The schematic architecture of the overall system.



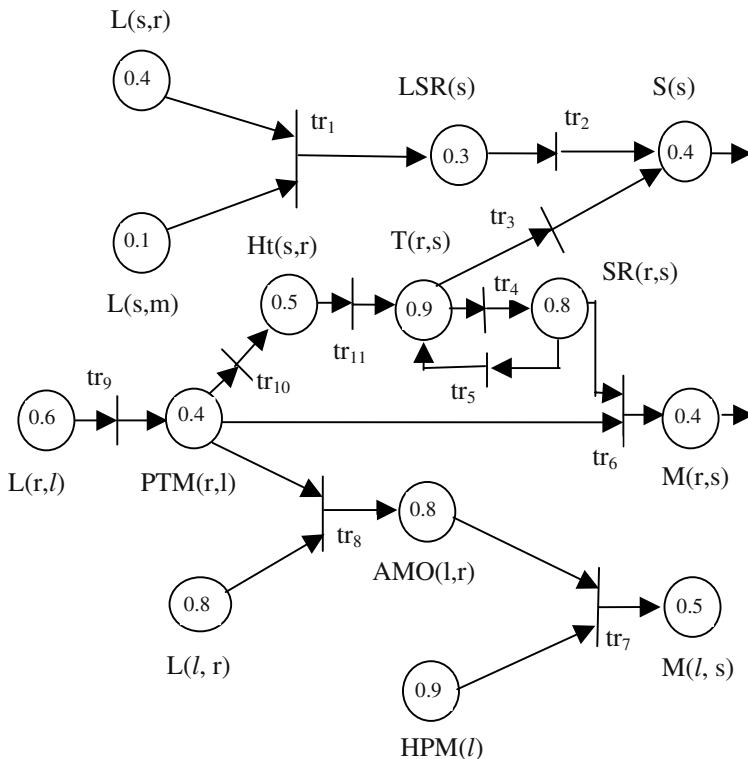
**Fig 14.8:** Representation of DB using datatree for improving searching efficiency.



**Fig 14.9:** Normalization of fuzzy beliefs using membership function.

The reasoning techniques adopted in the chapter are advantageous to the ES designer on the following counts [8]. First, an expert in his specialized domain can submit rules to the system in the form of simple PR and he need not necessarily be conversant with the formalisms of the system. Secondly, explanation tracing, which was highly time-inefficient in production system [2], can be efficiently done here by back-tracking in the FPN, starting from the concluding place and terminating at the axioms. Thirdly, while forming the

FPN, the search-cost for instantiation of the PR with the data in DB has been reduced significantly by representing the database in the form of a two level data-tree. The first level in such tree should include the predicates with variable arguments, while the unifiable predicates with constant arguments may be defined as children of the former predicates in the second level (Fig.14.8) [8]. Analogous search [2] is also carried out in production system before firing a rule. But the partitioning of the DB in hierarchical form for increasing the search efficiency has not yet, to the best our knowledge, been exploited in knowledge engineering. Finally, the highest possible degree of parallelism in an ES can be fully supported by implementing its IE on an FPN. Finally, the normalization [Fig. 14.9] of beliefs from the known characteristics of the information sources keep the reasoning system free from personal bias.



L= Loves, LSR= Loses Social Recognition, S = Commits Suicide, HT =Hates, AMO = Accepts Marriage Offer from , M= Murdered, PTM = Proposed To Marry , HPM = Has Precedence of Murder, SR = Has Strained Relations between , T = Tortures.

**Fig 14.10:** An FPN representing a murder history of a housewife s, where the husband r and girlfriend l of r are the suspects.

## Exercise

1. Draw a cyclic FPN. Identify its axioms and construct self-loops through virtual transitions around the axioms. Name the virtual transitions, and then describe the FPN by P and Q matrices.
2. Consider an FPN with a initial assignment of beliefs at the places. Suppose the net includes 2 pairs of contradictory information. Show the traces of non-monotonic reasoning algorithm to eliminate inconsistency from the network.
3. Show 2 steps of Belief revision in an FPN with initial assignment of beliefs.
4. Verify *Reducenet* and *Evaluatenet* algorithms on an FPN of your own choice.
5. Show traces of limitcycle elimination algorithm on a cyclic FPN having limit cyclic behavior.
6. Identify the transition in the FPN shown in Fig. 14.10, the threshold of which needs to be adapted to eliminate the scope of limitcycles.
7. Construct a non-linear set of curves to represent the mapping from user-provided belief to actual belief taking into consideration of the authenticity level of the source.

## References

- [1] Bernard, P., *An Introduction to Default Logic*. Springer Verlag, ch. 3, pp. 13-30, 1989.
- [2] Buchanan, B. G. and Shortliffe,E. H., *Rule-Based Expert System: The MYCIN Experiments of the Standford University Heuristic Programming Projects*, Addison-Wesley, Addison-Wesley, MA, 1984.
- [3] Bugarin A. J., “Fuzzy reasoning supported by Petri nets,” *IEEE Trans. on Fuzzy Systems*, vol. 2, no. 2, pp. 135-150, 1990.
- [4] Cao, T. and Sanderson, A. C., “Task sequence planning using fuzzy Petri nets,” *IEEE Trans. on System, Man and Cybernetics*, vol. 25, no. 5, pp. 755-769, 1995.
- [5] Chen, S., Ke, J. and Chang, J., “Knowledge representation using fuzzy Petri nets,” *IEEE Trans. Knowledge and Data Engineering*, vol. 2, no. 3, pp. 311-390, Sept.1990.

- [6] Eshera, M. A. and Brash, S. C., "Parallel rule based fuzzy inference engine on mess connected systolic array," *IEEE Expert*, Winter 1989
- [7] Konar, A. and Mandal, A. K., "Nonmonotonic reasoning in expert systems using fuzzy Petri nets," *Advances in modeling and Analysis*, vol.23, no.1, pp. 51-63,1992.
- [8] Konar, A., *Uncertainty management in expert systems using fuzzy Petri nets*, Ph D Thesis, Jadavpur University, India, 1994.
- [9] Konar, A. and Mandal, A. K., "Uncertainty management in expert systems using fuzzy Petri nets," *IEEE Trans. on Knowledge and Data Engineering*, vol. 8, no. 1, pp. 96-105, 1996.
- [10] Lipp, H. P. and Gunther, G., "A fuzzy Petri net concept for complex decision making process in production control, *Proc. of First European Congress on Fuzzy and Intelligent Technology (EUFIT'93)*, Aachen, Germany, vol. I, pp. 290-294, 1993.
- [11] Looney, G. C., "Fuzzy Petri nets for rule based decision making," *IEEE Trans systems, Man, and Cybernetics*, vol. 18,no. 1, Jan.-Feb. 1988.
- [12] Pearl, J., "Distributed revision of composite beliefs," *Artificial Intelligence*, vol. 33, pp. 173-213, 1987.
- [13] Peterson, J. L., *Petri net Theory and Modeling Systems*, Prentice-Hall, NJ, ch. 2, pp.7-30, 1981.
- [14] Pedrycz, W. and Gomide, F., "A generalized fuzzy Petri net model," *IEEE Trans. on Fuzzy Systems*, vol. 2, no. 4, pp. 295-301, 1994.
- [15] Reichenbach, H., *Elements of Symbolic Logic*, Macmillan, NY, pp. 218, 1976.
- [16] Scarpelli, H., Gomide, F. and Yager, R., "A reasoning algorithm for high level fuzzy Petri nets," *IEEE Trans. on Fuzzy Systems*, vol. 4, no. 3, pp. 282-295, 1996.
- [17] Shafer, G and Logan, R., "Implementing Dempster's rule for hierarchical evidence," *Artificial Intelligence*, vol. 33 1987.
- [18] Toagi, M. and Watanabe, H., "Expert system on a chip: an engine for real time approximate reasoning," *IEEE Expert*, Fall 1986.

- [19] Wang, H., Jiang, C. and Liao, S., “Concurrent reasoning of fuzzy logical Petri nets based on multi-task schedule,” *IEEE Trans. on Fuzzy Systems*, vol. 9, no. 3, 2001.
- [20] Yu, S. K., Knowledge representation and reasoning using fuzzy Pr/T net-systems,” *Fuzzy Sets and Systems*, vol. 75, pp. 33-45, 1995.
- [21] Zadeh, L. A., “Knowledge representation in fuzzy logic,” *IEEE Trans. on Knowledge and Data Engineering*, vol. 1, no. 1, Mar. 1988.

# 15

## Fuzzy Models for Face Matching and Mood Detection

*The chapter aims at designing a new methodology for matching of digital gray images using fuzzy membership-distance products, called moment descriptors. These descriptors are estimated for three common kinds of image attributes namely edge, shade and mixed-range. The existing methods for matching of digital images, which are concerned with the comparison of the positions of directed edges, shades and mixed-range in an image with the same of another image, are often prone to error, due to noise and/or variation in illumination. Fuzzy moment descriptors being less sensitive to noise, makes the matching process invariant to the above stray external disturbances. Further, the normalization and sorting of the moment descriptor vectors keep the matching process invariant to size and rotation of images. The general scheme for image matching presented here has successfully been applied to facial image database for personnel identification. The chapter also explores the scope of template matching and human mood detection from facial images using fuzzy logic.*

### 15.1 Introduction

There exist quite an extensive literature on image matching [2, 3, 6, 13, 14, 20]. Most of these methods employ eigen space analysis [17], [22], profile analysis

[9], feature matching [15] and neural nets [11], [21]. Each of these techniques has its own merits and demerits, as already analyzed by Chellapa in his recent paper [5]. One generic limitation of these works lie in exact matching of the image features/ attributes. The images of most scenes, however, are corrupted with noise because of non-uniform lighting of the sources. Further, facial images impose additional restriction in exact matching due to diversity in poses and aging conditions. Spot identification of persons through exact matching of their image attributes thus is not feasible in most cases. Fuzzy logic, which has proved itself successful in matching of inexact data, can equally be used for inexact matching of close image attributes.

Recently, Dellepiane et al. [6] and Wu [24] have developed two distinct approaches for image matching using fuzzy membership functions. [1]. But none of these techniques consider both the membership value of the regions and their relative distances as image descriptors. The above two information together are unique descriptors of images and thus have considerable differences for two alike but distinct images. Further, the membership and relative distance between regions remain unchanged in spite of rotation and size variation of the images. As a consequence the proposed technique is superior to all other existing techniques of inexact matching. A brief outline of the technique is presented below.

In this chapter, a gray image has been partitioned into  $n^2$  non-overlapped blocks of equal dimensions. Blocks containing regions of three possible characteristics, namely, ‘edge’, ‘shade’ and ‘mixed-range’ [19], are then identified and the sub-classes of edges based on their slopes in a given block are also estimated. The degree of membership of a given block to contain edges of typical sub-classes, shades and mixed-range is measured subsequently with the help of a few pre-estimated image parameters like average gradient, variance and the difference of the maximum and the minimum of gradients. Fuzzy moment, which informally means the membership-distance product of a block  $b[i,w]$  with respect to a block  $b[j,k]$ , is computed for all  $1 \leq i, w, j, k \leq n$ . A feature called ‘*sum of fuzzy moments*’, that keeps track of the image characteristics and their relative distances, is used as *image descriptor*. The descriptors of an image are compared subsequently with the same ones of other images. We used an Euclidean distance measure to determine the distance between the image descriptors of two images. For finding the ‘best matched image’ among a set of images, we compute the Euclidean distance of the image descriptors of the reference image with all the available images. The image with the smallest Euclidean distance is considered to be the ‘best matched image’.

The chapter has been classified into twelve sections. Section 15.2 is devoted to the estimation of fuzzy membership distributions of a given block to contain edge, shade and mixed-range. A scheme for computing the fuzzy moments and a method for construction of the image descriptors are presented in

section 15.3. Estimation of Euclidean distances between descriptors of two different images is also presented in this section. An algorithm for image matching along with its time complexity analysis is presented in section 15.4. The insensitivity of the matching process to rotation and size variation of image and noise are discussed in section 15.5 and 15.6 respectively. The simulation results for the proposed matching algorithm are discussed in section 15.7. Implication of the results on image matching using the proposed scheme is listed in section 15.8. A novel scheme for template matching is presented in section 15.9. This scheme is useful for identifying a template face from a group photograph of persons. Simulation results of the template matching scheme is covered in section 15.10. A scheme for human mood detection is addressed in section 15.11. Conclusions are listed in section 15.12.

## 15.2 Image Features and their Membership Distributions

A set of image features such as edge, shade and mixed-range and their membership distribution are formally defined in this section.

**Definition 15.1:** An **edge** is a contour of pixels of large gradient with respect to its neighbors in an image.

**Definition 15.2:** A **shade** is a region over an image with a small or no variation of gray levels.

**Definition 15.3:** A **mixed-range** is a region excluding edges and shades on a given image.

**Definition 15.4:** A linear edge segment that makes an angle  $\alpha$  with respect to a well defined line (generally the horizontal axis) on the image is said to be an **edge with edge-angle  $\alpha$** . In this chapter we consider edges with edge-angle  $-45^\circ$ ,  $0^\circ$ ,  $45^\circ$  and  $90^\circ$ .

**Definition 15.5:** Fuzzy membership distribution  $\mu_Y(x)$  denotes the degree of membership of a variable  $x$  to belong to  $Y$ , where  $Y$  is a subset of a universal set  $U$ .

**Definition 15.6:** The **gradient** [7] at a pixel  $(x, y)$  in an image is estimated by taking the square root of the sum of difference of gray levels of the neighboring pixels with respect to pixel  $(x, y)$ .

**Definition 15.7:** The **gradient difference** ( $G_{diff}$ ) within a partitioned block is defined as the difference of maximum and minimum gradient values in that block.

**Definition 15.8:** The gradient average ( $G_{avg}$ ) within a block is defined as the average of the gradient of all pixels within that block.

**Definition 15.9:** The variance ( $\sigma^2$ ) of gradient is defined as the arithmetic mean of square of deviation from mean. It is expressed formally as

$$\sigma^2 = \sum (G - G_{avg})^2 P(G) \quad (15.1)$$

for all pixels  
in a block

where  $G$  denotes the gradient values at pixels, and  $P(G)$  [18] represents the probability of the particular gradient  $G$  in that block.

### 15.2.1 Fuzzy Membership Distributions

Once the features of the partitioned blocks in an image are estimated following the above definitions, the same features may be used for the estimation of membership value of a block containing edge, shade and/or mixed-range.

Generally, the partitioned blocks in an image contain either edge and shades together or mixed range. Let us first consider the case, when a block contains edge and shades. The gradient in such blocks will have nonzero values only on the edges. So, there must be a small positive average gradient of the pixels in these blocks. Consequently,  $\sigma^2$  should be a small positive number. We may thus generalize that when  $\sigma^2$  is close to zero but positive, the membership of a block  $b_{ij}$  to contain edge is high, and low otherwise. Based on these intuitive assumptions, we presumed the membership curves  $\mu(b_{jk}) = 1 - \exp(-bx^2)$ ,  $b > 0$ . The exact value of  $b$  can be evaluated by employing genetic algorithms [16]. It may be noted that  $1 - \exp(-bx^2)$  has a zero value at  $x = \sigma^2 = 0$  and approaches unity as  $x = \sigma^2 \rightarrow \infty$ . The square of  $x$  ( $x^2$ ) represents the faster rate of growth of the membership curve  $1 - \exp(-bx^2)$ .

Now, let us consider a block containing shades. Here,  $\sigma^2$  should ideally be zero. So, closer is the value of  $x = \sigma^2 = 0$ , the larger should be the membership function of the block  $b_{ij}$  to contain shade. This can be realized with a membership function of the form  $\exp(-ax)$ ,  $a > 0$ .

However, for a block containing mixed range, no definite value of  $\sigma^2$  can be predicted. The value of  $\sigma^2$  for such block depends on the type and pattern of the mixed range. It can, however, be easily ascertained that  $\sigma^2$  for such blocks will neither be too large nor be too small. Thus the membership of a block  $b_{ij}$  to contain mixed range will be higher for moderate value of  $\sigma^2$ . This can be represented by the function

$$cx^2 / (d + ex^2 + fx^3), \quad c, d, e, f > 0. \quad (15.2)$$

Analogously, the average gradient  $G_{avg}$  of a block  $b_{ij}$  containing only edge is large. Thus the membership of a block  $b_{ij}$  to contain edge will be high, when  $G_{avg}$  is large. This phenomena can be represented by the membership function  $1 - \exp(-\alpha x)$ ,  $\alpha > 0$ . Again, for shades, the average gradient is very close to zero. Consequently, the membership of the block to contain shade is high, when  $G_{avg} \rightarrow 0$ . This can be modeled by the membership function  $\exp(-a x^2)$ ,  $x = G_{avg}$  for blocks containing shades. The  $G_{avg}$  of a block containing mixed range will be neither too low nor too high. Thus a function like  $\eta x^2 / (\rho + \theta x^2 + \varphi x^3)$ ,  $\eta, \rho, \theta, \varphi > 0$  may be used to model the membership function of for block  $b_{ij}$  to contain mixed range.

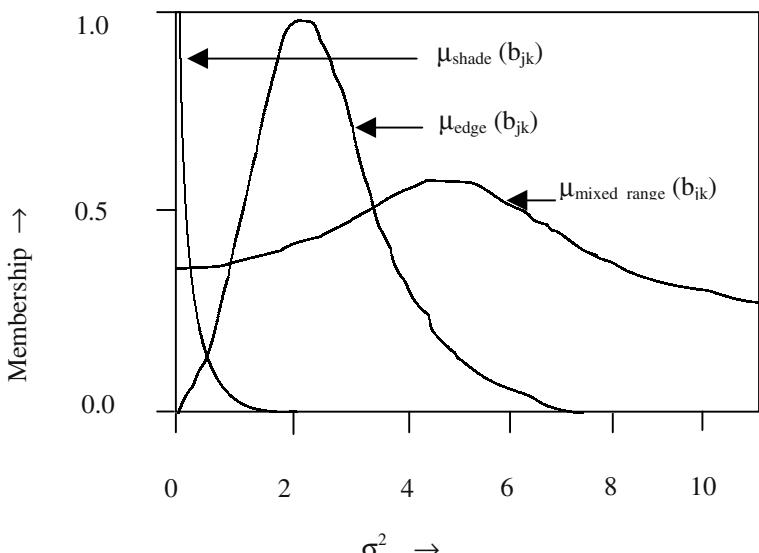
Further for a block containing shade, the maximum and minimum gradients are virtually identical. So,  $G_{diff}$  should be close to zero. Thus a decaying membership curve with a unity value at  $x = \sigma^2 = 0$  and a sharp falloff is required. This has been modeled in our work by  $\exp(-ax^4)$ ,  $a > 0$ . It is to be noted that the power 4 to  $x$  causes a sharp declination in the membership value, as  $x$  rises from zero. The selection of the membership curve for edge and mixed range w.r.t  $G_{diff}$  directly follows from the previous discussions. Table 15.1 below presents the list of membership functions used in this chapter.

**Table 15.1:** Membership functions for features

PARAMETER	Mixed Range Membership	Edge Membership	Shade Membership
$x =$			
$G_{avg}$	$\frac{(\eta x^2)}{(\rho + \theta x^2 + \varphi x^3)}$ $\eta, \rho, \theta, \varphi > 0$	$1 - e^{-8x}$	$e^{-ax^2}$ $a > 0$
$G_{diff}$	$\frac{\alpha x^2}{(\beta + \lambda x^2 + \delta x^3)}$ $\alpha, \beta, \lambda, \delta > 0$	$1 - e^{-bx^2}$ $b > 0$	$e^{-ax^4}$ $a > 0$
$\sigma^2$	$\frac{cx^2}{(d + ex^2 + fx^3)}$ $c, d, e, f > 0$	$1 - e^{-bx^2}$ $b > 0$	$e^{-ax^2}$ $a >> 0$

One sample set of membership distribution of a block containing edge, shades or mixed range against  $\sigma^2$  is presented in Fig. 15.1 for the sake of convenience of the readers unfamiliar with fuzzy systems. It may be noted from this figure that a block having a particular  $\sigma^2$  may be said to contain edge, shade or mixed range but with different membership values.

The membership values of a block  $b[j, k]$  containing edge, shade and mixed-range can be easily estimated if the parameters and the membership curves are known. The fuzzy production rules, described below, are subsequently used to estimate the degree of membership of a block  $b[j, k]$  to contain edge (shade or mixed-range) by taking into account the effect of all the three parameters:  $G_{avg}$ ,  $G_{diff}$ , and  $\sigma^2$  together.



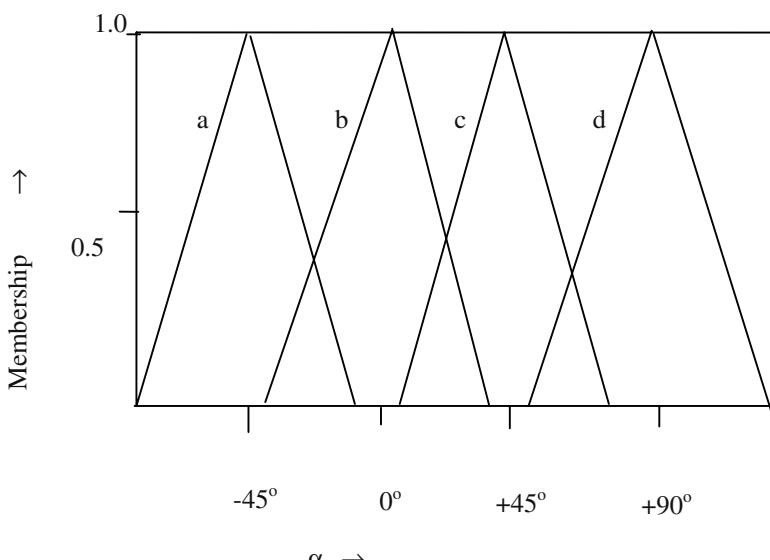
**Fig. 15.1** Membership variations of a block  $b[j, k]$  containing edge, shade or mixed range with respect to  $\sigma^2$ .

### 15.2.2 Fuzzy Production Rules

A fuzzy production rule is an If-Then relationship representing a piece of knowledge in a given problem domain. For the estimation of fuzzy memberships of a block  $b[j, k]$  to contain, say, edge, we need to obtain the composite membership value from their individual parametric values. The If-Then rules represent logical mapping functions from the individual parametric memberships to the composite membership of a block containing edge. The production rule PR1 is a typical example of the above concept.

**PR1:** IF  $(G_{avg} > 0.142)$  AND  
 $(G_{diff} > 0.707)$  AND  
 $(\sigma^2 \leq 1.0)$   
THEN (the block contain edges).

Let us assume that the  $G_{avg}$ ,  $G_{diff}$  and  $\sigma^2$  for a given partitioned block have been found to be 0.149, 0.8 and 0.9 respectively. The  $\mu_{edge}(b_{j,k})$  now can be estimated first by obtaining the membership values  $\mu_{edge}(b_{j,k})$  w.r.t.  $G_{avg}$ ,  $G_{diff}$  and  $\sigma^2$  respectively by consulting the membership curves and then by applying the fuzzy AND (minimum) operator over these membership values. The single valued membership, thus obtained, describes the degree of membership of the block  $b[j, k]$  to contain edge. For edges with edge-angle  $\alpha$ , we use the membership curves vide Fig.15.2 and obtain the composite membership of a block containing edge with edge-angle  $\alpha$  by ANDing the membership of a block containing an edge with the membership of its having an edge angle  $\alpha$ .



**Fig. 15.2** Membership functions of a block  $b[j,k]$  containing edge with edge angle a)  $\alpha = -45^\circ$ , b)  $\alpha = 0^\circ$ , c)  $\alpha = +45^\circ$  and d)  $\alpha = 90^\circ$ .

The composite degree of membership of a block containing shade and mixed-range has been computed similarly with the help of more production rules, the format of which is similar to that of PR1.

### 15.3 Fuzzy Moment Descriptors

In this section, we define fuzzy moments and evaluate image descriptors based on those moments. A few definitions, which will be required to understand the concept, are in order.

**Definition 15.10:** *Fuzzy shade moment*  $[M_{iw}]_{\text{shade}}^{jk}$ , is estimated by taking the product of the membership value  $\mu_{\text{shade}}(b_{j,k})$  (of containing shade in the block  $b[j, k]$ ) and normalised Euclidean distance  $d_{i,w,j,k}$  of the block  $b[j, k]$  w.r.t.  $b[i, w]$ . Formally,

$$[M_{iw}]_{\text{shade}} = \sum_{j,k} d_{i,w,j,k} \times \mu_{\text{shade}}(b_{j,k}) \quad (15.3)$$

Fuzzy mixed-range and edge moments with edge-angle  $\alpha$  are also estimated using definition 15.10 with only replacement of the term “shade” by appropriate features.

**Definition 15.11:** *The fuzzy sum of moments (FSM), for shade*  $S_{iw}$ , w.r.t. block  $b[i, w]$  is defined as the sum of shade moments of the blocks where shade membership is the highest among all other membership values. Formally,

$$S_{iw} = \sum_{\exists j,k} d_{i,w,j,k} \times \mu_{\text{shade}}(b_{j,k}) \quad (15.4)$$

where  $\mu_{\text{shade}}(b_{j,k}) \geq \mu_x(b_{j,k})$ ,  $x \in$  set of features.

The FSM of the other features can be defined analogously following expression (15.4).

After the estimation of fuzzy membership values for edges with edge-angle  $\alpha$ , shades and mixed-range, the predominant membership value for each block and the predominant feature are saved. The FSMs with respect to the predominant features are evaluated for each block in the image. For each of six predominant features (shade, mixed-range and edges with edge-angle  $-45^\circ$ ,  $0^\circ$ ,  $45^\circ$  and  $90^\circ$ ) we thus have six sets of FSMs. Each set of FSM (for example the FSM for shade) is stored in a one-dimensional array and is sorted in a descending order. These sorted vectors are used as descriptors for the image.

For matching a reference image with a set of known images, one has to estimate the image descriptors for the known images. Normally, the image descriptors for a known set of images are evaluated and saved prior to the matching process. The descriptors for the reference image, however, are evaluated in real time when the matching process is invoked. The time required for estimation of the descriptors, therefore, is to be reduced to an extent, whatever possible, to keep the matching process executable in real time.

The matching of images requires estimation of Euclidean distance between the reference image with respect to all other known images. The measure of the distance between descriptors of two images is evident from definition 15.12.

**Definition 15.12:** The **Euclidean distance**,  $[E_{i,j}]_k$  between the corresponding two  $k$ -th sorted FSM descriptor vectors  $V_i$  and  $V_j$  of two images  $i$  and  $j$  of respective dimensions  $(n \times 1)$  and  $(m \times 1)$  is estimated first by ignoring the last  $(n - m)$  elements of the first array, where  $n > m$  and then taking the sum of square of the elemental differences of the second array with respect to the modified first array having  $m$  elements.

It may be added that the elements of the second and the modified first array are necessarily non-zero.

**Definition 15.13:** The measure of distance between two images, hereafter called **image distance**, is estimated by taking exhaustively the Euclidean distance between each of the two similar descriptor vectors of the two images and then by taking the weighted sum of these Euclidean distances.

Formally, the distance  $D_{r,y}$  between a reference image  $r$  and a image  $y$  is defined as

$$D_{r,y} = \sum_k \beta_k \times [E_{i,j}]_k \quad (15.5)$$

where the suffix  $i$  and  $j$  in  $[E_{i,j}]_k$  corresponds to the set of vectors  $V_i$  for image  $r$  and  $V_j$  for image  $y$ , for  $1 \leq i, j \leq 6$ .

For identifying the best matched image (among the set of known images) with respect to the reference image, one has to estimate the image distance  $D_{r,y}$  where  $y \in$  the set of known images and  $r$  denotes the reference image. The image  $q$  for which the image distance  $D_{r,q}$  is the least among all such image distances is considered the **best matched image**.

## 15.4 Image Matching Algorithm

Procedure Image-matching presented below is concerned with the matching of a reference image, here  $IM_1$  with a set of  $m$  images  $IM_2, IM_3, \dots, IM_{m+1}$  by using the concept of fuzzy moments presented in the previous section. The procedure is self-explanatory with the comments presented therein.

### Procedure Image-matching ( $IM_1, IM_2, \dots, IM_{m+1}$ )

// $IM_1$  = reference image //

**Begin**

**For**  $p:=1$  to  $m+1$  **do Begin**

Partition  $IM_p$  into non-overlapping blocks of  $(n \times n)$  pixels;

// estimation of parameters and membership values //

**For**  $block:=1$  to  $n^2$  **do Begin**

---

```

Find-parameters (f(x,y),Gavg ,Gdiff , σ2) ; //f(x,y)=gray level at
(x,y)//
Find-membership (Gavg, Gdiff, σ2, μedge(block), μshade(block),
μMR(block));
End for;
// Sum of moment computation //
For i:=1 to n do Begin
For w :=1 to n do Begin
k:=n × (i - 1)+ w; // Mapping from 2-d with indices (i, w) to 1-d with
index k //
Find-moment-sum(Si w, MRi w, E0i w, EP45i w, E45i w, E90i w);
SP[k]:=Si w;
E45P[k]:=E45i w;
EP45[k]:=EP45i w ;
E0P[k]:=E0i w ;
E90P[K]:=E90i w;
MRP[K]:=MRi w;
End For;
End For;
Sort(SP , MRP , E45P , EP45P , E0P , E90P );
// This procedure sorts arrays SP , MRP etc. into descending order and
places the resulting vectors into corresponding arrays //
// Image identification from Euclidean distance //
For p:=2 to (m+1) do Begin
// p= an index to represent image //
EuclidP :=0; Euclid1 :=0;
For XP ∈ { SP , MRP , E0P , E45P , EP45P , E90P} and
X1 ∈ {S1 , MR1 , E01 , E451 , EP451 , E901} in order do Begin
Find-distance (XP ,X1 ,dXP);
EuclidP :=[EuclidP +dXP2 ]1/2 ;
End For;
If EuclidP > EuclidP-1 Then
image:= p -1 ;
Else image :=p;
End For;
End For;
End.

```

### Time-complexity

Assuming that there exist  $n^2$  partitioned blocks each with  $(m \times m)$  pixels, it can be shown following the above procedure that for 6 descriptors, we at most have  $6n^2$  number of scalar features. Thus instead of  $(n^2 m^2)$  pixel-wise comparisons, one has to perform  $6m^2$  comparisons. The time-efficiency of the algorithm thus

becomes  $(m^2n^2/6n^2) = m^2/6$ . Therefore, larger is the block size, better should be the time efficiency. However, one cannot execute the algorithm when block size becomes equal to the image size. Experimentally for images of dimension  $(512 \times 512)$  it has been found that for block size approximately equal to the  $(1/16)$ th of the image size, the matching algorithm runs perfectly.

## 15.5 Rotation and Size Invariant Matching

In order to keep the matching process free from size and rotational variance of the reference image, the following strategies have been used.

1. The Euclidean distances used for estimation of fuzzy moments are normalized with respect to the diagonal of the image, which is assumed to have a unit distance. Thus the Euclidean distance between each two blocks of an image are normalized with respect to the image itself. This normalization of distances keeps the matching process insensitive to the size variation of the images.
2. The descriptor vectors are sorted so as to keep the moment sums for blocks with most predominant features at the beginning of the array, which only participate subsequently in the matching process. Thus the matching process is free from rotational variance of the reference image. The insensitivity of matching process to size and rotational variance of the reference image has also been established through computer simulation.

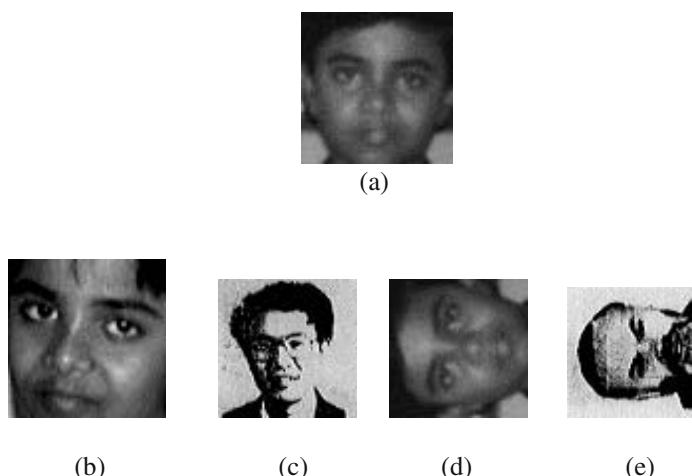
## 15.6 Noise-Insensitive Matching

The matching process used in the chapter does not directly depend on the elementary image attributes like  $\sigma^2$ ,  $G_{avg}$  and  $G_{diff}$  but includes a fuzzy mapping from these features to fuzzy memberships. The fuzzy moments in turn depend on fuzzy memberships and Euclidean distance between blocks. The Euclidean distance between partitioned blocks, for obvious reasons are insensitive to noise. The fuzzy membership distributions, on the other hand, because of their special form of non-linearity, blocks the passage of small Gaussian noise in the elementary image attribute to the computation process of fuzzy moments. For instance, a small Gaussian noise of  $\pm 0.05$  over a  $\sigma^2 = 0.2$  will have as small as  $10^{-6}$  variations when  $1 - \exp(-bx^2)$  with  $x = \sigma^2$  and  $b = 0.005$  is used as the edge membership distribution.

The effect of noise on elementary image attributes is further reduced because of the non-linear mapping of the memberships by the fuzzy production rules.

## 15.7 Computer Simulation

The technique for matching presented in this chapter has been simulated in C language under C<sup>++</sup> environment on an IBM PC/AT with 16 digital images, some of which include the rotated and concise version of the original. The images were represented by  $64 \times 64$  matrices with 32 gray levels. The simulation program takes each image in turn from the corresponding image files and computes the gradients for each pixel in the entire image. The gradient matrix for the image is then partitioned into  $16 \times 16$  non-overlapping sub blocks. Parameters like  $G_{avg}$ ,  $G_{diff}$  and  $\sigma^2$  are then computed for each sub-block. The fuzzy edge, shade and mixed-range moment vectors are also computed subsequently. The above process is repeated for all 16 such images, the first one being the reference (boy image) in Fig.15.3 (a). The simulation program then estimates the normalized Euclidean distance between the reference image and all other subsequent images, of which the first four are shown in fig 15.3 (b)-(e). The Euclidean distance between the reference boy image and the Fig. 15.3 (d) is minimum and found to be zero. It may be noted that Fig.15.3 (b), which corresponds to the image of the same boy taken from a different angle has an Euclidean distance of 1.527 units with respect to the reference boy image. The Euclidean distance of images 15.3 (c) and 15.3 (e) with respect to 15.3 (a) being large enough of the order of 4.92 and 5.15 units respectively proves the disparity of matching. The rotational and size invariance of the proposed matching algorithm is evident from the resulting zero image distance between the reference boy image and the size-magnified and rotated version of the same image.



**Fig.15. 3** Matching of a reference image (a) with images (b) through (e).

It may be added that the feature extraction and descriptor formation for a known set of images, which were performed in the real time by our program, however should be carried out offline before the matching process is invoked. This will reduce significantly the time required for the matching process.

## 15.8 Implication of the Results

It is clear from the experiments presented above that the proposed scheme of image matching is free from size and rotational variance and requires insignificantly small time for the matching process. The smaller the size of the partitioned blocks in the image, the higher the computational time for matching. On the other hand, increasing the dimension of the partitioned blocks hampers the resolution of matching. The choice of the size of each partitioned block, therefore, is a pertinent decisive factor in connection with the process of matching.

The fuzzy membership functions used in this context have been chosen intuitively. There exists ample scope of selecting appropriate membership functions in a judicious manner. Selection of membership functions that causes the least error in the process of matching is yet to be identified. This is an open problem to the best of the author's knowledge to date.

## 15.9 Template Matching Using Interleaved Search

So far we discussed matching of a given image in a image database. This, of course, is needed in identifying a criminal among a set of possible suspects, whose image database is available with the investigation department. There are however situations when the suspect is accidentally photographed with other members in a group. Template matching then becomes essential to check the existence of the suspect in the given photograph. The subsequent part of the chapter provides a novel scheme of template matching, which is concerned with a search for a *template* on a given image in an interleaved manner.

For academic interest only we first propose 3 models of template matching. It will later be shown that all the three models are not equally effective in our application. The composite benefits of these models, however, far exceed their individual merits. So, we will use more than one of these models together for determining the approximate location of the template in a given image first and then design a strategy to uniquely identify the location of the target position by utilizing the composite merits of the these models.

Prior to presenting the models, we briefly narrate the principle of the proposed scheme of template matching. Suppose, we are given an image of  $(n \times n)$  pixels and we want to search a template of  $(m \times m)$  pixels on that image, and

we allow the template as if to move across the image. As the template moves across the image the portion of the image beneath it may be compared with the template. The obvious question then arises: how to select the traversal of the template on the given image? The principle adopted in the present context is first to place the template at the left topmost corner on the image and then compare the selected features of the template with the part of the image beneath the template. The template is now shifted  $m/2$  pixels right, so that its top left vertex overlaps with the  $(m/2+1)$ -th pixel starting from the left top corner of the image.

After the comparison of the template features with the features of the selected part of the image is over, the template is again shifted right by  $m/2$  pixels. The process is thus continued until the template reaches the right boundary of the image. The template is now positioned in a manner so that its top leftmost pixel overlaps with the  $(m/2+1, m/2+1)$  pixel of the original image. The matching of the template is thus continued as it gradually traverses the entire image by moving left to right and top to bottom of the image leaving  $m/2$  pixels in each turn. Such traversal ensures that no part of the image remains unexplored in the matching process. Mathematically, we can describe the above process by a mapping function. Let  $r$  denotes the total number of shifts of the template so far undergone. Then after  $r$ -th shift, the top leftmost pixel of the template should occupy the  $(i, j)$ -th pixel of the given image, where

$$i = \text{DIV}((m/2) r / n) + 1 \quad (15.6)$$

$$\text{and } j = \text{MOD}(r / (2n/m - 1)) \quad (15.7)$$

where  $r$  can take any value between 1 to  $(2n/m-1)$  and DIV and MOD have their usual meaning. Why  $r$  has a maximum value of  $(2n/m-1)^2$  is explained below. Since the template moves  $m/2$  pixels left or down in each turn, it can have a maximum number of  $(2n/m-1)$  number of horizontal shifts. Analogously, it will have  $(2n/m-1)^2$  number of vertical shifts. Thus total number of shifts of the template is  $(2n/m-1)^2$ . The example below best describes the mapping process.

**Example 15.1:** Let the template has been shifted by  $r = 26$  times over the given image of  $(64 \times 64)$  pixels. Assume that the template size is  $(8 \times 8)$  pixels. So, we have  $n=64$ , and  $m=8$ . Suppose we want to determine the top leftmost pixel position of the template in the image. If the top leftmost pixel overlaps with the  $(i, j)$ th pixel of the image then,

$$i = \text{DIV}((m/2) r / n) + 1 = \text{DIV}((8/2) \times 26 / 64) + 1 = 2$$

$$\text{and } j = \text{MOD}(r / (2n/m-1)) = \text{MOD}(26 / (2 \times 64 / 8 - 1)) = 11$$

Thus after 26 shifts the template is positioned on the image with its left topmost corner overlapping with  $(2, 11)$  pixel of the image.

In order to initiate the search process of the template over the given image we need to define some features. The following statistical features [81] have been selected for the activation of the proposed matching scheme.

$$\text{Mean, } M_i = \sum_{b=0}^{L-1} b P(b) \quad (15.8)$$

$$\text{Variance, } V_i = \sum_{b=0}^{L-1} (b - M_i)^2 P(b) \quad (15.9)$$

$$\text{Skewness, } Sk_i = (1/V_i^3) \sum_{b=0}^{L-1} (b - M_i)^3 P(b) \quad (15.10)$$

$$\text{Kurtosis, } Ku_i = (1/V_i^4) \sum_{b=0}^{L-1} (b - M_i)^4 P(b) - 3 \quad (15.11)$$

$$\text{Energy, } E_i = \sum_{b=0}^{L-1} (P(b))^2 \quad (15.12)$$

where  $b$  represents the gray level of each pixel, and  $L$ = number of gray levels in the image and  $P(b)$  represents the probability of a particular gray level  $b$  in that block.

Let  $\delta m_{ij}^2$ ,  $\delta v_{ij}^2$ ,  $\delta sk_{ij}^2$ ,  $\delta ku_{ij}^2$ ,  $\delta e_{ij}^2$  be the normalized squared difference of the respective statistical features of the template with the same of the given image. Specifically, let

$$\delta m_{ij}^2 = (M_{i,j} - M_{\text{template}})^2 / [\max \{\max(M_{i,j} - M_{\text{template}})^2\}] \quad (15.13)$$

$$\forall i \quad \forall j$$

$$\delta v_{ij}^2 = (v_{i,j} - v_{\text{template}})^2 / [\max \{\max(v_{i,j} - v_{\text{template}})^2\}] \quad (15.14)$$

$$\forall i \quad \forall j$$

$$\delta sk_{ij}^2 = (sk_{i,j} - sk_{\text{template}})^2 / [\max \{\max(sk_{i,j} - sk_{\text{template}})^2\}] \quad (15.15)$$

$$\forall i \quad \forall j$$

$$\delta ku_{ij}^2 = (ku_{i,j} - ku_{\text{template}})^2 / [\max \{\max(ku_{i,j} - ku_{\text{template}})^2\}] \quad (15.16)$$

$$\forall i \quad \forall j$$

$$\delta e_{ij}^2 = (e_{ij} - e_{\text{template}})^2 / [\max \{\max \{(e_{ij} - e_{\text{template}})^2\}\}]. \quad (15.17)$$

$\forall i \quad \forall j$

where the suffixes  $i, j$  denote the  $(i, j)$ -th position of a block in a partitioned image of  $(n/m)^2$  blocks. The parameters with suffixes  $i, j$  thus represent their estimated values at the  $(i, j)$ th positioned block. The suffix template denotes that respective parameters correspond to their evaluation at the template image.

The models that have been employed to identify the location of the template in the given image are now presented below:

◆ **Worst-case Best Model:** The worst case best (**WCB**) model attempts to identify the block in the given image where maximum of all the 5 deviations:  $\delta m_{ij}^2, \delta v_{ij}^2, \delta s_{kj}^2, \delta k_{uj}^2, \delta e_{ij}^2$  is minimum. Symbolically, the  $(p, q)$ th block is selected in the worst case best model where

$$d_{ij} = \max (\delta m_{ij}^2, \delta v_{ij}^2, \delta s_{kj}^2, \delta k_{uj}^2, \delta e_{ij}^2) \quad (15.18)$$

$$(2n/m - 1) \quad (2n/m - 1)$$

and  $d_{p, q} = \min_{i=1}^{(2n/m - 1)} \min_{j=1}^{(2n/m - 1)} (d_{ij}). \quad (15.19)$

◆ **Best-case Best Model:** The best case best (**BCB**) model attempts to identify the block in the given image where minimum of all the 5 deviations:  $\delta m_{ij}^2, \delta v_{ij}^2, \delta s_{kj}^2, \delta k_{uj}^2, \delta e_{ij}^2$  is minimum. Symbolically, the  $(p, q)$ th block is selected in the best case best model where

$$d_{ij} = \min (\delta m_{ij}^2, \delta v_{ij}^2, \delta s_{kj}^2, \delta k_{uj}^2, \delta e_{ij}^2) \quad (15.20)$$

$$(2n/m - 1) \quad (2n/m - 1)$$

and  $d_{p, q} = \min_{i=1}^{(2n/m - 1)} \min_{j=1}^{(2n/m - 1)} (d_{ij}). \quad (15.21)$

◆ **Least Min Square (LMS) Model:** In this model we compute the length of the vector  $(|\delta m_{ij}|, |\delta v_{ij}|, |\delta s_{kj}|, |\delta k_{uj}|, |\delta e_{ij}|)$  with respect to the origin of the 5- dimensional space. This is repeated for all  $i, j \in [1, (2n/m - 1)]$  and the block with minimum vector length is then selected. Symbolically, we need to determine the  $(p, q)$  th block in the image where

$$d_{ij} = [\delta m_{ij}^2 + \delta v_{ij}^2 + \delta s_{kj}^2 + \delta k_{uj}^2 + \delta e_{ij}^2]^{1/2} \quad (15.22)$$

$$\text{where } d_{p,q} = \min_{i=1}^{(2n/m-1)} (\min_{j=1}^{(2n/m-1)} d_{i,j}). \quad (15.23)$$

Since these 3 models do not always return the same values of p, q, we employ the following technique for identification of the block that resembles to be the closest match with the template image.

Let  $(p_1, q_1)$  be the position of the left topmost pixel of the template in the original image obtained by least min square model, and  $(p_2, q_2)$  be the same obtained by worst-case-best model. For correctly determining the location of the template in the original image, we execute the following 2 steps. First we construct a square such that the intersection of its diagonals correspond to the point  $(p_1, q_1)$  and one of its vertices correspond to  $(p_2, q_2)$ . In the next step, we match the pixel intensities of the template with the part of the square beneath it by taking the Euclidean distance of the corresponding pixels of the 2 images. The process is repeated for all possible positions of the template within the square. Consequently, the position of the template for which the Euclidean distance is minimum is declared as the nearest matched location of the template in the given image. When  $(p_1, q_1)$  and  $(p_2, q_2)$  are close enough, such that the distance between them is less than half the diagonal of the template, then we construct a square around the centroid  $(p_1, q_1)$ , with its side 1.5 times the side of the template image. An exhaustive pixel-wise search of the template within the square thus formed correctly identifies the location of the template in the given image.

## 15.10 Computer Simulation of Template Matching

A series of experiments with a large number of group photographs of persons and their corresponding facial templates reveals that WCB- and LMS- schemes can correctly identify the location of the template faces in the respective images. Since the search is carried out with an interleaving of half the template height/width, exact localization of the template in the given group photograph is not feasible. However, a pixel-wise search around the selected location as described in the previous section can correctly locate the template in the group photograph. One illustrative template and a group photograph containing the template are presented in Fig. 15.4(b) and (a) respectively. The LMS and WCB-scheme identifies the location of the template just 2 pixels off in the vertical and 3 pixels off in the horizontal frame of reference. The exact position of the template, however, could be correctly identified by the subsequent pixel-wise search around the location determined by the LMS scheme.



(a)



(b)

**Fig. 15.4:** Illustrating template matching with (a) one given group photograph of 2 persons and (b) one facial template; the template has been correctly located in the given photograph.

## 15.11 Human Mood Detection from Facial Expressions

Identification of human moods from their facial expressions is of paramount importance in human-computer interactive systems. Facial expressions that include variations in mouth opening, eye opening and eyebrow-constriction are representative patterns for the detection of human moods. Very few works have so far been reported on mood detection of human beings from their facial expressions. Among the successful attempts, the template matching of facial expressions [23], neural network based approach [10], fuzzy measure and integral approach [8] and other techniques need special mention.

Facial expressions of different persons having a particular mood vary to a wide extent. Consequently matching of facial templates by classical logic no longer provides a promising solution. The logic of fuzzy sets, however, has

proved itself successful in partial matching of events. Very recently it has also been established as a good matching tool for facial templates [4]. However, no works on the use of fuzzy logic in identifying moods from facial template matching have yet been reported. The chapter addresses the issues of human mood detection from facial expressions using fuzzy relational model. The proposed approach is different from the existing fuzzy integral techniques [8] of mood detection.

Rest of the section is organized as follows. Section 15.11.1 includes a detailed discussion on image segmentation for localization of human facial organs, such as eyes, mouth and eyebrows. Eyes and mouth regions have been segmented by thresholding a gray image into a binary image. Eyebrows have been localized from the measure of 2<sup>nd</sup> order statistics including variance, skewness, curtosis etc. Section 15.11.2 of the paper is concerned with measurements of facial features and their fuzzification for detection of moods. Section 15.11.3 provides a set of judiciously selected fuzzy production rules to clearly map the facial extracts onto human moods. In section 15.11.4 fuzzy relational scheme has been presented for identification of the moods. Section 15.11.5 presents a procedure for tuning the fuzzy relational matrix.

### **15.11.1 Image Segmentation and Localization of Facial Components**

This section provides a brief outline to image segmentation and localization of important regions on the image like mouth, eyes and eyebrows. Identification of mouth and eyes has been performed on the Jaya image (Fig. 15.5) by histogram based segmentation with a judiciously selected threshold of 34. Fig. 15.6 shows the results of segmentation. It is clear from Fig. 15.6 that the whole image is segmented into 5 major components: mouth region, eye regions, chin region, nose region and hair region. Localization of the mouth and the eye regions has been accomplished in this paper by template matching.

Measurement of eyebrows-constriction is an important issue in the detection of human moods. Identification of eyebrows requires evaluation of second order statistics. In order to obtain the second order statistics we first computed variance, skewness and curtosis on the given image. The computation of these information is then repeated on the derived results obtained in the previous step. Fig. 15.7 below presents another version of Jaya image demonstrating the results of eyebrow-constrictions.



**Fig. 15.5:** Facial image of Jaya.



**Fig. 15.6:** Segmented image of Jaya shown in Fig. 15.5.

### **15.11.2 Facial Extracts and their Measurements for Mood Analysis**

After the image is partitioned into regions containing mouth, eyebrows and eyes, the following image attributes are identified for the detection of moods

- a) **mouth opening:** Mouth opening is an important parameter to identify the level of happiness and the degree of surprise of most human beings. A large mouth opening is indicative of the subject's high degree of astonishment, whereas a moderate mouth opening is closely co-related with the subject's happiness. Most of the existing literature on human moods [8] reveal that tightening of human lips is an indication of the person's anger or hatred.
- b) **eye opening:** Poets usually compare large eyes with seas. Psychotherapists also believe that large eye opening usually is an indication of the subject's happiness. On being afraid people usually close their eyes. Anger of persons is also reflected in the partial closure of the subject's eyes. Eye opening thus is an important attribute to human mood detection.
- c) **eyebrow-constriction:** Eyebrow-constriction usually is co-related with person's anger or hatred. Both length and width of eyebrow-constrictions are of paramount importance in the analysis of human moods.
- d) **wrinkles around mouth:** Wrinkles around mouth sometimes become an important attribute for mood analysis. In fact it has co-relation with mouth opening. So, in case of imprecise measurements of mouth opening, wrinkles around mouth provide additional information about the subject's mood.



**Fig. 15.7:** Segmentation of eyebrow constriction in another image of Jaya.

Measurements of the above image attributes have been performed by a self-defined windowing scheme. Users need to construct a rectangular window

around the important facial attribute. The measurements of the attributes such as mouth opening, eye opening, eyebrow constriction can easily be accomplished by the proposed windowing scheme. The length of wrinkles around mouth is measured by a self-devised software that traces the wrinkles using the mouse.

### 15.11.3 Fuzzification of Facial Extracts

The measurements undertaken for the proposed work are not free from errors. To eliminate the scope of noise from the measurements, one simple approach is to fuzzify the measurements and employ a set of fuzzy production rules for reasoning about the mood. In this paper we fuzzify the measurements into three fuzzy sets: HIGH, MEDIUM and LOW. The generic form of the membership functions with respect to any parameter  $x$  is given below:

$$\begin{aligned}\mu_{\text{HIGH}}(x) &= 1 - \exp(-ax), a>0, \\ \mu_{\text{LOW}}(x) &= \exp(-bx), b>0, \\ \mu_{\text{MODERATE}}(x) &= \exp[-(x - x_{\text{mean}})^2 / 2\sigma^2]\end{aligned}$$

where

$x_{\text{mean}}$  and  $\sigma^2$  are the mean and variance of the parameter  $x$ .

In this section, we need to fuzzify 4 image attributes such as mouth opening (mo), eye-opening (eo), eyebrow-constriction (ec) and wrinkles around mouth (wam) into 3 fuzzy sets such as HIGH, MEDIUM and LOW. Thus we shall have as many as 12 fuzzy membership distributions.

### 15.11.4 Fuzzy Relational Scheme for Mood Detection

A set of fuzzy production rules is constructed to represent the mapping from fuzzy measurements to fuzzy moods. A few sample rules are presented below to illustrate the scheme.

Rule 1: IF (*eye-opening* is LARGE) &  
           (*mouth-opening* is LARGE) &  
           (*eyebrow-constriction* is SMALL)  
       THEN status of mood = VERY-MUCH *surprised*.

Rule 2: IF (*eye-opening* is LARGE) &  
           (*mouth-opening* is SMALL/ MODERATE) &  
           (*eyebrow-constriction* is SMALL)  
       THEN status of mood = VERY-MUCH *happy*.

A fuzzy relational matrix comprising of 12 fuzzy face descriptors, such as *mouth-opening* LARGE and 18 fuzzy moods, such as VERY-MUCH *surprised* is constructed intuitively for the determination of the influence of the fuzzified face

descriptors on the fuzzy moods. The row and column indices of the relational matrix  $R$ (face descriptors, moods) are determined by taking the following Cartesian products:

column index  $\in \{\text{happy, angry, sad, surprised, afraid, disgusted}\} \times \{\text{VERY, MODERATE, NOT-SO}\}$

row index  $\in \{\text{mo, eo, ec, wam}\} \times \{\text{HIGH, MODERATE, LOW}\}$

The following two vectors are required to represent the fuzzy measurements of facial extracts and the fuzzified moods.

#### Fuzzy descriptor vector $\mathbf{F}$

$$= [\mu_S(\text{eo}) \ \mu_M(\text{eo}) \ \mu_L(\text{eo}) \ \mu_S(\text{mo}) \ \mu_M(\text{mo}) \ \mu_L(\text{mo}) \ \mu_S(\text{ebc}) \ \mu_M(\text{ebc}) \\ \mu_L(\text{ebc}) \dots \mu_L(\text{wam})..]$$

where suffices S, M and L stand for SMALL, MEDIUM and Large respectively.

#### Mood vector $\mathbf{M}$

$$= [\mu_{VH}(\text{mood}) \ \mu_{MH}(\text{mood}) \ \mu_{N-So-H}(\text{mood}) \ \mu_{VA}(\text{mood}) \ \mu_{MA}(\text{mood}) \ \mu_{N-So-A}(\text{mood}) \\ \mu_{VS}(\text{modd}) \ \mu_{MS}(\text{mood}) \ \mu_{N-So-S}(\text{mood}) \ \mu_{Vaf}(\text{mood}) \ \mu_{Maf}(\text{mood}) \ \mu_{N-So-Af}(\text{mood}) \\ \mu_{VSr}(\text{mood}) \ \mu_{MSr}(\text{mod}) \ \mu_{N-So-Sr}(\text{mood}) \ \mu_{Vag}(\text{mood}) \ \mu_{Mag}(\text{mood}) \\ \mu_{N-So-Ag}(\text{mood})]$$

where

V, M, N-SO in suffices denote VERY, MODERATELY and NOT-SO, and H, A, S, Af, Sr and Ag in suffices denote HAPPY, ANXIOUS, SAD, AFRAID, SURPRISED and ANGRY respectively.

The relational equation used for the proposed system is given below:

$$\mathbf{M} = \mathbf{F} \circ \mathbf{R}_{FM}$$

where  $R_{FM}$  is the fuzzy relational matrix with the row and column indices described above.

Given a  $\mathbf{F}$  vector and the relational matrix  $\mathbf{R}_{FM}$ , we can easily compute the fuzzified mood vector  $\mathbf{M}$  using the above relational equation. Finally, to

determine the membership of the moods from their fuzzy memberships we need to defuzzify them using the following type defuzzification rule:

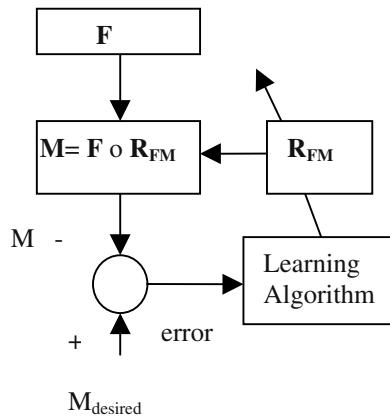
$$\mu_{\text{happy}}(\text{mood}) =$$

$$\frac{\mu_{\text{VH}}(\text{mood}) * w_1 + \mu_{\text{MH}}(\text{mood}) * w_2 + \mu_{\text{N-VH}}(\text{mood}) * w_3}{\mu_{\text{VH}}(\text{mood}) + \mu_{\text{MH}}(\text{mood}) + \mu_{\text{N-VH}}(\text{mood})}$$

where  $w_1, w_2$ , and  $w_3$  denote weights, which in the present context have been set equal to 0.33 arbitrarily.

### 15.11.5 Adaptation of the Relational Matrix

The relational matrix considered in the present problem was initially assigned on an arbitrary basis. The paper, however, provides an automated scheme for adaptation (learning) of the relational matrix. Fig. 15.8 below provides a scheme for automated adaptation of the relational matrix by a learning algorithm. The scheme takes care of the computation of the error vector by taking the difference of the computed mood vector with the actual mood vectors for practical problems. The error vector thus generated is added to each row of the relational matrix so as to reduce the error vector in successive iterations.



**Fig. 15.8:** The adaptation of  $R_{\text{FM}}$ .

### 15.12 Conclusions

The chapter presented a novel scheme for image matching using fuzzy moments as the image descriptors. The proposed technique has successfully been applied on a large database of facial images. The size- and rotation-invariant characteristics of the proposed matching scheme enhanced its domain of

applications to a great extent. Experimental results of image matching also demonstrate that a small angular shift in the orientation of faces does not hamper the matching process. The semantic interpretation of the above phenomena directly follows from the organized matching of the fuzzy moments. Specifically, the ordering of the fuzzy moments (before matching them with other images) organizes the image descriptors in a structured manner, thereby allowing only the significant part of the descriptors to participate in the matching process. Consequently, a small off-orientation of the facial poses does not cause a major change in the image descriptor, thereby ensuring the suitability of the matching scheme for criminal identification from facial images.

The template matching scheme is useful in localization of a module (sub-image) in a given image. Though there exist a number of existing techniques of image localization [12] most of these techniques are applicable on segmented images. Further, efficiency of these techniques are greatly dependent on the neighborhood information of the image. The proposed technique of template matching, however, is free from neighborhood bias.

An experiment with a large number of facial images reveals that happiness, sadness and other moods like being surprised can be detected by the proposed scheme to an accuracy of 98%. On the other hand, moods expressing anger, anxiety and fear have been correctly identified to an accuracy of 80.2% by our scheme.

## Exercise

1. A template of  $(m \times m)$  pixels is searched in an image of  $(n \times n)$  pixels with a horizontal and vertical interleaving of  $(m/2)$  pixels Given that  $n$  is divisible by  $m$ , determine the number of shifts of the template on the image.

[**Answer:** No. of vertical shifts =  $(2n/m - 1)$ . No. of horizontal shifts per unit vertical shift =  $(2n/m - 1)$ . Thus total no. of shifts =  $(2n/m - 1)^2$ .]

2. Fig. 15.9 below provides an image partitioned into equal sized 4 blocks. The edge membership of each block is written on the corresponding block. Find the edge-moment sum with respect to the left-topmost block, A.

A 0.6	B 0.4
0.8 C	0.9 D

**Fig. 15.9:** The edge memberships in a partitioned image.

[**Hints:** Edge-moment sum with respect to left topmost block A=  $0.4 \times 1 + 0.8 \times 1 + 0.9 \times \sqrt{2} = 2.44$ , because the Euclidean distance between blocks A and B , A and C and A and D are 1, 1 and  $\sqrt{2}$  respectively. ]

3. Repeat computation of edge-moment sum with respect to all blocks in the above figure and hence determine the sorted edge moment sum vector.

[**Hints:** Edge moment sum with respect to block B

$$\begin{aligned} &= 0.6 \times 1 + 0.8 \times \sqrt{2} + 0.9 \times 1 \\ &= 1.62. \end{aligned}$$

Edge-moment sum with respect to block C

$$\begin{aligned} &= 0.6 \times 1 + 0.4 \times \sqrt{2} + 0.9 \times 1 \\ &= 2.06. \end{aligned}$$

Edge-moment sum with respect to block D

$$\begin{aligned} &= 0.4 \times 1 + 0.6 \times \sqrt{2} + 0.8 \times 1 \\ &= 2.04. \end{aligned}$$

Edge-moment sum vector

$$\begin{aligned} &= [\text{edge-moment sum w.r.t A, edge-moment sum w.r.t B,} \\ &\quad \text{edge-moment sum w.r.t C, edge-moment sum w.r.t D}] \\ &= [2.46 \quad 1.62 \quad 2.06 \quad 2.04] \end{aligned}$$

The sorted (in descending order) edge-moment sum vector is given by

$$[2.46 \quad 2.06 \quad 2.04 \quad 1.62]. ]$$

4. A 2-gray level (binary) image of  $(128 \times 128)$  pixels is partitioned into 4 non-overlapped blocks of equal size (See Fig. 15.10). Determine shade-membership of the blocks with respect to variance. Assume that

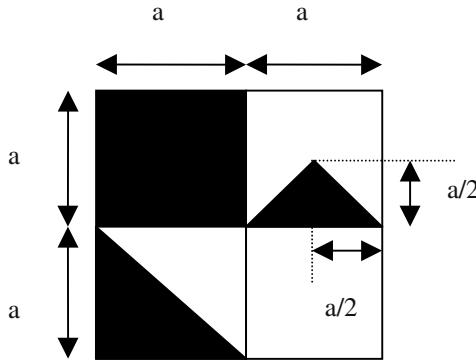
$$[\mu_{\text{SHADE}}(b_{jk})]_\sigma^2 = \text{Exp}(-\sigma^2),$$

where  $\sigma^2$  is the variance in block  $b_{jk}$ . Assume the binary intensity levels in the image are 64 and 0 only.

$$[\text{Hints: } \mu_{\text{SHADE}}(b_{11}) = [\text{Exp}(-\sigma^2)]_{\sigma=0}^2 = 1.$$

$$\mu_{\text{SHADE}}(b_{22}) = [\text{Exp}(-\sigma^2)]_{\sigma=0}^2 = 1.$$

For computing  $\mu_{\text{SHADE}}(b_{21})$ , we first determine average value ( $G_{av}$ ) of pixel intensity in that block.



**Fig. 15.10:** A binary image of  $(128 \times 128)$  pixels partitioned into 4 blocks containing shades.

$$Gav = \frac{(\frac{1}{2})(64 \times 64) \times 0 + (\frac{1}{2})(64 \times 64) \times 64}{(64 \times 64)}$$

$$= 32$$

$$[\sigma^2] \text{ in block } b_{21} = \frac{(\frac{1}{2}) \times 64 \times 64 \times (64 - 32)^2 + (\frac{1}{2}) \times 64 \times 64 (0 - 32)^2}{64 \times 64}$$

$$= (32)^2$$

$$\mu_{\text{SHADE}}(b_{21}) = \exp(-32 \times 32) \approx 0$$

Similarly compute  $\mu_{\text{SHADE}}(b_{12})$ . ]

5. The moment sum vectors of two images are given below. Find their image distance. Consider equal weights of edge -, shade- and mixed range moments for image distance calculation.

Reference image

Edge moment sum vector = [10 8 6 4]

Shade moment sun vector = [6 4 2 1]

Mixed range moment sun vector = [16 12 8 2]

Unknown image

Edge moment sun vector = [8 6 4 2]

Shade moment sun vector = [4 3 1 0]

Mixed range moment sun vector = [12 10 8 4].

## References

- [1] Biswas, B., Konar, A., Mukherjee, A.K., “Fuzzy moments for digital image matching,” *Proceedings of the International Conference on Control, Automation, Robotics and Computer Vision (ICARCV' 96)*, Singapore, 1996.
- [2] Biswas, B., Mukherjee, A. K. and Konar, A., “Matching of digital images using fuzzy logic,” *AMSE Publication*, vol. 35, no. 2, pp. 7- 11, 1995.
- [3] Biswas, B., Konar, A. and Mukherjee, A. K., “Image matching with fuzzy moment descriptors,” *Engineering Applications of Artificial Intelligence*, vol. 14, pp. 43-49, 2001.
- [4] Biswas, B. and Mukherjee, A. K, “Template matching using fuzzy logic,” *J. of Institute of Engineers (India)*, 1999.
- [5] Chellappa, R., Wilson, C. L. and Sirohey, S., “Human and machine recognition of faces: a survey,” *Proceedings of the IEEE '83*, pp. 705-740, 1983.
- [6] Dellepiane, S. and Vernazza, G., “Model generation and model matching of real images by a fuzzy approach,” *Pattern Recognition*, vol. 25, no. 2, pp. 115-137, 1992.
- [7] Gonzalez, R. C. and Wintz, P., *Digital Image Processing*, Addison-Wesley, Reading, MA, 1993.
- [8] Izumitani, K., Mikami, T., and Inoue, K. “A model of expression grade for face graphs using fuzzy integral,” *System and Control*, vol.28, no. 10, pp.590-596,1984.

- [9] Kaufman Jr., G. J., Breeding, K.D., "The automatic recognition of human faces from profile silhouettes," *IEEE Transactions on Systems, man and Cybernetics*, SMC 6, 113 – 121, 1976.
- [10] Kobayashi, H. and Hara,F. "Recognition of six basic facial expressions and their strength by neural network," *IEEE International Workshop on Robot and Human Communication*, 1992.
- [11] Kohonen, T., *Self-Organization and Associative Memory*, Springer-Verlag, Berlin, 1988.
- [12] Konar, A., *Artificial Intelligence and Soft Computing: Behavioral and Cognitive Modeling of the Human Brain*, CRC Press, Boca Raton, 1999.
- [13] Kwon, Y.H., da Victoria Lobo, N., "Face detection using templates," *Proc. of the Int. Conf. of Pattern Recognition*, pp. 764-767, 1994.
- [14] Lee, S. Y., Ham, Y. K. and Park, R. H., "Recognition of human front faces using knowledge based feature extraction and neuro-fuzzy algorithm," *Pattern Recognition*, vol. 29, no. 11, pp. 1863-1876, 1996.
- [15] Manjunath, D. S., Chellappa, P., Malsburg, C.V.D., "A feature based approach to face recognition," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 373-378, 1992.
- [16] Man, K. F., Tsang, K. S., Kwong, S., *Genetic Algorithms*, Springer-Verlag, London, pp. 176-184, 1999.
- [17] Pentland, A., Moghaddam, B., Stener, T. and Turk, M., "View based modular eigen spaces for face recognition," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 84-91, 1994.
- [18] Pratt, W.K., *Digital Image Processing*. Wiley-Interscience, New York, 1978.
- [19] Ramamurthy, B., Gershe, A., "Classified vector quantization of images," *IEEE Transaction on Communications*, vol. Com-34, no. 11, pp. 1105-1115, 1986.
- [20] Sinha, P., "Recognition of changed facial appearances with the aid of a facial image recognition system," *Polish Research and Development*, Ministry of Home Affairs, Government of India, pp. 37-41, 1995.

- 
- [21] Sohan, T.B., Practical face recognition and verification with WISARD, In *Aspects of face Processing*, Ellis, H. D., Jeeves, M. A., Newcomb, F. and Young, A. (Eds.), Kluwer Academic, Nijhoff, Dordrecht, pp. 426-441,1981.
  - [22] Turk, M.A. and Pentland, A., "Face recognition using eigen spaces," *Proceedings of International Conference on Pattern Recognition*, pp.536-591, 1991.
  - [23] Uwechue, O. A. and Pandya, S. A, *Human Face Recognition Using Third Order Synthetic Neural Networks*, Kluwer Academic Publisher, Boston, 1997.
  - [24] Wu, H., Chen, Q. and Yachida, M., "Face Detection from color images using a fuzzy pattern matching method," *IEEE Transaction on Pattern Matching and Machine Intelligence*, vol. 21, no. 6, pp. 557-563, 1999.

# 16

## Behavioral Synergism of Soft Computing Tools

*The chapter introduced the synergistic aspects of different computational tools of machine intelligence including the logic of fuzzy sets, artificial neural networks, genetic algorithms and belief networks. Each of these tools has its inherent merits and demerits. However, a judicious mixture of these tools may sometimes improve the performance of the overall system to a great extent. The chapter explores some of the possible applications, where the integral effect of two or more computational models far exceeds their individual effects. A case study indicating the synergism of 2 different neural nets and GA has been undertaken in this chapter to study its application in motion planning of mobile robots.*

### 16.1 Introduction

The chapters we studied so far covered individual tools of computational intelligence and their applications in different engineering problems. This

---

chapter, however, has a different flavor; it is concerned with the collective utilization of two or more tools, so that the integral effect of the composite tool supersedes their individual power of computation. This has been referred to as *synergism* in the literature of computational intelligence [6]. To understand what we mean by synergism, let us take an example. It is a well-known fact that the oxides of Nitrogen ( $\text{NO}_x$ ), unsaturated oxides of carbon ( $\text{CO}_x$ ) and particulate matters are all individually toxic, but a mixture of them in humid weather far exceeds the sum of their individual toxicity. Thus one plus one is much more than two, and the environmental engineers refer this as the synergism of toxic substances.

It is needless to mention that the example of environmental synergism we presented above is harmful, but the synergism of computational intelligence tools indeed is conducive to human beings. To derive the maximum benefits from the composition of the computational tools, we should first learn their individual merits and demerits, and then attempt to combine them so that the demerits of one tool can be overcome by the merits of the others. For example, fuzzy logic is well known for its power of approximate reasoning, whereas neural network is popularly being used for its power of machine learning. A composition of these two tools may thus be utilized for i) machine learning from noisy training instances, or ii) increasing the accuracy of fuzzy inferences by tuning the parameters of a fuzzy engine using neural networks.

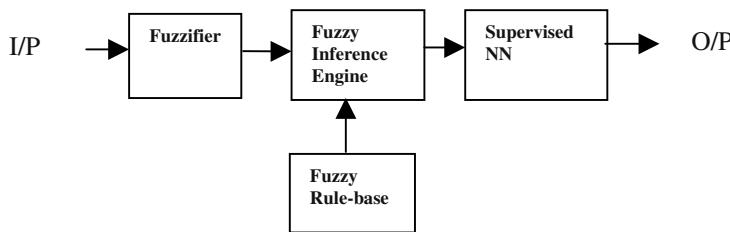
Further, suppose we need to fuse the optimization characteristics of genetic algorithm with the approximate reasoning characteristics of fuzzy logic. Definitely the combination will give rise to a more accurate fuzzy engine. As an alternative example, we can fuse the diagnostic characteristics of a probabilistic belief network with the optimization characteristic of a genetic algorithm, resulting in an efficient and more accurate diagnostic system. Thus fusion of two or more computational tools makes sense.

It is indeed important to note that not only the selection of the constituent members, but the structural combination of them also is an important factor in determining the performance of the overall system. For example, suppose our objective is to design an object recognition system capable of recognizing objects from noisy measurements. We present two alternative schemes to handle the problem. In the first scheme, we may employ fuzzy logic to reduce the effect of measurement noise and neural network for recognizing the object. The second scheme may utilize a fuzzy rule based recognition system with a neural net as a membership tuner. Naturally the question arises: which scheme is better? The answer to this, however, depends on many other issues such as number and structure of the fuzzy production rules, the shape of the membership curves and the neural algorithm used for parameter tuning of the fuzzy unit.

The chapter outlines some of these aspects with examples. It includes 6 sections. Section 16.2 introduces some schemes of neuro-fuzzy synergism. In section 16.3 and 16.4 we present fuzzy-GA and neuro-GA synergism respectively. The principles of synergism in a GA-belief network is outlined in section 16.5. Synergism among GA and two distinct models of neural nets in path planning application of a mobile robot is introduced in section 16.6. Conclusions and future directions are included in section 16.7.

## 16.2 Neuro-Fuzzy Synergism

We already know that an artificial neural network (ANN) is good enough for autonomous machine learning, while a fuzzy system has a significant potential of reasoning with inexact (approximate) data and knowledge. A neuro-fuzzy system, which integrates the behavioral properties of these two systems, thus is capable of learning from approximate data/ knowledge and utilizes the acquired knowledge for futuristic reasoning. We can broadly classify the neuro-fuzzy systems into two types: i) **weakly coupled systems** and ii) **tightly coupled systems**. In a weakly coupled system, the ANNs and the fuzzy systems maintain their identity. A tightly coupled system, on the other hand, employs neurons capable of handling signals using the theory of fuzzy sets; thus the basic elements in the network have the composite characteristics of both neural nets and fuzzy sets. A brief outline to these 2-type of neuro-fuzzy systems is presented in order.

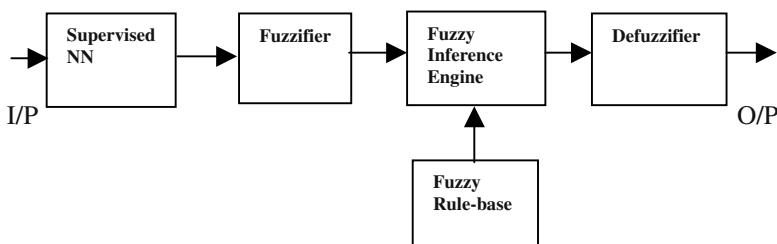


**Fig. 16.1:** A neural net system followed by a fuzzy logic unit.

### 16.2.1 Weakly Coupled Neuro-Fuzzy Systems

A weakly coupled neuro-fuzzy system employs both a neural net unit and a fuzzy system unit in cascade. Depending on the type of applications, the ordering of these 2 units in a composite system may take different forms. In this section we present 2 most common configurations. The first configuration, given in Fig. 16.1 is employed to recognize an object from its approximate features. Suppose we have the approximate measurements about the features of an object, which are transformed to membership values by using intuitively constructed membership functions in the fuzzifier module. A fuzzy inference engine then

maps these primary fuzzified features to other secondary fuzzy features by using a set of fuzzy production rules. The secondary features are usually less affected by the measurement noise of the primary features. The membership values of the secondary features are then supplied to the input of a pre-trained neural net for classification of objects. The output of the neural net in the present context is a vector with one output equal to 1 and the remaining outputs equal to zero. The output that comes up with a one corresponds to a particular object assigned to that output.



**Fig. 16.2:** A Fuzzy-logic system followed by a neural net.

In the second configuration a neural net essentially is placed before the fuzzy logic unit in a neuro-fuzzy system (Fig. 16.2). Here, the neural net first transforms a given set of measurements to a target pattern, which is then fuzzified by a set of membership functions. A fuzzy inference engine in consultation with a set of fuzzy production rules is then used to transform the calculated membership values to a desired set of fuzzy inferences. A defuzzification algorithm [7] then may be invoked to transform the fuzzy variables to the desired non-fuzzy variables. Such systems have major applications in control systems, where the measurement signals and controller input parameters are connected through some nonlinear functions, whose closed form representation is not known.

### 16.2.2 Tightly Coupled Neuro-Fuzzy Systems

In a tightly coupled system the neurons, the basic elements of a neural net, are constructed by amalgamating the composite characteristics of both a neuronal element and fuzzy logic. A large number of tightly coupled neuro-fuzzy systems are referred to in the current literature. Hirota and Pedrycz's AND-OR neuronal model [2], [7], Konar and Pal's fuzzy Petri net model [5], Paul, Konar and Mandal's Fuzzy ADALINE model [8], Pal and Mitra's fuzzy perceptron model [9] are some of the well-known examples of the tightly coupled neuro-fuzzy system.

Among the classical fuzzy neural nets, OR-AND neuron model of Pedrycz [2], [7] needs special mention. In an OR-AND neuron, we have two OR nodes and one AND node (Fig.16.3).

$$\text{Here, } Z_1 = \wedge_{1 \leq j \leq n} (W_{ij} \vee x_j) \quad (16.1)$$

$$Z_2 = \vee_{1 \leq j \leq n} (V_{ij} \wedge x_j) \quad (16.2)$$

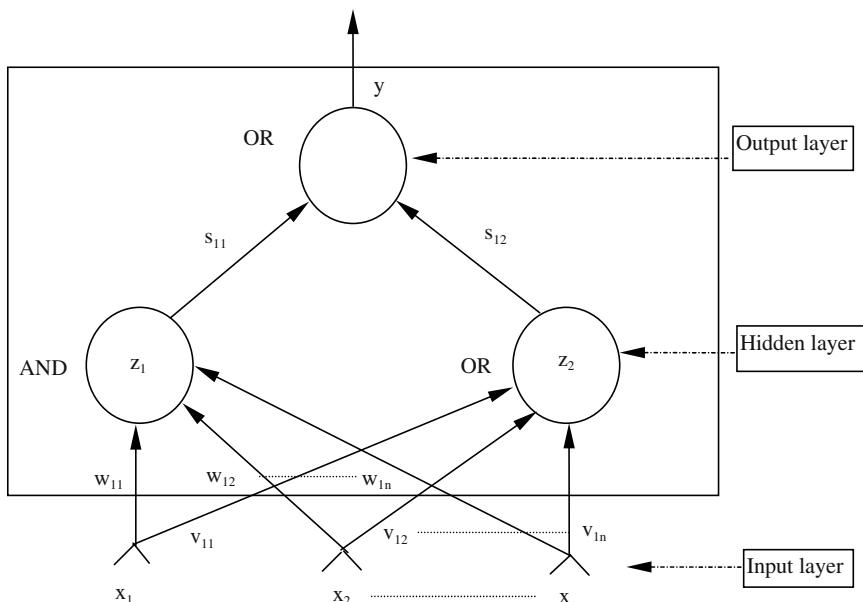
$$\text{and } y = (S_{11} \wedge Z_1) \vee (S_{12} \wedge Z_2) \quad (16.3)$$

where ' $\wedge$ ' and ' $\vee$ ' denote fuzzy ' $t$ ' and ' $s$ ' norm operators. A formal definition of the above two operators is given below.

$$x_1 \wedge x_2 = x_1 \cdot x_2 \quad (16.4)$$

$$x_1 \vee x_2 = (x_1 + x_2 - x_1 \cdot x_2) \quad (16.5)$$

where '.' and '+' are typical algebraic multiplication and summation operations.



**Fig. 16.3:** Architecture of an AND-OR neuron.

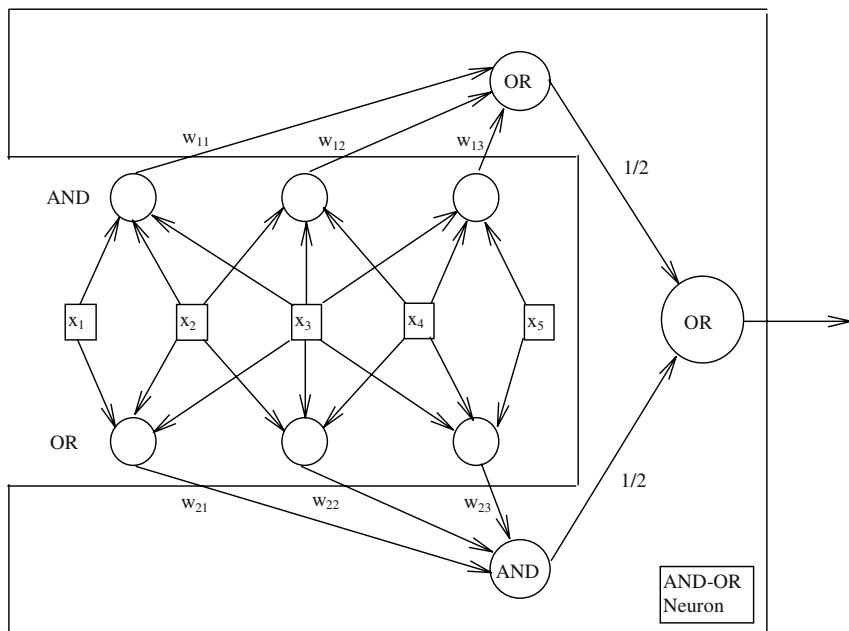
Pedrycz devised a new algorithm for training an AND-OR neuron, when input membership distributions  $x_i$  for  $i = 1 \text{ to } n$  and the target scalar  $y$  are given. The training algorithm attempts to adjust the weights,  $w_{ij}$  so that a pre-defined performance index, is optimized. Exercises 1 and 2 given at the end of the chapter illustrate the scheme of training the AND/ OR neurons by Pedrycz' s scheme.

Pedrycz also designed a pseudo-median filter for using AND-OR neurons, which have many applications in median filtering under image processing. The typical organization of a pseudo-median filter for five data points is given in Fig.16.4. In the figure, the pseudo-median is defined as:

### *pseudomedian ( $\chi$ )*

$$\begin{aligned} &= (1/2) \operatorname{Max} [\operatorname{Min} (x_1, x_2, x_3), \operatorname{Min} (x_2, x_3, x_4), \operatorname{Min} (x_3, x_4, x_5)] + \\ &\quad (1/2) \operatorname{Min} [\operatorname{Max} (x_1, x_2, x_3), \operatorname{Max} (x_2, x_3, x_4), \operatorname{Max} (x_3, x_4, x_5)], \end{aligned} \quad (16.6)$$

which loses little accuracy with respect to a typical median filter.



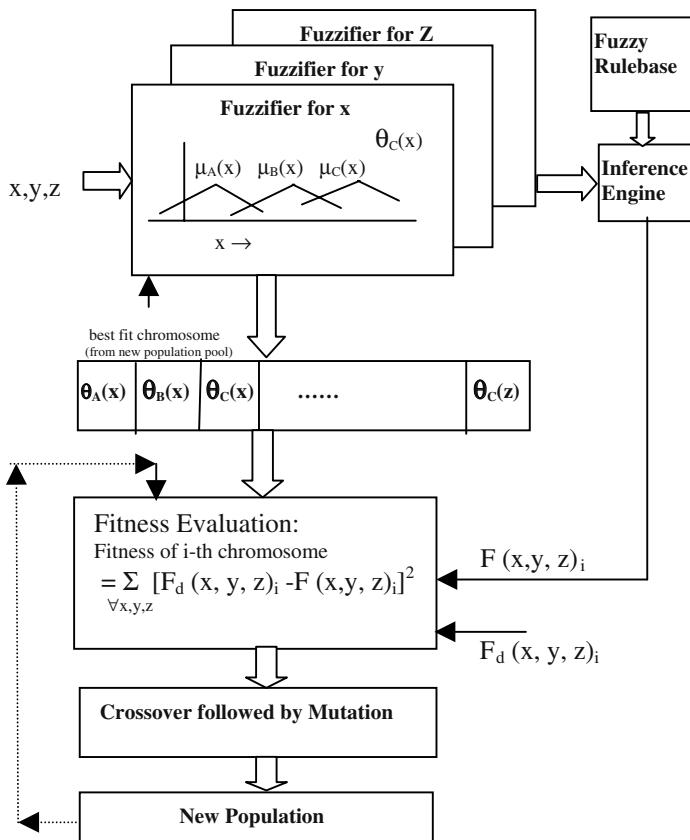
**Fig. 16.4:** A typical pseudo-median filter.

A set of 10 input-output patterns have been used to train pseudo-median filter. The weights  $w_{ij}$  obtained after training the net are saved and used for subsequent

recognition phase. In the recognition phase, the neurons can compute the pseudo-median, when the input data points are given.

### 16.3 Fuzzy-GA Synergism

Fuzzy logic and genetic algorithm can be used in a composite system for one or more of the following reasons. First, the membership functions, which are usually chosen intuitively in a fuzzy system can be optimized by using GA. Secondly, the GA may be employed to adapt the parameters of a fuzzy relational system. Thirdly, the evolutionary process of GA can be realized with fuzzy operators and logic. We briefly outline the first type of synergism of a Fuzzy-GA system below.



**Fig. 16.5:** Parameter adjustment of the fuzzy membership functions by a GA.

Let  $x$ ,  $y$  and  $z$  be three fuzzy variables and  $\mu_A(x)$ ,  $\mu_B(x)$ ,  $\mu_C(x)$  are three fuzzy membership distributions of the fuzzy linguistic variable  $x$  with respect to fuzzy sets A, B and C respectively. Similarly, we have the membership distributions  $\mu(y)$ ,  $\mu_B(y)$ ,  $\mu_C(y)$  and  $\mu_A(z)$ ,  $\mu_B(z)$ ,  $\mu_C(z)$  for linguistic variables  $y$  and  $z$  respectively. The membership distributions mentioned above, suppose, have been constructed intuitively. Now we want to optimize the membership functions by using GA. We can realize it by adjusting the parameters of the membership functions. For brevity of our representation of the chromosomes, let us assume that the membership curves are isosceles triangles with an internal angle between the 2 equal sides =  $\theta$ . For convenience let us denote such angle for the curve  $A_x(x)$  by  $\theta_A(x)$ , say. The same nomenclature is also applicable to  $\theta$  of all curves. The chromosomes in the present context thus has 9 fields, one each for each membership curves. The crossover and mutation operations in the present context are realized in a conventional way, and thus they have no special characteristics.

A question now may arise: how should we select the fitness function? We remember that here we want to select the membership curves to optimize the system behavior. One approach to implement this is to compare the system responses  $F(x, y, z)$  with the desired system responses  $F_d(x, y, z)$  for a given  $i$ -th input vector  $(x, y, z)_i$ , for all known input-output instances  $i$ . So, we must have *a priori* knowledge about the desired system response to tune the membership curves. A brief schematic overview of the system is presented in Fig. 16.5.

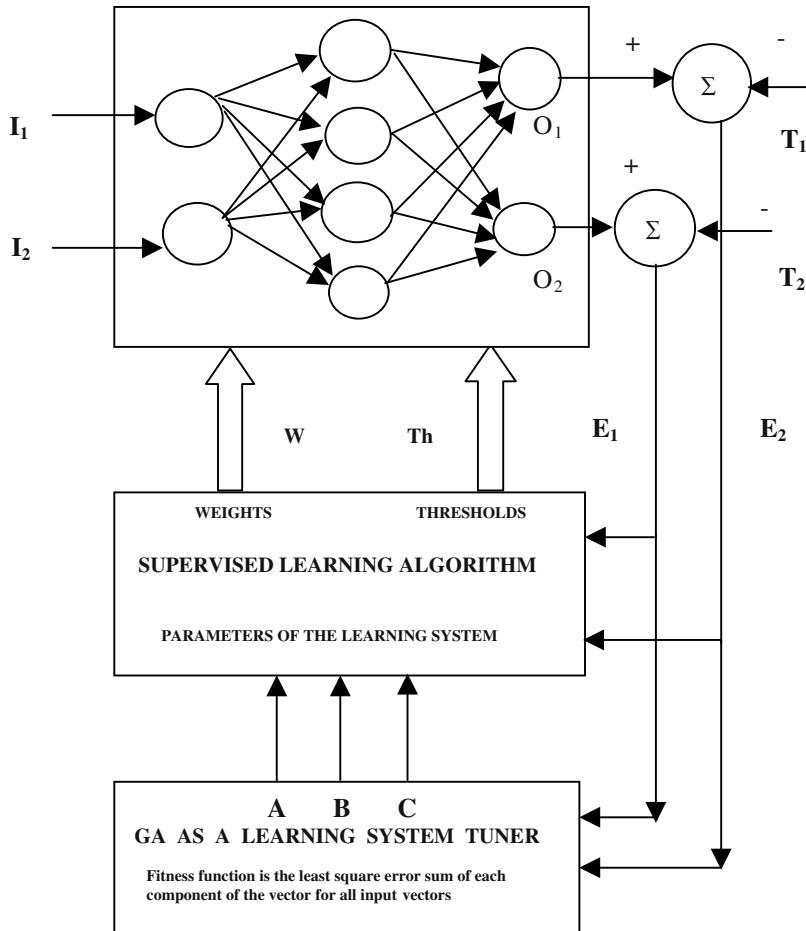
## 16.4 Neuro-GA synergism

The first point that appears in our mind about an ANN is its capability of machine learning. The most important aspect of a GA, on the other hand, is perhaps its application in optimization problems. Fusion of these 2 characteristics of ANN and GA together can give rise to the development of many intelligent machines. This is usually referred to as the neuro-GA synergism. Though GA and neural nets have other form of symbiosis, building machines by utilization of the above 2 characteristics is most common in the current literature [10]. In this section, we briefly outline the scope of the above form of symbiosis in an optimal learning of a neural net.

### 16.4.1 Adaptation of a Neural Learning Algorithm Using GA

A supervised neural network having input vectors  $I_1, I_2, \dots, I_n$  and corresponding target vectors  $T_1, T_2, \dots, T_n$  is usually trained by a learning rule for adaptation of the weight vector  $W$  (and threshold vector  $Th$ ) (Fig. 16.6) of the neural net. Generally, the learning rule includes some intuitively selected parameters like A, B and C. A GA may be employed to optimize the learning algorithm by judiciously adjusting the parameters A, B and C. The

chromosomes in the present context thus should have 3 fields: A, B and C. The fitness function in the present context is the sum of the norm of the error vectors for all possible combination of the input and the target vectors  $I_i$  and  $T_i$  respectively. The norm of an error vector  $E$  here is evaluated by taking the square root of the sum of its squared components  $E_1$  and  $E_2$ . The details of the scheme are available in [4].



**Fig.16.6:** A GA as a learning system tuner.

## 16.5 GA-Belief Network Synergism

It is clear from our discussion on belief networks in chapter 13 that for deriving beliefs of an inference we must have a prior knowledge of the conditional

probabilities of the reasoning system. Generally, experts in a domain assign the conditional probabilities of a reasoning system. But on occasions the conditional probabilities need to be adjusted [3]. A genetic algorithm may be employed for an on-line tuning of the conditional probabilities by comparing the response (here beliefs) of the reasoning system with that of an expert. A mean square error norm fitness function constructed from the system's response and that of the expert may be used to adapt the chromosomes comprising of the conditional probabilities of the system. The tuning range of the probabilities should be small (0.1 say) so that the analysis of a particular case history cannot influence the results much.

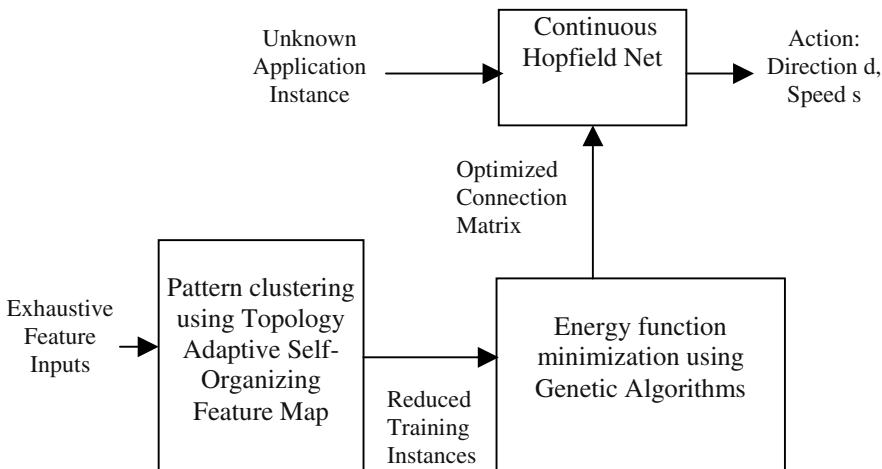
## **16.6 A Case Study of Synergism of 2 Neural Topology and GA**

This section investigates a new technique for efficient obstacle avoidance and path planning of a mobile robot by exploiting the synergism of neural networks and genetic algorithms. The essence of the work lies in the elegant use of the continuous Hopfield net, the topology adaptive self-organizing neural net (TASONN) and the genetic algorithms (GA). The TASONN is employed to classify the exhaustive set of the training instances into a few clusters, which subsequently is used to train the Hopfield net. Determination of the optimal connection weight matrix in the Hopfield net requires an exhaustive search, which can alternatively be accomplished by using GA. The optimal weight matrix is needed for mapping a new training instance to one of the stable states memorized by the continuous Hopfield net. Path planning and obstacle avoidance in a local sense can be achieved in a static and/ or dynamic environment by using the above characteristics of the Hopfield net. Compared to the conventional approaches, the results of our approach revealed a marked improvement in convergence time and efficiency even when the external input is highly distorted.

### **16.6.1 The Problem**

Mobile robots have proved themselves successful in many industrial and hazardous environments. An important problem in mobile robotics is concerned with the path planning and obstacle avoidance of the robot in a static and/or dynamic environment. The section under consideration aimed at building a new technique for path planning and obstacle avoidance of a mobile robot by combining three well-known soft-computing tools, namely the topology adaptive self-organizing feature map, the genetic algorithm and the continuous Hopfield net. The present work establishes experimentally the excellent synergistic effect due to the composition of these three tools.

The input training instances for the continuous Hopfield net are judiciously chosen from the exhaustive set of all possible inputs by the nearest neighbor clustering algorithm, realized by a topology adaptive self-organizing neural net (TASONN). The genetic algorithm has been employed to determine the optimum set of connection weights by minimizing the Liapunov energy function associated with the continuous Hopfield net. The schematic architecture of the overall system is presented in Fig.16.7 below.



**Fig. 16.7:** The schematic architecture of the overall system.

### 16.6.2 Clustering by TASONNN

The training instances for the TASONNN, represented by 11-dimensional feature vectors in Fig. 16.8 are generated exhaustively. These vectors are associated with a 9-dimensional input vector, the fields being range  $R_i$  ( $i=1,2,\dots,8$ ) and goal angle  $\theta$ , and a 2-dimensional output vector, the fields of which are direction  $d$  and speed  $s$ . In our experiment  $R_i$ ,  $\theta$ ,  $d$ ,  $s$  belong to the valuation space in the interval :  $0 \leq R_i, d, s \leq 3$ ,  $0^\circ \leq \theta \leq 360^\circ$ . Due to the topological adaptation of the self-organizing net, logically close patterns are mapped to the same neuron. Consequently, a large number of training instances is clustered into a limited number of patterns. The reduced training instances are later realized as stable points of the Hopfield net.

### 16.6.3 Continuous Hopfield Nets- A Review

A continuous Hopfield neural network is a single layered network where each neuron receives signals  $u_1, u_2, \dots, u_n$  from all other ( $n-1$ ) neurons. Further, each

neuron has an instantaneous input  $\theta_i$  for  $1 \leq i \leq n$ . The dynamics of a Hopfield neural net in vector-matrix form can be stated as

$$\mathbf{C} \frac{d\mathbf{u}}{dt} = -\boldsymbol{\alpha} \mathbf{u} + \mathbf{W} \mathbf{x} + \boldsymbol{\theta} \quad (16.7)$$

where  $\mathbf{C} = \mathbf{C} \mathbf{I}$  is a  $(n \times n)$  matrix,  $\boldsymbol{\theta} = [\theta_1 \ \theta_2 \dots \theta_n]^T$  is the instantaneous input vector,  $\mathbf{x}$  is the output vector,  $\mathbf{u}$  is a  $(n \times 1)$  vector whose  $i$ -th component  $u_i$  is the stored information at the  $i$ -th neuron.  $\boldsymbol{\alpha}$  is a diagonal matrix =  $\text{diag}(\alpha_1, \alpha_2, \dots, \alpha_n)$  and  $\mathbf{W}$  is a weight matrix whose element  $W_{ij}$  denotes connectivity from the neuron  $i$  to neuron  $j$ . It is also known that

$$\mathbf{x} = f(\mathbf{u}) = \mathbf{G} \mathbf{u} \text{ (say)}, \quad (16.8)$$

where  $\mathbf{G}$  is a diagonal gain matrix whose diagonal elements  $G_{ii} = 10^4$  say. When  $\mathbf{G}$  is such a large diagonal matrix, the Liapunov Energy function  $E(\mathbf{x})$  of the Hopfield net has only 2 terms as given below.

$$E(\mathbf{x}) = -1/2 \mathbf{x}^T \mathbf{W} \mathbf{x}^T - \mathbf{x}^T \boldsymbol{\theta} \quad (16.9)$$

At dynamic equilibrium  $d\mathbf{u}/dt = 0$ . Thus expression (16.7) reduces to

$$\boldsymbol{\alpha} \mathbf{u} = \mathbf{W} \mathbf{x} + \boldsymbol{\theta}. \quad (16.10)$$

From expressions (16.8) and (16.10) we have

$$\mathbf{u} = (\boldsymbol{\alpha} - \mathbf{W} \mathbf{G})^{-1} \boldsymbol{\theta}. \quad (16.11)$$

Now, substituting (16.8) in (16.9) we have

$$E(\boldsymbol{\theta}) = -1/2 (\mathbf{G} \mathbf{u})^T \mathbf{W} (\mathbf{G} \mathbf{u})^T - (\mathbf{G} \mathbf{u})^T \boldsymbol{\theta} \quad \left. \right\} \quad (16.12)$$

where  $\mathbf{u} = (\boldsymbol{\alpha} - \mathbf{W} \mathbf{G})^{-1} \boldsymbol{\theta}$ .

In our application of mobile robots,  $\boldsymbol{\theta}$  is the training instance. The diagonal elements of  $\boldsymbol{\alpha}$  have been chosen arbitrarily between 1 to 9.  $\mathbf{W}$  is a random weight matrix to be determined for minimizing  $E(\boldsymbol{\theta})$ .

#### 16.6.4 Perception-to-Action Transformation

The non-linear and associative properties of the continuous Hopfield net have been exploited to generate the action i.e. direction  $d$  and speed  $s$  (Fig. 16.8) of the mobile robot corresponding to an unknown input instance representing the then world map. The weight matrix of the continuous Hopfield net has been determined by minimizing  $E$  using GA as explained in the next section. The

significance of the continuous Hopfield net in the present context lies in its inherent capability of mapping a noisy input instance to a stable state. This has also been experimentally verified.

### 16.6.5 Optimization of the Energy Function Using GA

The topological space for the current problem being quite large as well as diverse, determination of the optimal set of connection weights poses a crucial problem. The Liapunov energy function E is the fitness function used for the selection of good schemas in GA by evaluating the performance of the matrix-type chromosomes.

$$\mathbf{W} = \begin{pmatrix} W_{11} & W_{12} & \dots & W_{1n} \\ W_{21} & W_{22} & \dots & W_{2n} \\ W_{31} & W_{32} & \dots & W_{3n} \\ \vdots & \vdots & & \vdots \\ W_{m1} & W_{m2} & \dots & W_{mn} \end{pmatrix}$$

$W_{ij}$ , in matrix  $\mathbf{W}$ , denotes the weight corresponding to the  $i$ th input of the  $j$ th neuron of the continuous Hopfield net. Here the crossover operation is defined as follows: Let  $\mathbf{W}$  and  $\mathbf{G}$  be the weight matrices whose elements are  $W_{ij}$  and  $G_{ij}$  respectively,  $1 \leq i, j \leq 11$ . The crossover between the matrices  $\mathbf{W}$  and  $\mathbf{G}$  is illustrated below.

$$\begin{pmatrix} W_{11} & W_{12} & \dots & W_{1n} \\ W_{21} & W_{22} & \dots & W_{2n} \\ W_{31} & W_{32} & \dots & W_{3n} \\ \dots \\ W_{m1} & W_{m2} & \dots & W_{mn} \end{pmatrix} \times \begin{pmatrix} G_{11} & G_{12} & \dots & G_{1n} \\ G_{21} & G_{22} & \dots & G_{2n} \\ G_{31} & G_{32} & \dots & G_{3n} \\ \dots \\ G_{m1} & G_{m2} & \dots & G_{mn} \end{pmatrix}$$

which for example yields 2 matrices like

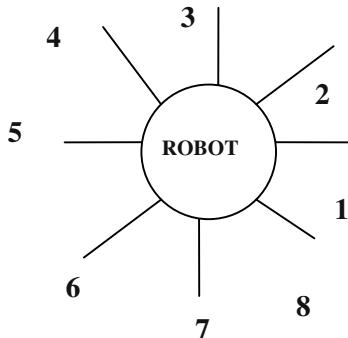
$$\begin{pmatrix} W_{11} & W_{12} & \dots & W_{1n} \\ W_{21} & G_{22} & \dots & G_{2n} \\ W_{31} & G_{32} & \dots & G_{3n} \\ \dots \\ W_{m1} & G_{m2} & \dots & G_{mn} \end{pmatrix}, \quad \begin{pmatrix} G_{11} & G_{12} & \dots & G_{1n} \\ G_{21} & W_{22} & \dots & W_{2n} \\ G_{31} & W_{32} & \dots & W_{3n} \\ \dots \\ G_{m1} & W_{m2} & \dots & W_{mn} \end{pmatrix}.$$

The crossover point is selected randomly, in the entire matrix space. For example, (2, 2) is the crossover point in the above example. The mutation operation in the present context replaces a randomly selected matrix element by taking its 9's complement.

### 16.6.6 Experimental Details

The mobile robot we used in our experiment is a Nomad Super Scout II model, manufactured by Nomadic Technologies, USA. The robot has a built-in Pentium motherboard configured under Linux client mode. The experiment also needed a Pentium server capable of running C programs for the client control under the Linux environment. Two radio modem devices, each having an antenna were employed for online connection between the client and the server. The robot has 16 ultrasonic transducers around its cylindrical structure, but we used only 8 transducers covering a overall angle of  $360^\circ$ .

Fig. 16.8 (a) provides a vertical (cross sectional) view of the robot and its 8 sonar transducers. The sonar transducers emit sonar rays, which on being reflected by a physical object bounces back to the source, and the source calculates the distance (range) of the obstacle from the measurement of time of flight of the rays. Fig. 16.8 (b) provides the general structure of a training instance. Table 16.1 is obtained for different obstacle maps of the robot in a given workspace. It may be noted that each row of the Table describes an obstacle map in the format of Fig. 16.8(b).



**Fig. 16.8(a):** A mobile robot with 8 sonar sensors emitting maximum sonar radiation along the directed lines. The numbers corresponding to the directed lines designate the sonar positions.

$R_1$	$R_2$	$R_3$	$R_4$	$R_5$	$R_6$	$R_7$	$R_8$	$\theta$	$d$	$s$
-------	-------	-------	-------	-------	-------	-------	-------	----------	-----	-----

**Fig. 16.8(b):** Representation of a training instance by a vector, where the component  $R_i$  denotes the range of obstacles measured by the  $i$ th sonar transducer,  $\theta$  denotes the angle formed by the line joining the center of gravity of the robot and the goal position w.r.t a reference axis of the robot, while  $d$  and  $s$  denote the direction and speed of the robot's future motion respectively.

**Table 16. 1:** Sample training instances

<b>R<sub>1</sub></b>	<b>R<sub>2</sub></b>	<b>R<sub>3</sub></b>	<b>R<sub>4</sub></b>	<b>R<sub>5</sub></b>	<b>R<sub>6</sub></b>	<b>R<sub>7</sub></b>	<b>R<sub>8</sub></b>	<b>θ</b>	<b>d</b>	<b>s</b>
1	2	0	0	1	0	3	1	45	1	1
0	3	2	1	0	1	2	1	30	2	2
1	0	0	3	1	1	2	3	180	3	1
3	0	1	0	2	2	1	0	225	3	2
2	1	1	0	1	1	2	1	315	4	2
1	3	2	1	0	0	0	2	135	2	2
3	1	3	2	3	3	0	0	270	3	3
1	0	0	1	0	0	1	0	0	1	1
3	2	1	0	0	1	2	0	115	2	1



(a)

(b)

**Fig. 16.9:** Snapshots of 2 instances of path planning amidst obstacles by a Nomad Super Scout II mobile robot using the proposed Neuro-GA approach. In (a) the robot has started moving from the initial position. In (b) it has traversed some path towards the goal (the other end of the workspace) without hitting an obstacle like chairs.

After generating the training instances, we reduce the number of instances by clustering using SOFM. The robot is then trained with the reduced instances. But how can we train a Hopfield net? We submit the training instance as the input vector ( $\theta$ ) of the net and for all the input instances we need a single weight matrix  $W$  such that the energy function  $E(\theta)$ , given in expression (16.12) is

---

minimized. The minimization in the present context is implemented by using GA. The weight matrix we thus obtain is used later in the application phase.

In the application phase, the goal angle and the input sensory data is mapped to the Hopfield net as its internal state  $x(0)$ . Assuming the input vector  $\theta = 0$ , we solve the Hopfield dynamics (16.7) to obtain the nearest stable state. The stable state we thus obtain has 9 components, out of which the last two components correspond to direction of motion and speed of the robot.

## 16.7 Conclusions and Future Directions

From the individual characteristics of computational intelligence techniques, it can now easily be visualized that fuzzy logic is most useful for reasoning with approximate data and knowledge. Neural nets have much scope in machine learning, and GA is used mainly for the purpose of search, optimization and machine learning as well. Belief network model is an alternative approach to approximate reasoning that works following the fundamental principles of Bayesian statistics.

The chapter presented different schemes for synergism of two or more computational tools. Both weakly coupled and tightly coupled neuro-fuzzy synergism has been addressed. The weakly coupled schemes are widely used in many engineering systems. The tightly coupled models are application-specific and thus may have considerable differences among its realizations. For instance, Konar-Pal's Petri net model has a significant difference with Pal-Mitra's fuzzy perceptron models.

A number of schemes of fuzzy-GA synergism is feasible. We, however, considered a simple scheme for optimization of fuzzy membership functions using GA. We also presented one simple scheme of neuro-GA synergism for tuning a neural learning system using GA. This has enormous applications in information systems, control and pattern recognition.

The case study we have taken up provides a thorough insight of a practical system where two different neural models and GA jointly play an important role in pattern clustering and recognition. Though the application has been focussed to robot motion planning, there is ample scope to extend the application in other engineering domains as well.

## Exercise

1. Height, weight and average speed profile of 3 persons is given in Table 16.2 below.

**Table 16.2:** Height, weight and speed profile of personnels

Height (h)	Weight (w)	Speed (s)
5'	45Kg	5 m/s
6'	48Kg	6 m/s
8'	60 Kg	8 m/s

Also given the following fuzzy production rule:

If height is TALL and weight is MODERATE Then speed is HIGH,

where

$$\mu_{\text{TALL}}(\text{height}) = 1 - \exp(-0.2 \times \text{height})$$

$$\mu_{\text{MODERATE}}(\text{weight}) = \exp(-0.2 \times (\text{weight} - 45)^2)$$

$$\text{and } \mu_{\text{HIGH}}(\text{speed}) = 1 - \exp(-\text{speed}).$$

- a) How will you use a fuzzy-OR neuron to compute  $\mu_{\text{HIGH}}(\text{speed})$  from known distributions of  $\mu_{\text{TALL}}(\text{height})$  and  $\mu_{\text{MODERATE}}(\text{weight})$ ? Clearly mention the topology of the neuron and the input-output parameters of the neuron.
- b) Using steepest descent learning law, develop an algorithm for weight adaptation of the neuron.
- c) Give a complete architecture of the pattern recognition system constructed with a fuzzy OR-neuron. Given the height and weight of a person to be 7' and 50 Kg respectively, determine his speed.

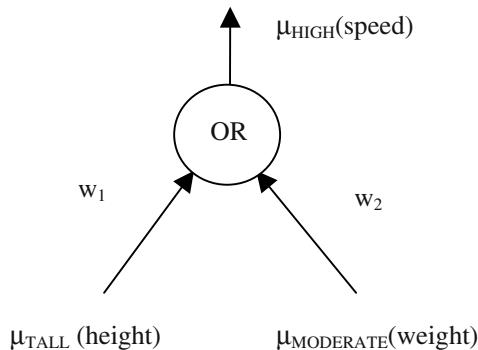
**[Hints:** The fuzzy-OR neuron to be employed for the present application should have two inputs:  $\mu_{\text{TALL}}(\text{height})$  and  $\mu_{\text{MODERATE}}(\text{weight})$ , and one output  $\mu_{\text{HIGH}}(\text{speed})$ . Table 16.3 below presents the input-output training instance for the OR-neuron.]

**Table 16.3:** Input-Output training instance for the OR-neuron

Input instance $\mu_{TALL}$ (height)	$\mu_{MODERATE}$ (weight)	Output instance $\mu_{HIGH}$ (speed)
$\mu_{TALL}(5')$	$\mu_{MODERATE}(45 \text{ Kg})$	$\mu_{HIGH}(5 \text{ m/s})$
$\mu_{TALL}(6')$	$\mu_{MODERATE}(48\text{Kg})$	$\mu_{HIGH}(6 \text{ m/s})$
$\mu_{TALL}(8')$	$\mu_{MODERATE}(60 \text{ Kg})$	$\mu_{HIGH}(8 \text{ m/s})$

Here,  $\mu_{HIGH}(\text{speed}) = (w_1 \wedge \mu_{TALL}(\text{height})) \vee (w_2 \wedge \mu_{MODERATE}(\text{weight}))$

Fig. 16.10 describes the structure of the OR-neuron that performs the logical function mentioned above.

**Fig. 16.10:** The fuzzy-OR neuron used in the present context.

- b) Denoting  $\mu_{TALL}(\text{height}_i)$  by  $x_{1i}$ ,  $\mu_{MODERATE}(\text{weight}_i)$  by  $x_{2i}$  and  $\mu_{HIGH}(\text{speed})$  by  $y_i$  we have

$$y_i = (w_1 \wedge x_{1i}) \vee (w_2 \wedge x_{2i}).$$

The performance index here is given by

$$E = \sum_{i=1}^3 (y_i - y_i^{\text{desired}})^2$$

where  $y_i$  and  $y_i^{\text{desired}}$  are the computed and desired values of variable  $y$ .

The steepest descent learning rule is given by

$$w_1(t+1) \leftarrow w_1(t) - \eta_1 \frac{\partial E}{\partial w_1}, \quad \eta_1 > 0$$

$$w_2(t+1) \leftarrow w_2(t) - \eta_2 \frac{\partial E}{\partial w_2}, \quad \eta_2 > 0$$

where

$$\begin{aligned} \frac{\partial E}{\partial w_1} &= \sum_{i=1}^3 \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial w_1} \\ &= 2 \sum_{i=1}^3 (y_i - y_i^{\text{desired}}) \frac{\partial y_i}{\partial w_1} \end{aligned}$$

For any two fuzzy variables: a and b, we now define fuzzy min and max operators as follows:

$$a \wedge b = a.b$$

$$\text{and} \quad a \vee b = a + b - ab.$$

With the above definitions, we thus have:

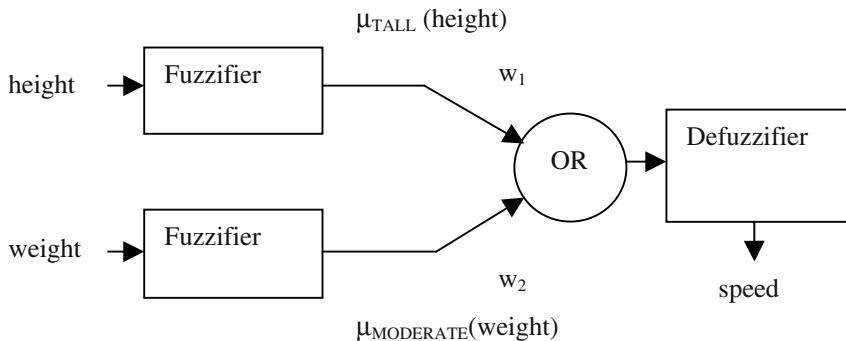
$$\begin{aligned} \frac{\partial y_i}{\partial w_1} &= \frac{\partial}{\partial w_1} [(w_1 x_{1i}) + (w_2 x_{2i}) - (w_1 w_2 x_{1i} x_{2i})] \\ &= x_{1i}(1 - w_2 x_{2i}). \end{aligned}$$

$$\text{Similarly, } \frac{\partial y_i}{\partial w_2} = x_{2i}(1 - w_1 x_{1i})$$

$$\text{and } \frac{\partial E}{\partial w_2} = 2 \sum_{i=1}^3 (y_i - y_i^{\text{desired}}) \frac{\partial y_i}{\partial w_2}$$

Now, iterate  $w_1(t+1)$  and  $w_2(t+1)$  until they converge within a permissible tolerance.

- c) The complete architecture is given below.



**Fig. 16.11:** The complete system.

In Fig. 16.11, the output parameter of the defuzzifier: speed is obtained from the definition of  $\mu_{HIGH}(\text{speed})$  given below:

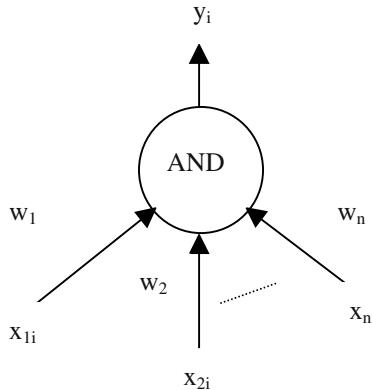
$$\mu_{HIGH}(\text{speed}) = 1 - \exp(-\text{speed}).$$

$$\text{Or, } \text{speed} = \ln[1 / (1 - \mu_{HIGH}(\text{speed}))].$$

Interested readers can evaluate the speed first by computing  $\mu_{HIGH}(\text{speed})$ , and then by evaluating speed using the above formula.]

2. Using steepest descent learning policy, design a learning algorithm for a fuzzy-AND neuron.

[**Hints:** Let  $y_i$  be the output and  $x_{1i}, x_{2i}, \dots, x_{ni}$  are the inputs of the fuzzy-AND neuron. A schematic view of the neuron is given in Fig. 16.12.



**Fig. 16.12:** A fuzzy-AND neuron.

Here,  $y_i = (x_{1i} \vee w_1) \wedge (x_{2i} \vee w_2) \wedge \dots \wedge (x_{ni} \vee w_n)$

Let the desired value of  $y_i = y_i^{\text{desired}}$ . Thus the performance index is given by

$$E = \sum_{\forall i} (y_i - y_i^{\text{desired}})^2$$

and we want to minimize E.

By steepest descent learning rule, we minimize E by setting

$$w_k(t+1) \leftarrow w_k(t) - \eta_k \frac{\partial E}{\partial w_k}, \quad \eta_k > 0$$

for  $k=1$  to  $n$ .

$$\begin{aligned} \text{Now, } \frac{\partial E}{\partial w_k} &= \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial w_k} \\ &= 2 \sum_{\forall i} (y_i - y_i^{\text{desired}}) \frac{\partial y_i}{\partial w_k} \end{aligned}$$

where  $n$

$$y_i = \bigwedge_{\forall j=1}^n (x_{ji} \vee w_j)$$

$$\begin{aligned} &= \prod_{j=1}^n (x_{ji} + w_j - x_{ji} w_j) \\ &= (x_{ki} + w_k - x_{ki} w_k) \prod_{\substack{j=1 \\ j \neq k}}^n (x_{ji} + w_j - x_{ji} w_j) \end{aligned}$$

Therefore,

$$\frac{\partial y_i}{\partial w_k} = (1 - x_{ki}) \prod_{\substack{j=1 \\ j \neq k}}^n (x_{ji} + w_j - x_{ji} w_j).$$

Substitute the results in the expression of  $w_k(t+1)$  and iterate until  $w_k$  converges.]

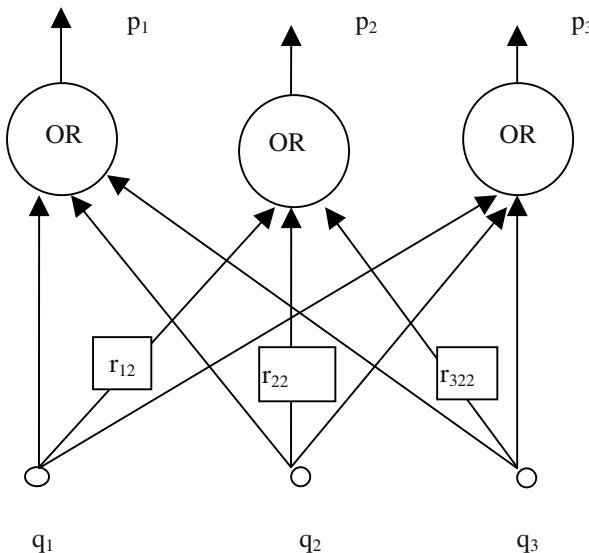
3. Given two vectors  $\mathbf{q} = [q_1 \ q_2 \ q_3]$  and  $\mathbf{p} = [p_1 \ p_2 \ p_3]$ , and also given a relational equation:

$$\mathbf{q} \circ \mathbf{R} = \mathbf{p}$$

where  $\mathbf{R} = [r_{ij}]$  is a fuzzy relational matrix and  $\circ$  is a max-min composition operator.

- Represent the relation  $\mathbf{q} \circ \mathbf{R} = \mathbf{p}$  by a fuzzy neural net.
- Evaluate  $\mathbf{R}$  by steepest descent learning algorithm.

[**Hints:** Fuzzy neural net for the given problem is presented below.



**Fig.16. 13:** A fuzzy neural net representing  $\mathbf{q} \circ \mathbf{R} = \mathbf{p}$ .

- For evaluation of  $r_{ij}$ , we can consider the three OR-neurons separately and evaluate  $r_{ij}$  by steepest descent learning rule.]
4. a) Design the weights of a feed-forward neural network using GA when the network has
- a single input-output training instance, and

- ii) multiple input-output training instances.
- b) Why GA-based learning is better than back-propagation learning in a neural net?

## References

- [1] Abraham, A. and Koppen, M., *Hybrid Information Systems*, Physica-Verlag, Heidelberg, 2002.
- [2] Hirota, K. and Pedrycz, W., "OR/AND neuron in modeling fuzzy set connectives," *IEEE Trans. on Fuzzy Systems*, vol. 2, pp. 151-161, 1994.
- [3] Jamshidi, M., Titli, A., Zadeh, L. A. and Boverie, S., *Application of Fuzzy Logic: Towards High Machine Intelligence Quotient Systems*, Prentice-Hall, NJ, 1997.
- [4] Konar, A. and Jain, L. C., An introduction to computational intelligence paradigms, In *Practical Applications of Computational Intelligence Techniques*, Jain, L. and Wilde, P. D. (Eds.), Kluwer Academic Press, Dordrecht, 2001.
- [5] Konar, A. and Pal, S., Modeling Cognition with Fuzzy Neural Nets, In *Fuzzy Theory Systems: Techniques and Applications*, Leondes, C. T. (Ed.), vol. 3, Academic Press, 1999.
- [6] Kosko, B., *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*, Prentice-Hall, NJ, 1992.
- [7] Pedrycz, W., *Fuzzy Sets Engineering*, CRC Press, Boca Raton, FL, pp. 73-106, 1995.
- [8] Paul, B., Konar, A. and Mandal, A. K., "Fuzzy ADALINEs for gray image recognition," *Neurocomputing*, vol. 24, pp. 207-223, 1999.
- [9] Pal. S. K. and Mitra, S., *Neuro-Fuzzy Pattern Recognition*, John Wiley & Sons, NY, 1999.
- [10] Ruan, D., *Intelligent Hybrid Systems: Fuzzy Logic, Neural Networks and Genetic Algorithms*, Kluwer Academic, Dordrecht, 1997.

# 17

## Object Recognition from Gray Images Using Fuzzy ADALINE Neurons

*The chapter aims at extending the scope of application of Widrow-Hoff's ADALINE model from binary to gray level (fuzzy) pattern recognition. The condition of stability for the extended ADALINE model has been derived and the algorithm for training the multi-layered feed-forward neural net consisting of ADALINE neurons has been presented. The time required for training the neural net is insignificantly small. The scheme for the recognition of objects from their gray level images, using fuzzy ADALINE model, is translation-, rotation- and size- invariant.*

### 17.1 Introduction

In early sixties, Prof. Widrow of Electrical Engineering Department, Stanford University opened up a new frontier of research on Neurocomputing, which had

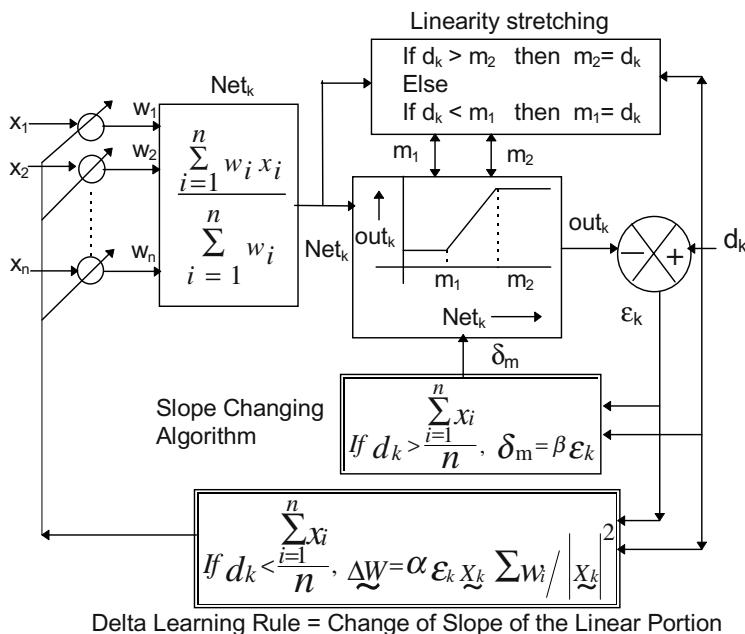
a far-reaching impact on intelligent signal processing for years to come. He introduced the concept of ADALINEs, the electrical analogue of the so-called biological neurons, which were self-adaptive and responsive to changes in input-output pattern by adjusting their weights autonomously. A non-linear signum type inhibiting element rendered the output of the ADALINES to binary  $\{+1, -1\}$  levels. The artificial neural net proposed by him is of multi-layered feed-forward geometry, consisting of a number of cascaded layers of ADALINES. The training algorithm for adaptation of weights consists of i) identification of an ADALINE at a given layer based on minimum disturbance principle [4, 10-14] and ii) adjustment of weights by using Delta rule [13]. The most important contribution of Widrow's work, however, was the selection of the weight matrices of ADALINES in retinal planes, in order to keep the output of the ADALINE-planes insensitive to translation, rotation and magnification of input binary patterns. The invariant scrambled binary patterns, thus obtained, could be descrambled by a two-layer trained neural net.

The delta learning rule also called the Least-Mean-Square (LMS) algorithm [1], proposed by Widrow-Hoff was successfully employed in solving many complex problems of signal processing and pattern recognition. For example, Robert W. Lucky of AT & T Bell Labs invented a new scheme of adaptive equalization of telecommunication channels by employing a "decision oriented learning" technique [5] following the delta rule. At present the LMS algorithm is extensively used by telecommunication industries for adaptive equalization of channels [13]. The "inverse modeling problem" [1] realized with ADALINE found applications in adaptive control and "deconvolution" in geophysical signal processing [13]. Even after the invention of the back-propagation training algorithm [9], Widrow-Hoff's model is still used in many real world applications, where hard non-linearity in the neuronal model is essential. Recently, Hui and Zak [2] applied the LMS algorithm to McCulloch-Pitts type neurons [6] with soft (differentiable) non-linearity and analyzed its convergence. They have shown the robustness of the algorithm in presence of imperfections in the non-linear activation function [6].

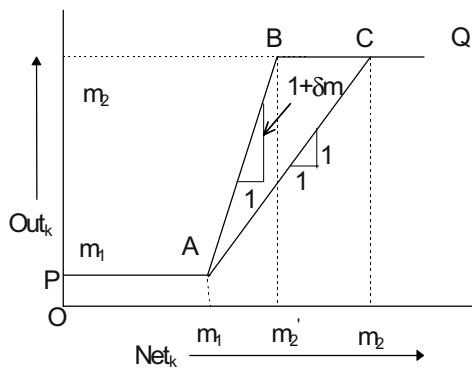
The chapter is an extension of Widrow-Hoff's ADALINE to enhance its scope of application from binary to gray images [8]. In order to handle gray images, the weights as well as the non-linear element of the ADALINEs are required to be adaptive. Further, the functional form of the non-linearity and its position in the neuronal structure also needs modifications. One possible type of the non-linearity is the two level clipper [7], that could be positioned inside the loop of ADALINE, which is in contrast to that of Widrow's model, where the non-linear element was placed outside the loop. Besides, to make the response of the system insensitive to translation, rotation and scale change, the MAJ operator [3] in the retinal planes in Widrow's model should be replaced by AVG (Average) operator, that computes the arithmetic average of the input signals.

The chapter presents the condition for stability of the neuronal parameters that guarantees the convergence of the Learning algorithm. Other important issues covered in the chapter are (i) the concurrent training of neurons in the entire network and (ii) the design of the scrambler network that yields a unique output pattern, when presented with input patterns having rotation, translation and size variance.

The chapter has been divided into 6 sections. Section 17.2 of the chapter deals with the proposed model, while the condition for stability of system weights is discussed in section 17.3. Section 17.4 delineates the training algorithm of the modified multilayered neural net consisting of ADALINE neurons. The aspects of translation, rotation and size-invariant pattern recognition are discussed in section 17.5. Finally, the conclusions are summarized in section 17.6.



**Fig. 17.1.** The proposed model of ADALINE having provision for adjustment of slope of the linear portion in the inhibiting function, when  $d_k > \sum x_i/n$ .



**Fig. 17.2:** Adjustment of slope in the central portion of the clipper from AC to AB.

## 17.2 Proposed Model of Fuzzy ADALINE

In order to widen the scope of applications of the ADALINE neurons to include the multilevel (gray) image recognition, the following modifications are envisaged.

1. The signum type non-linearity is to be replaced by a two level clipper.
2. The position of the non-linear function block is shifted inside the loop, unlike the case of Widrow's model, where it was on the forward path outside the loop.
3. A provision for stretching the linear portion of the non-linearity is to be incorporated in the model.

In the proposed model, both the input and the output being constrained in the range [0,1], the unity slope in the non-linearity of the neuron (Fig. 17.1) will fail to yield the desired output whenever the target signal is higher than the average of the inputs.

In order to circumvent this situation, an adaptive algorithm for adjustment of the slope of the linear portion in the non-linear clipper function is necessary. The detailed scheme for slope adjustment is presented in Fig. 17.1.

Unity slope of the linear region AC in Fig. 17.2 is increased by  $\delta m$  so that AC assumes new position AB.

**Theorem17.1:** *The functional representation of expression 17.1 describes the non-linearity PABQ.*

$$\text{Out}_k = [ \{ (\text{Net}_k \wedge m_2') \vee m_1 \} + ((\text{Net}_k \wedge m_2') \vee m_1) - m_1 \} \delta m ] \quad (17.1)$$

*Proof:* The non-linear function PACQ, in Fig. 17.2 can be described as follows.

$$\begin{aligned} \text{Out}_k &= m_1, & \text{Net}_k \leq m_1 \\ &= m_2, & \text{Net}_k \geq m_2 \\ &= \text{Net}_k, & m_1 < \text{Net}_k < m_2. \end{aligned}$$

Combining the above expressions, we find

$$\text{Out}_k = (\text{Net}_k \wedge m_2) \vee m_1 \quad (17.2)$$

When the slope of the central (linear) region is changed from AC to AB, the equation of line AB is given by

$$\text{Out}_k = (1 + \delta m) \text{Net}_k + c, \quad (17.3)$$

where  $c$  is a constant.

However, to keep equation (17.3) valid for  $m_1 < \text{Net}_k < m_2'$ , we rewrite it as follows:

$$\begin{aligned} \text{Out}_k &= (1 + \delta m) [(\text{Net}_k \wedge m_2') \vee m_1] + c, \\ \text{when } m_1 < \text{Net}_k < m_2' \end{aligned} \quad (17.4)$$

The discontinuity of the non-linearity at points A and B can be made (pseudo) continuous by replacing the symbol ' $<$ ' in (17.4) by ' $\leq$ '. Further, the equations for region PA and BQ are given by (17.5) and (17.6) respectively.

$$\text{Out}_k = m_1, \quad \text{when } \text{Net}_k \leq m_1 \quad (17.5)$$

$$= m_2, \quad \text{when } \text{Net}_k \geq m_2' \quad (17.6)$$

Now to compute the constant 'c', we consider the point A in Fig.17. 2 by using expressions (17.4) and (17.5), which yields:

$$\begin{aligned} m_1 &= (1 + \delta m) [ (m_1 \wedge m_2') \vee m_1] + c, \\ \Rightarrow m_1 &= (1 + \delta m) m_1 + c \\ \Rightarrow c &= -\delta m m_1 \end{aligned} \quad (17.7)$$

Substituting the value of 'c' from (17.7) into (17.4), we find

$$\begin{aligned} Out_k &= (1 + \delta m) [ (Net_k \wedge m_2') \vee m_1] - \delta m m_1 \\ &= [\{(Net_k \wedge m_2') \vee m_1\} + ((Net_k \wedge m_2') \vee m_1) - m_1] \delta m \quad [(17.1) \text{ re-written}] \end{aligned} \quad (17.8)$$

which satisfies expressions (17.4), (17.5) and (17.6) uniquely.

**Corollary:** When  $\delta m \rightarrow 0$ ,  $m_2' \rightarrow m_2$ , signifying that the linear portion is changed from AB to AC.

The above corollary is, however, for academic interest only.

### 17.3. Stability Analysis for Convergence of the ADALINE Model

Consider the ADALINE neuron of Fig. 19.1 where  $d_k$  and  $Out_k$  denote the target and the output signal of the ADALINE at iteration  $k$ . Thus the error signal  $\varepsilon_k$ , at iteration  $k$ , is given by

$$\begin{aligned} \varepsilon_k &= d_k - Out_k \\ &= d_k - [ \{(Net_k \wedge m_2') \vee m_1\} + \{((Net_k \wedge m_2') \vee m_1) - m_1\} \delta m ] \\ &= d_k - [Net_k + (Net_k - m_1)\delta m], \text{ if } m_1 < Net_k < m_2' \end{aligned} \quad (17.9)$$

where  $Net_k = \frac{\sum_{i=1}^n w_i x_i}{\sum_{i=1}^n w_i}$

It may be verified easily that when  $Net_k < m_1$  or  $Net_k > m_2'$ , the training algorithm is unconditionally stable. We, thus attempt to find the condition for stability of neuronal parameters for  $m_1 < Net_k < m_2'$ .

Assuming that the input signals of the ADALINE at instant  $k$  be a vector  $\underline{\underline{X}}_k$  whose  $i$ -th component  $x_i$  denotes the scalar signal at the  $i$ -th input.

Analogously, let  $\underline{\underline{W}}_k$  be the weight vector at instant  $k$ , whose  $i$ -th component  $w_i$  is the weight for the signal at the  $i$ -th input.

$$\text{Thus, } \sum w_i x_i = \underline{\underline{X}}_k^T \cdot \underline{\underline{W}}_k \quad (17.10)$$

Substituting (17.10) in (17.9) we find,

$$\epsilon_k = d_k - \frac{\underline{\underline{X}}_k^T \cdot \underline{\underline{W}}_k}{\sum_{i=1}^n w_i} - \left( \frac{\underline{\underline{X}}_k^T \cdot \underline{\underline{W}}_k}{\sum_{i=1}^n w_i} - m_1 \right) \delta m, \quad (17.11)$$

$$\begin{aligned} \text{so, } \Delta \epsilon_k &= \Delta d_k - \frac{1}{\sum_{i=1}^n w_i} \left( \underline{\underline{X}}_k^T \cdot \Delta \underline{\underline{W}}_k (1 + \delta m) \right) + \Delta (m_1 \delta m) \\ &= - \frac{1}{\sum_{i=1}^n w_i} \underline{\underline{X}}_k^T \cdot \Delta \underline{\underline{W}}_k (1 + \delta m) \end{aligned} \quad (17.12)$$

Now, we shall use the following form of Widrow-Hoff's delta rule

$$\Delta \underline{\underline{W}}_k = + \alpha \epsilon_k \sum_{i=1}^n w_i \frac{\underline{\underline{X}}_k}{|\underline{\underline{X}}_k|^2} \quad (17.13)$$

Substituting expression (17.13) in expression (17.12), we find:

$$\Delta \epsilon_k = - \alpha \epsilon_k (1 + \delta m) \quad (17.14)$$

The solution of the above difference equation is given by

$$\epsilon_k = \epsilon_0 [1 - \alpha (1 + \delta m)]^k \quad (17.15)$$

For the stability of the system, it is sufficient that the magnitude of error at iteration  $k$  should be less than that at iteration  $k - 1$ , i.e.,

$$|1 - \alpha(1 + \delta_m)| < 1, \quad (17.16)$$

which yields,  $0 < \alpha < \frac{2}{1 + \delta_m}$ .

For instability,  $\varepsilon_k \rightarrow \infty$  as  $k \rightarrow \infty$  which yields

$$|1 - \alpha(1 + \delta_m)| > 1$$

$$\Rightarrow \alpha > \frac{2}{1 + \delta_m}, \quad (17.17)$$

since  $\alpha < 0$  is not feasible.

Lastly, for oscillatory behaviour, one can easily find that

$$\alpha = \frac{2}{1 + \delta_m} \quad (17.18)$$

## 17.4. Training of the Proposed Neural Net

The proposed neural net consists of a number of layers with a number of ADALINE neurons in each layer (vide Fig. 17.4). Before describing the training process of the neural net, we first present the training of a single ADALINE neuron.

### 17.4.1. Training Algorithm of ADALINES

The training algorithm of an ADALINE neuron having input vector  $\underline{\underline{X}}$  and target scalar  $d_k$  is presented below.

**Procedure Adjust\_Adaline****Begin****Repeat**

$$\text{if } d_k \leq \frac{\sum_{i=1}^n x_i}{n}$$

$$\text{then } \Delta \tilde{w}_k := \alpha \varepsilon_k \sum_{i=1}^n w_i \frac{x_i}{|X_k|^2}$$

$$\text{else } \delta m := \beta \varepsilon_k;$$

$$\text{if } d_k > m_2$$

$$\text{then } m_2 := d_k;$$

$$\text{if } d_k < m_1$$

$$\text{then } m_1 := d_k;$$

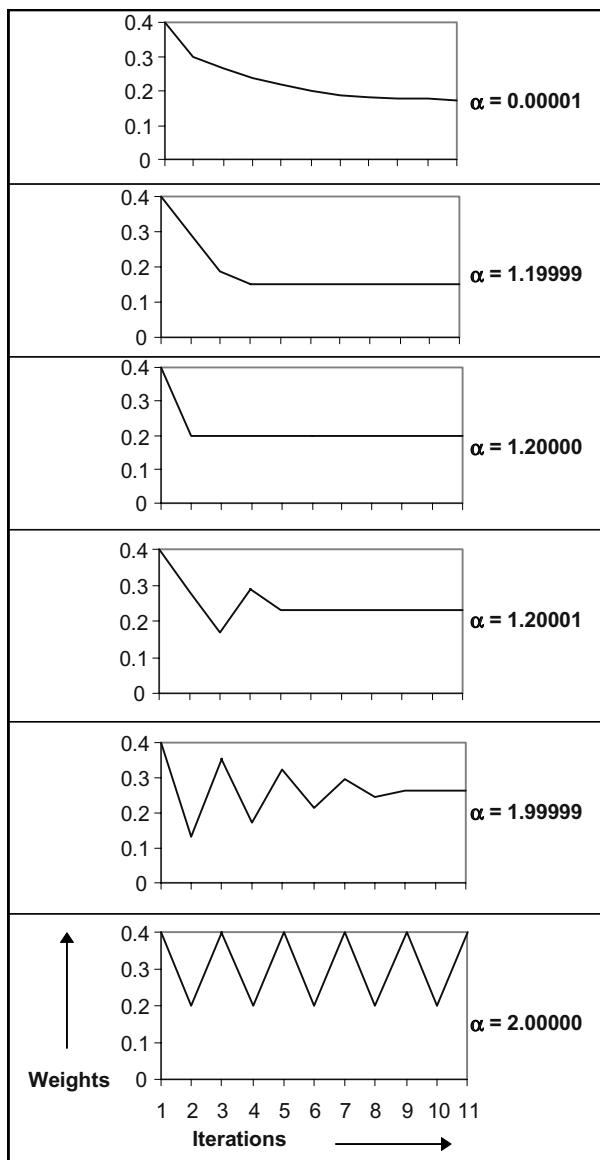
$$\varepsilon_k := d_k - [(1 + \delta m) \{ (Net_k \wedge m_2') \vee m_1 \} - (\delta m) m_1];$$

$$\text{Until } \varepsilon_k \leq \text{pre-assigned\_limit}_1$$

**End.**

Procedure Adjust-Adaline adapts the weight vector  $\mathbf{W}_k$  of an ADALINE based on the computed value of error  $\varepsilon_k$  at iteration  $k$ . The repeat-until loop in the algorithm continues adapting the weights until the error  $\varepsilon_k$  is below a prescribed limit. The procedure contains three main checks. If  $d_k < \sum x_i / n$ , a learning rule similar to Widrow-Hoff's delta learning rule is employed for adaptation of weights. An obvious question that naturally arises: why should we satisfy the above pre-condition for weight adaptation? The answer to this follows from the principle of signal transmission through a passive circuit. Since the desired signal  $d_k$  is less than the average value of the inputs, we can always generate  $out_k$  less than  $\sum x_i$ , and thus  $d_k$  is set accordingly.

If the sign of inequality is reversed, i.e.,  $d_k > \sum x_i / n$ , we need an amplifier with gain greater than one to enhance the average of inputs  $\sum x_i / n$  to  $d_k$ . This is realized by increasing the slope of AC in Fig. 17.2. But how can we set  $\delta m$ ? In procedure adjust-adaline, we updated  $\delta m$  as a linear function of error. So, when error is positive and increasing,  $\delta m$  also increases proportionately. But this increase of  $\delta m$  cannot generate an  $out_k$  with amplitude greater than  $m_2$ . So, when  $d_k > m_2$ , we assign  $m_2 = d_k$ . Similarly, if  $d_k < m_1$ , we extend the linearity by assigning  $m_1$  to  $d_k$ . The repeat until loop next computes the error  $\varepsilon_k$  from the measured values of  $d_k$ ,  $Net_k$  and  $\delta m$ . The procedure is continued until  $\varepsilon_k$  comes below a prescribed threshold.



**Fig. 17.3:** Effect of  $\alpha$  on dynamic behaviour of weights.

**Example 17.1:** The algorithm for training a single ADALINE neuron was simulated on a computer. The adaptation of one of the weights for different

value of  $\alpha$ , satisfying  $d_i < \sum x_i/n$  has been presented in Fig. 17.3. It is clear that as  $d_i < \sum x_i/n$ , no adaptation of slope is required. Thus the range of  $\alpha$  for stability of the system is  $0 < \alpha < 2$ , since  $\delta m = 0$ . It is evident from the figure that as  $\alpha \rightarrow 2.00000$ , the weight demonstrates temporal oscillatory behaviour. It may also be noted that the weights demonstrate underdamped, overdamped and critically damped behaviour for  $2 > \alpha > 1.2$ ,  $0 < \alpha < 1.2$  and  $\alpha=1.2$  respectively.

### 17.4.2 Training of the Neural Net

For a given input and output pattern vector  $\underline{x}$  and  $\underline{z}$  respectively, the components of  $\alpha$  are mapped at nodes  $x_i$  and  $z_j$  for all  $i, j$  (vide Fig. 17.4(a) & 17.4(b)). The algorithm for training the neural net requires updating of the weights and/or slopes of linear region of the ADALINEs in the network. The training algorithm may be initiated with arbitrary value of signals at the nodes of all intermediate layers. The algorithm for weight adaptation of all neurons may be executed concurrently as described below.

**Procedure Train\_Neuralnet**

**Begin**

Randomize output of all ADALINEs excluding those at the last layer in the interval [0,1];

**While** error of each neuron > preassigned limit  $\epsilon$

**Par Begin** //parallel adaptation begins//

Adjust weights and nonlinearity of each neuron by procedure

Adjust\_Adaline ;

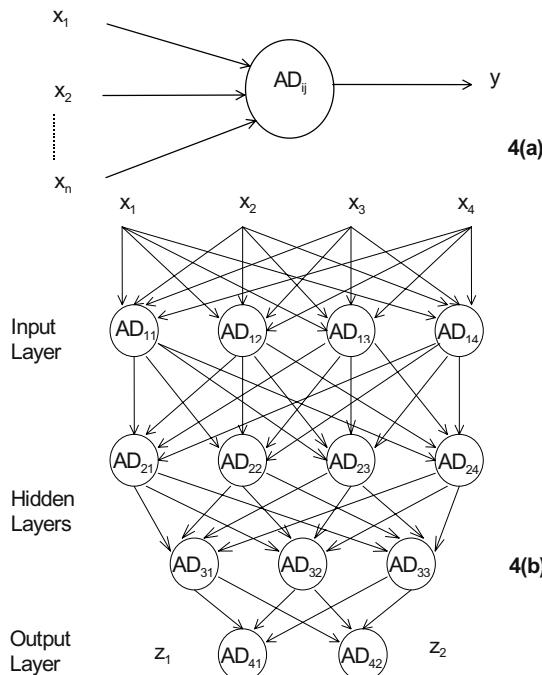
**Par End;** //parallel adaptation ends//

**End While;**

**End.**

Procedure Train\_neuralnet randomizes the desired value  $d_k$  of all neurons except those in the last layer. Naturally a question arises: why? This is because of the fact that the desired values of neurons at the last layer are already pre-fixed by the output components of the training instance. The desired values of the remaining neurons are irrespective of any input-output training instance. This, however, poses another important question: If the last layer only can take care of the re-generation of the target values at the output, can we then use a single layered net? The answer to this is definitely in the affirmative. But for multiple input-output training instances, a single layer may not be adequate.

Once the desired values of  $d_k$  for the neurons in the network are set, we can adapt the weights and non-linearity of all the neurons in parallel by adjust\_adaline procedure.



**Fig. 17.4(a):** Symbolic representation of Fuzzy ADALINE  $AD_{ij}$  with inputs  $[x_1, x_2, \dots, x_n]^T$  and output  $y$ . **Fig. 17.4(b):** A typical feed-forward neural net of Fuzzy ADALINE with hidden layers.

### 17.4.3. Training with Multiple Input-Output patterns

The algorithm for training the neural net with  $m$ -sets ( $m > 1$ ) of input-output patterns is presented below. The algorithm yields a set of steady-state weights and non-linearity for each ADALINE of the network.

```

Procedure Train_with _multiple_I / O_patterns;
randomize output of all ADALINEs, excluding those in the last layer in [0,1];
For r := 1 to max_pass do // r = pass_number //
Begin
Repeat
   $P\ I_r := 0$ ; //  $P\ I_r$  ≡ performance index in r-th pass //
  For k := 1 to m do
    Begin

```

---

**For** the k-th pair of I / O patterns **do**  
**Begin**

- (1) Adjust weights and nonlinearity of all ADALINEs once only;
- (2) Estimate error vector  $\underline{\mathcal{E}_k}$  that includes components of error ( $\epsilon_j$ ) of all ADALINEs in the network;
- (3)  $S_k := \sum_{\forall j} \mathcal{E}_j^{1/3}$ , where  $\mathcal{E}_j$  is the j-th component of  $\underline{\mathcal{E}_k}$ ;

**End For;**  
 $P I_r := P I_r + S_k$  ;  
**End For;**  
**Until**  $P I_r - P I_{r-1} < \text{pre\_assigned\_positive\_limit}$ ;  
**End For;**  
**End.**

Procedure *Train-with-multiple-I/O-patterns* includes 3 ‘for loops’. The innermost for loop takes care of the k-th input-output training instance. The loop allows adaptation of weights and non-linearity of neurons once only and evaluates a scalar  $S_k$  by taking the sum of the cube-roots of error at the output layer. The cube-root is taken to increase the amplitude of error with proper sign. The middle ‘for loop’ repeats the above process for  $k= 1$  to  $m$  number of training instances. The outer for loop keeps track of performance index and determines termination of the program based on the difference of  $PI_r$  and  $PI_{r-1}$ .

It is important to note that steady state value of weights and nonlinearity of the ADALINEs depend largely on their initial random output values  $Out_k$ . Can we then design a strategy to select the initial values of the outputs so as to improve the performance index? The following scheme suggests a solution to the said problem.

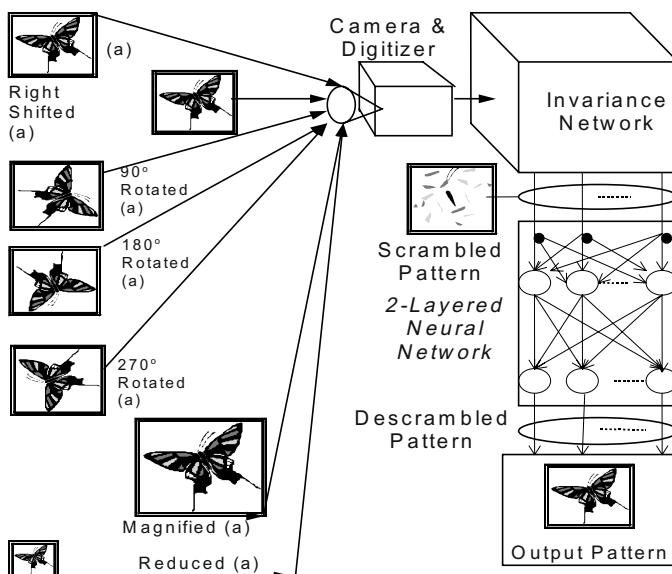
One may run procedure *Train-with-multiple-I/O-patterns* for different sets of randomized output patterns of ADALINE and then save the steady-state value of weights and nonlinearity for that set of random pattern for which the performance index ( $PI_r$ ) is the smallest. A simple way to handle the above problem might be to employ a Genetic Algorithm to select the randomized output of ADALINEs for minimizing  $PI_r$ .

## 17.5. Translation, Rotation and Size Invariant Gray Pattern Recognition

In the first step, we design an invariance network which transforms a shifted, rotated or scale changed image into a unique scrambled image pattern. Such scrambled pattern may be descrambled with the help of a trained neural net of the proposed topology to reproduce the original patterns in "standard" position, orientation, scale, etc. The overall scheme of the system is presented in Fig. 17.5.

The invariance network consists of a number of planes of ADALINE (vide Figures 17.6 & 17.8) [13], each of which receives signals from the retinal plane Fig. 17.8.

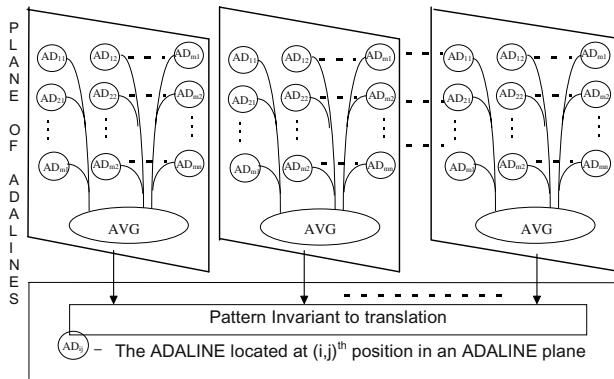
The weights assigned to the input terminals of an ADALINE are taken together to form a matrix, called weight matrix. Each retinal plane is assumed to correspond to a number of such weight matrices.



**Fig. 17.5:** The overall scheme of gray pattern recognition. Each pattern is applied once. Scrambled Pattern is insensitive to variation of input patterns. The output pattern is unique for every input pattern.

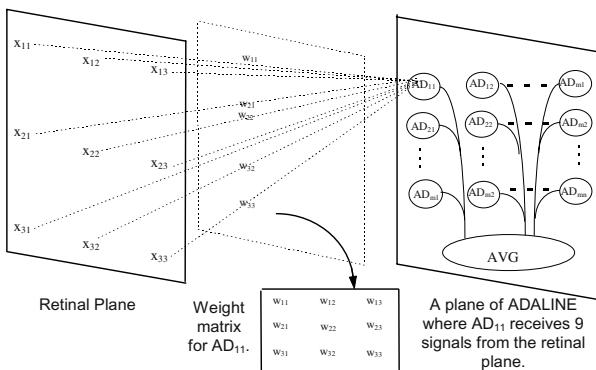
### 17.5.1. Design of Translational Invariance Network

Let us assume that a 2D view of an object moves horizontally in front of the retinal plane but remains within the field of view of the plane.



**Fig. 17.6:** The Invariance Network.

It is further assumed that each weight matrix in the plane has  $p \times p$  pixels, whereas the entire plane consists of  $n \times n$  pixels, where  $n \gg p$ .

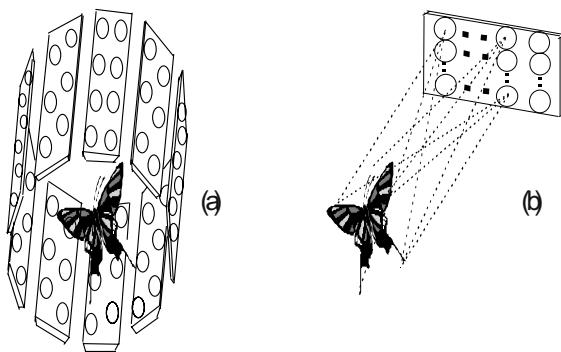


**Fig. 17.7:** Retinal plane and weight matrices for ADALINES.

For translational invariance, we organize, following Widrow, the weight matrices of all ADALINES, first by selecting the weight matrix for ADALINE  $AD_{11}$  and then constructing the other weight matrices by the following

procedure. For  $AD_{ij}$ , the weight matrix should be horizontally rolled right and vertically down by i and j position respectively compared to  $AD_{11}$ .

Now if the 2D view of the object shifts right horizontally by m pixels for  $p < m < 2p$ , then the ADALINE closest to the right terminal of the 2D view will yield more signal strength and the ADALINE closest to the left terminal of the 2D view will have a decrease in output signal strength. As a consequence, the average of the output of the ADALINEs remains constant. We, therefore, connect an AVG module to take the average of the output of the ADALINEs. The details of each ADALINE plane are given in Fig. 17.7 and the geographical orientation of the ADALINE planes in the invariance network is given in Fig. 17.8.



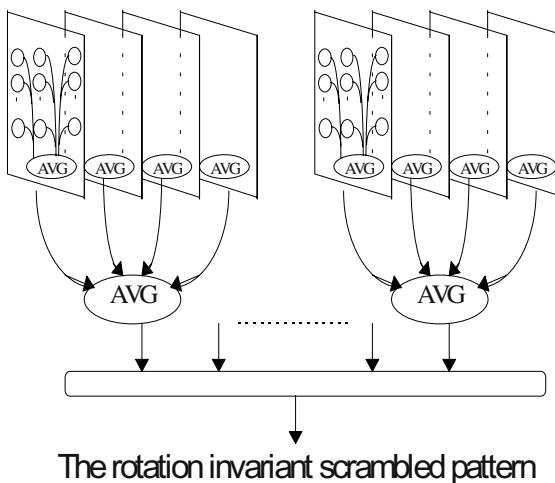
**Fig. 17.8(a):** The geographical orientation of the ADALINE planes in the invariance network, where each ADALINE of each plane can receive signals from the butterfly as in Fig. 17.8(b).

### 17.5.2 Design of Rotational Invariance Network

For  $90^\circ$ ,  $180^\circ$ ,  $270^\circ$  rotational invariance we group, following Widrow [6], 4 planes of ADALINEs and thus resolution of the system is reduced to 1/4. The outputs of 4 ADALINEs within one group are passed through an AVG module. The weight matrices of  $AD_{11}$  of the 2nd, 3rd and the 4th planes are constructed by rotating the weight matrix of  $AD_{11}$  under the first plane by  $90^\circ$ ,  $180^\circ$ ,  $270^\circ$  with respect to a fixed pivot. The other weight matrices under each plane of ADALINE are designed satisfying translational invariance. The complete scheme for the system is given in Fig. 17.9.

### 17.5.3. Design of Size Invariance Network

For size invariance recognition of gray image, we need not make any modification to Widrow Hoff's scheme.



**Fig. 17.9:** Generation of rotation invariant scrambled pattern.

## 17.6. Conclusions

Widrow-Hoff's ADALINE based scheme for recognition of binary images has been extended in this chapter to make it appropriate for gray images. For gray input-output patterns, both weights and non-linearity of the ADALINE require adaptation unlike Widrow's scheme, where only the weights require adaptation. The slope of the inhibiting two-sided clipper-type non-linearity requires enhancement, when  $Net_i > d_i$ . This, however, causes a further reduction in the range of  $\alpha$  for attaining stability. Under extreme cases, when  $\delta m$  approaches 1,  $\alpha (= 2/(1+\delta m))$  becomes unity, i.e. the permissible range of  $\alpha$  for stability of the system is  $0 < \alpha \leq 1$ .

The adaptation of weights and non-linearity of each neuron in the proposed neural net can be carried out in parallel, thereby reducing the training time significantly.

The proposed scheme for training with multiple input-output patterns has been simulated on IBM PC/AT and the results thus obtained support the theoretical foundations developed in this chapter.

## Exercise

1. Find a solution for the difference equation:

$$\Delta \epsilon_k = -\alpha \epsilon_k (1 + \delta m).$$

[**Hints:**  $\Delta \epsilon_k = (E - 1)\epsilon_k$

$$\therefore [E - 1 + \alpha (1 + \delta m)] \epsilon_k = 0$$

$$\Rightarrow E = 1 - \alpha (1 + \delta m)$$

So, complementary function is given by

$$\epsilon_k = A [1 - \alpha (1 + \delta m)]^k$$

At  $k = 0$ ,  $\epsilon_k = 0$ , and then  $A = \epsilon_0$ .

$$\therefore \epsilon_k = \epsilon_0 [1 - \alpha (1 + \delta m)]^k.$$

2. Given the delta learning rule, find its equivalent continuous version model and level determine the condition for consequence.

[**Solution:** The discrete rule is

$$\Delta \epsilon_k = -\alpha \epsilon_k.$$

The equivalent continuous model is given by

$$\frac{d\epsilon_k}{dk} = -\alpha \epsilon_k$$

$$\Rightarrow \frac{d\epsilon_u}{\epsilon_u} = -\alpha$$

$$\Rightarrow \ln(\epsilon_k) = -\alpha k$$

$$\Rightarrow \epsilon_k = A \exp(-\alpha k)$$

When  $\alpha = 0$ ,  $A = \epsilon_0$ .

$$\therefore \epsilon_k = \epsilon_0 \exp(-\alpha k). ]$$

3. The solutions of discrete and continuous delta learning rules are  $\epsilon_0 (1 - \alpha)^k$  and  $\epsilon_0 \exp(-\alpha k)$  respectively. When are they equal?

[**Hints:** Expanding  $\exp(-\alpha k)$  by McClurin's series we have:

$$\exp(-\alpha k) = 1 - \alpha k + (\alpha k)^2 / (2!) - (\alpha k)^3 / (3!) + \dots + (-1)^n - (\alpha k)^n / (n!)$$

Again,

$$\begin{aligned} (1 - \alpha)^k &= (1)^k - {}^k c_1 (1)^{k-1} \alpha + {}^k c_2 (1)^{k-2} \alpha^2 - {}^k c_3 (1)^{k-3} \alpha^3 + \dots + (-1)^n - {}^k c_n (1)^{k-n} \alpha^n \\ &= 1 - \alpha k + (k! / (k-2)! 2!) \alpha^2 - (k! / (k-3)! 3!) \alpha^3 + \dots + (-1)^n (k! / (k-n)! n!) \alpha^n \\ &= 1 - \alpha k + (k(k-1)/ 2!) \alpha^2 - (k(k-1)(k-2)/ 3!) \alpha^3 + \dots + (-1)^n k(k-1)(k-2)\dots(k-n+1)/ n! \alpha^n \end{aligned}$$

Thus for large  $k$ ,  $(k-1)$ ,  $(k-2)$ , ...,  $(k-n+1)$  all approach  $k$ . Consequently, the above expression reduces to

$$\begin{aligned} 1 - \alpha k + (\alpha k)^2 / (2!) - (\alpha k)^3 / (3!) + \dots + (-1)^n (\alpha k)^n / (n!) \\ = \exp(-\alpha k). \end{aligned}$$

In summary, the solution to continuous delta learning approaches the solution to discrete learning for infinitely large  $k$ .]

## References

- [1] Fiesler, E. and Beale, R., *Handbook of Neural Computation*, IOP Publishing Ltd., Oxford University Press, Amsterdam, Oxford, 1997.

- 
- [2] Hui, S. and Zak, S. H., "The Widrow-Hoff algorithm for McCulloch-Pitts type neuron," *IEEE Trans. on Neural Networks*, vol. 5, no.6, pp. 924-929, 1994.
  - [3] Lau, C., *Neural Networks: Theoretical Foundations and Analysis*, IEEE Press, New York, 1992.
  - [4] Lee, J., Nguyen, D. and Lin, C., "Adaptive object tracking: integrating neural networks and intelligent processing," *Neural Networks (Supplement: INNS Abstracts)*, vol. 1, p. 590, 1988.
  - [5] Lucky, R. W., "Automatic equalization for digital communication," *Bell Systems Tech. J.*, vol. 44, pp. 547-588, 1965.
  - [6] McCulloch, W. S. and Pitts, W., "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophysics.* vol. 5, pp. 115-133, 1943.
  - [7] Millman, J. and Taub, H., *Pulse, Digital and Switching Circuits*, McGraw-Hill, New York, 1981.
  - [8] Paul, B., Konar, A. and Mandal, A. K., "Fuzzy ADALINEs for gray image recognition," *Neurocomputing*, Elsevier, vol. 24, pp. 207-223, 1999.
  - [9] Rumelhart, D. E., McClelland, J. L., *Parallel and Distributed Processing*, vol. I and II, MIT Press, Cambridge, M.A., 1986.
  - [10] Widrow, B. and McCool, J., "A comparison of adaptive algorithms based upon the methods of steepest descent and random search," *IEEE Trans. Antennas and Propagation*, AP-24, pp. 615-637, 1976.
  - [11] Widrow, B. and Hoff., M., Adaptive Switching Circuits, *WESCON Convention Record*, Part -4, pp. 96-104, 1960.
  - [12] Widrow, B. and Stearns, S., *Adaptive Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1962.
  - [13] Widrow, B, Winter, R., "Neural nets for adaptive filtering and adaptive pattern recognition," *IEEE Computer*, pp. 25-39, 1988.
  - [14] Widrow, B, Winter, R. and Baxrwe, R., "Layered neural networks for pattern recognition," *IEEE Trans. Acoustics, Speech and Signal Processing*, AASP-36, pp. 1109-1118, 1988.

# 18

## Distributed Machine Learning Using Fuzzy Cognitive Maps

*Mammals perform spatial reasoning by a specialized structure called cognitive maps located in the hippocampus region of their forebrain. In the treatises of computational intelligence, the phrase cognitive maps, however, has a wider meaning. It includes encoding of knowledge about causal events and their automated recall. Modeling of cognitive maps by fuzzy logic is apparent because of the inherent fuzziness of most real world knowledge bases. The chapter provides a thorough overview of various models of cognitive maps and their learning behavior. The dynamics of the learning models have been analyzed to determine the condition for their stability. The chapter ends with a discussion on the scope of application of the proposed models in practical engineering systems.*

### 18.1 Introduction

The word ‘cognition’ generally refers to a faculty of mental activities dealing with abstraction of information from a real world scenario, their representation

and storage in memory and automatic recall [1-15], [31], [34]. It also includes construction of high-level percept [33] from primitive/low level information/knowledge, hereafter referred to as perception. There is plenty of literature dealing with psychological models of cognition [16-22]. The models include representation of human thoughts on memory [24-28], [36] iconic and echoic representation [21-29], [31-32], [34-35] of knowledge, part and whole relationship of objects in human memory, understanding complex problems [24] and many others.

In the treatises of brain sciences, the phrase *cognitive map* has a traditional meaning, referring to a specialized structure of the mammalian brain. Computational scientists, however, use the phrase cognitive map in a lucid sense to represent cause-effect relationships of events in a knowledge base. For instance, *a unit rise in temperature will cause a severe degree of malfunctioning in the system* can easily be modeled with a cognitive map [30], which by classical models of knowledge representation cannot be described for their limitation in representing *causal* relations.

*Fuzzy cognitive map* has very recently been introduced in the fields of machine intelligence. Coined by Bart Kosko, the phrase *fuzzy cognitive map* refers to a graph theoretic structure capable of encoding knowledge by employing the logic of fuzzy sets. The chapter outlines different models of fuzzy cognitive maps and their learning paradigm.

The chapter has been divided into 8 sections. Section 18.2 outlines the Axelrod's model of cognitive maps. The extension of Axelrod's model by Bart Koko is presented in section 18.3. Further extensions of Kosko's classical model are presented in section 18.4 and 18.5. Zhang, Chen and Bezdek's model is presented in section 18.6. Pal and Konar's model is outlined in section 18.7. Conclusions are listed in section 18.8.

## 18.2 Axelrod's Cognitive Maps

Axelrod (vide [19]) introduced cognitive maps in 1970s for representing social scientific knowledge. These maps are signed directed graphs where "nodes" denote concepts (like "social instability") and the directed edges denote causal connections. A positive or negative sign is attached with the edges to denote causal increase or decrease respectively. For instance, a positive (negative) edge from node A to node B implies that A causally increases (decreases) B.

### Axelrod's Model

Axelrod employed adjacency matrix to represent cognitive maps. Let  $e_{ij}$  be an edge describing the causal relation from concepts  $c_i$  to  $c_j$ . Then

$e_{ij} = 1$ , if  $c_i$  causally increases  $c_j$ ,  
 $= -1$ , if  $c_i$  causally decreases  $c_j$ ,  
 $= 0$ , if  $c_i$  imparts no causality to  $c_j$ .

Thus for a given map, shown in Fig. 18.1, we get the adjacency matrix E.

		To	$c_1$	$c_2$	$c_3$	$c_4$	$c_5$	$c_6$
		From	$c_1$	0	-1	1	0	0
E=		$c_2$	0	0	0	1	0	0
		$c_3$	0	0	0	0	1	0
		$c_4$	0	0	0	0	0	-1
		$c_5$	0	0	0	-1	0	-1
		$c_6$	0	0	0	0	0	0

$c_1, c_2, \dots, c_6$  in the above adjacency matrix correspond to the concepts in Fig. 18.1.

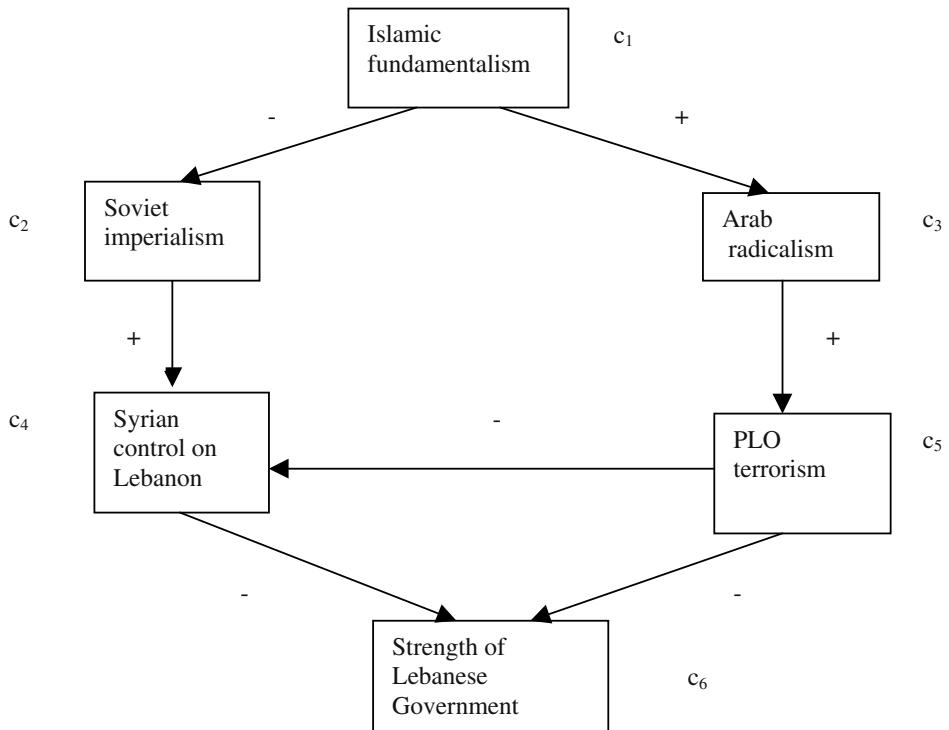
A question then naturally arises: what is the utility of the adjacency matrix E? Suppose, we want to see the causal effect of  $c_1$  and  $c_4$  jointly; in that case we can multiply the row vector C given below by the matrix E.

$$C = [ 1 \ 0 \ 0 \ 1 \ 0 \ 0 ] \\ c_1 \ c_2 \ c_3 \ c_4 \ c_5 \ c_6$$

Then

$$C E = [ 0 \ -1 \ 1 \ 0 \ 0 \ -1 ] \\ c_1 \ c_2 \ c_3 \ c_4 \ c_5 \ c_6$$

which indicates the effect of  $c_1$  on both  $c_2$  &  $c_3$  and the effect of  $c_4$  on  $c_6$ .



**Fig. 18.1:** A cognitive map describing the political relations for Middle East peace.

### 18.3 Kosko's Model

Kosko [19] formalized the definition of causality in an FCM. According to him,

Let

$c_i$  = a concept, and

$Q_i$  = a fuzzy linguistic set (Much  $c_i$ , More or less  $c_i$  etc.) of  $c_i$ .

Then for any 2 concepts  $c_i$  and  $c_j$ ,  $c_i$  causes  $c_j$  iff

- i)  $Q_i \subset Q_j$  and  $\neg Q_i \subset \neg Q_j$
- ii)  $Q_i \subset \neg Q_j$  and  $\neg Q_i \subset Q_j$

where “ $\subset$ ” stands for logical implication. In rule (i)  $c_i$  causally increases  $c_j$ , whereas in rule(ii)  $c_i$  causally decreases  $c_j$ .

**Example 18.1:** Let  $c_1$  = Islamic fundamentalism,  $c_2$  = Soviet imperialism,  $c_3$  = Arab radicalism. Then vide Fig. 18.1, we have

$$\begin{array}{ccc} + & & - \\ c_1 \rightarrow c_3 & \text{and} & c_1 \rightarrow c_2 \end{array}$$

where labels + or - represent positive or negative causality.

Suppose, we want to express the knowledge:

- 1) *Extensive Islamic fundamentalism increases massive Arab radicalism.*
- 2) *Extensive Islamic fundamentalism causes a severe fall in Soviet imperialism.*

Let  $Q_1$ = Extensive,  $Q_2$  = Severe and  $Q_3$ = Massive. Then the above 2 rules by the definition of causality should satisfy:

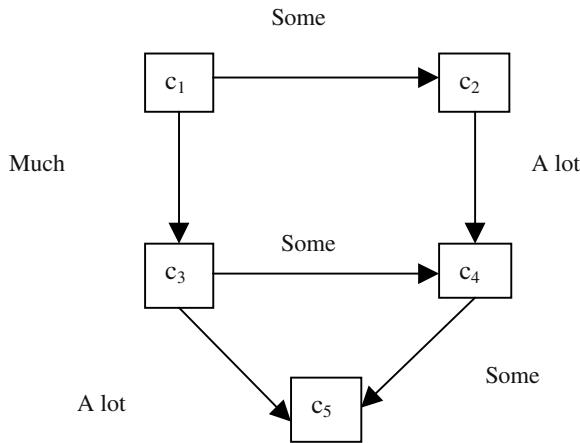
1.  $Q_1(c_1) \subset Q_3(c_3)$  and  
 $\neg Q_1(c_1) \subset \neg Q_3(c_3);$
2.  $Q_1(c_1) \subset \neg Q_2(c_2)$  and  
 $\neg Q_1(c_1) \subset Q_2(c_2).$

In extending FCMs, Kosko considered fuzzy labels of the edges in the map. According to him, if the labels satisfy a *partial ordered relation* with respect to  $\leq$  operator, then we can evaluate the causal effect of one node on the desired node. For instance, let

$$P = \{\text{none} \leq \text{some} \leq \text{much} \leq \text{a lot}\}$$

be the *partially ordered set* of attached label of the edges. Then for the given map (vide Fig. 18.2) we find the causal effect of  $c_1$  over  $c_5$  through 3 paths:

1.  $c_1 - c_2 - c_4 - c_5$
2.  $c_1 - c_3 - c_5$
3.  $c_1 - c_3 - c_4 - c_5.$



**Fig. 18.2:** A cognitive map with fuzzy labels at the edges.

Here, the causal effect of  $c_1$  over  $c_5$  is determined by taking the minimum of the attached labels of the individual paths. Let  $I_1$ ,  $I_2$  and  $I_3$  denote the effect of  $c_1$  on  $c_5$  through the paths 1 to 3 respectively, and  $e_{ij}$  be the label attached with edge from node  $i$  to node  $j$ . Then

$$\begin{aligned} I_1(c_1, c_5) &= \text{Min}\{e_{12}, e_{24}, e_{45}\} \\ &= \text{Min}\{\text{Some}, \text{A lot}, \text{Some}\} \\ &= \text{Some}, \end{aligned}$$

$$\begin{aligned} I_2(c_1, c_5) &= \text{Min}\{e_{13}, e_{35}\} \\ &= \text{Min}\{\text{Much}, \text{A lot}\} \\ &= \text{Much}, \end{aligned}$$

$$\begin{aligned} I_3(c_1, c_5) &= \text{Min}\{e_{13}, e_{34}, e_{45}\} \\ &= \text{Min}\{\text{Much}, \text{Some}, \text{Some}\} \\ &= \text{Some}. \end{aligned}$$

To determine the total effect of  $c_1$  on  $c_5$ , we take the maximum of  $I_1$  through  $I_3$ . Thus, total effect of  $c_1$  on  $c_5$ , denoted by  $T(c_1, c_5)$  is computed below:

$$\begin{aligned} T(c_1, c_5) &= \text{Max}\{I_1(c_1, c_5), I_2(c_1, c_5), I_3(c_1, c_5)\} \\ &= \text{Max}\{\text{Some}, \text{Much}, \text{Some}\} \\ &= \text{Much}. \end{aligned}$$

In words,  $c_1$  imparts *much* causality to  $c_5$ .

## 18.4 Kosko's Extended Model

Kosko extended the elementary model of Axelrod by including a non-linear function in the model [20]. To be specific, let  $E_{(n \times n)}$  be the incidence matrix of an FCM, and  $C = [c_1 \ c_2 \ \dots \ c_n]$  be the given state vector for the FCM. Here  $c_i$ , the  $i$ -th component of vector  $C$  denotes the strength of concept  $c_i$ . Then the next state vector can be evaluated by

$$C(t+1) = S [ C(t) \cdot E ] \quad (18.1)$$

where

$S$  is a non-linear function applied over the individual components of the vector-matrix product;

and  $t$  denotes the time.

The incorporation of non-linearity, sometimes, forces the cognitive map to recycle through states.

**Example 18.2:** Let  $E$  be the incidence matrix for a given FCM of 5 concepts:  $c_1, c_2, c_3, c_4, c_5$ . Let

		To				
		$c_1$	$c_2$	$c_3$	$c_4$	$c_5$
From	$c_1$	0	1	0	-1	0
	$c_2$	0	0	1	0	-1
	$c_3$	0	-1	0	1	-1
	$c_4$	1	0	-1	0	1
	$c_5$	-1	1	0	-1	0

Let  $S$  be a binary hard type non-linearity given by

$$\begin{aligned} S(a) &= +1 \text{ for } a > 0, \\ &= 0 \quad \text{for } a \leq 0. \end{aligned}$$

Then for a given initial state:

$$C_1 = [0 \ 0 \ 0 \ 1 \ 0],$$

We find limit cyclic behavior of the system through the following states:

$$c_1 - c_2 - c_3 - c_4 - c_1.$$

## 18.5 Adaptive FCMs

Kosko further extended his FCM by incorporating adaptive learning behavior of their edges [20]. Let  $e_{ij}$  be the fuzzy strength of an edge from node  $i$  to node  $j$ . Let  $x_i$  and  $x_j$  be the signal strength of concept  $c_i$  and  $c_j$  respectively. Let  $S$  be a non-linear function. Then the Hebbian type learning law used to adapt the edge strength  $e_{ij}$  is given below:

$$\frac{de_{ij}}{dt} = -\beta e_{ij} + S(x_i) \cdot S(x_j), \beta > 0 \quad (18.2)$$

where

$$S(x_k) = 1 / [1 + \exp(-\alpha x_k)], \alpha > 0, k \in \{i, j\} \quad (18.3)$$

is a Sigmoid function.

The  $-\beta e_{ij}$  term in expression (18.2) denotes a natural decay in the fuzzy edge strength, and the second term in the right side ensures strengthening of edge connectivity with the nodes.

A discrete version of equation (18.2) is given by

$$e_{ij}(t+1) = (1 - \beta) e_{ij}(t) + S(x_i) \cdot S(x_j). \quad (18.4)$$

For a given  $e_{ij}(0)$ , we can iterate the above expression until  $e_{ij}$  attains a steady value  $e_{ij}^*$ . Once  $e_{ij} = e_{ij}^*$  for all  $i, j$ , the encoding process is terminated. For recall in an FCM, Kosko used equation (18.1), as discussed earlier. The recall model, however, gets trapped within limit cycles.

Among the other adaptive learning equations, the differential Hebbian learning law presented below needs mention:

$$\frac{de_{ij}}{dt} = -\beta e_{ij} + \frac{dS(x_i)}{dt} - \frac{dS(x_j)}{dt} \quad \text{for } \beta > 0 \quad (18.5)$$

The advantage of differential Hebbian model over the Hebbian model lies in learning higher order causal relations in case it correlates multiple cause changes with effect changes.

The discrete version of the model has successfully been used for encoding edge strengths in “virtual reality” systems. This encoding model too is used in conjunction with the recall model (18.1), which because of its limit cyclic behavior, helps cycling the events in virtual reality systems.

## 18.6 Zhang, Chen and Bezdek’s Model

Zhang et al. [40] in late 1980s presented a novel scheme for cognitive reasoning using FCM. They defined **Negative Positive Neutral (NPN)** logic for both crisp and fuzzy variables. In case of a crisp variable, NPN logic allows a valuation space of  $\{-1, 0, 1\}$ , whereas for a fuzzy variable the valuation space is  $[-1, 1]$ . The following operations are essentially needed to use this logic.

$$1. \quad x * y = \text{Sign}(x) \text{Sign}(y) (|x| * |y|) \quad (18.6)$$

where  $*$  is any T-norm in  $[-1, 1]$ . Typically  $*$  is the fuzzy AND (min) operator.

$$2. \quad (x, y) * (u, v) = [\min(x*u, x*v, y*u, y*v), \\ \max(x*u, x*v, y*u, y*v)] \quad (18.7)$$

$$3. \quad (x, y) \text{ OR } (u, v) = [\min(x, u), \max(y, v)]. \quad (18.8)$$

In NPN logic based FCM, the edges are labeled with a doublet  $(u, v)$ , where  $u$  and  $v$  are the minimum and the maximum fuzzy strength of the edge. For formalization, let us consider a fuzzy relation  $R$  in  $X \times Y$  where  $X = \{x_i\}$  and  $Y = \{y_j\}$  are finite sets. Then  $\mu_R(x_i, y_j)$  is the membership of the edge connecting nodes  $x_i$  and  $y_j$ . Now, suppose for a given edge  $e_{ij}$ , a set of experts assigned different values of  $\mu_R(x_i, y_j)$ . Let there be  $k$  number of experts,  $m$  of which assigned negative values  $u_1, u_2, \dots, u_m$  say, and  $(k - m)$  number of experts assigned positive values  $v_1, v_2, \dots, v_{k-m}$  say. Then we define

$$a = \sum u_i / m \quad (18.9)$$

$$\text{and} \quad b = \sum v_i / (k - m) \quad (18.10)$$

and attach  $(a, b)$  as the label of the edge  $e_{ij}$ .

One important characteristic of NPN relation is the transitivity. Let  $X = \{x_1, x_2, \dots, x_n\}$  be a finite set. An NPN relation  $R$  is **transitive** iff for all  $i, j, k$ , such that  $0 < i, j, k \leq n$ ,

$$\mu_R(x_i, x_k) \geq \max_{x_j} [\mu_R(x_i, x_j) * \mu_R(x_j, x_k)] \quad (18.11)$$

The  $(\max \cdot *)$  composition of 2 NPN relations  $R \subseteq (X \times Y)$  and  $Q \subseteq (Y \times Z)$ , denoted by  $R \circ Q$  is defined below:

$$\mu_{R \circ Q} = \max [\mu_R(x, y) * \mu_Q(y, z)] \text{ for } x \in X, y \in Y \text{ and } z \in Z \quad (18.12)$$

In the last expression max is equivalent to OR. The  $n$ -fold composition of  $R$  is denoted by  $R^n$ , where

$$R^n = R \circ R \circ \dots \circ R \text{ (n-times).}$$

The transitive closure  $R^{TC}$  of an NPN relation  $R$  in  $X$  is the smallest  $(\max \cdot *)$  transitive NPN relation containing  $R$ . It can be shown that for  $R_{(n \times n)}$ ,

$$R^{TC} = R^1 + R^2 + \dots + R^{2^n} \quad (18.13)$$

Zhang et al. categorized their work into 3 components: i) cognitive map building, ii) cognitive map understanding, and iii) decision making. The first component is concerned with the fusion of multiple experts' opinion to determine the doublet of the fuzzy edges. The second component deals with determination of the heuristic transitive closure (HTC) of an NPN relation by a HTC algorithm, and finds 2 most effective paths between any 2 elements of the FCM by invoking a heuristic path (HP) finding algorithm. Given  $\mu_R(i, k) = (x, y)$  and  $\mu_R(k, j) = (u, v)$  say, the HTC algorithm determines

$$\mu_R(i, j) = (x, y) * (u, v) \quad (18.14)$$

for all  $i, j$  and  $k = 1$  to  $n$ . Once the computation is over, the algorithm evaluates

$$R^{TC} = R \circ ([I] + R) \quad (18.15)$$

where  $R = [\mu_R(i, j)]_{(n \times n)}$ .

The HP algorithm on the other hand explores depth first search on the graph to determine the most effective paths that cause a positive maximum and a negative minimum effect.

The third component is needed to answer the users' query from the resulting outcome of the second phase. Suppose, for instance, the second phase returns a label (-0.3, 0). Then the answer to the query: "whether there is any positive causal effect of  $x_i$  over  $x_j$ ?" should be answered in the negative.

## 18.7 Pal and Konar's FCM Model

Pal and Konar presented an alternative model [29] of fuzzy cognitive map with a slightly different nomenclature. According to them a cognitive map is an associative structure consisting of nodes and directed arcs where the nodes carry fuzzy beliefs and the arcs or edges carry connectivity strength.

Let

$n_i(t)$  and  $n_j(t)$  be the fuzzy beliefs of node  $N_i$  and  $N_j$  respectively and

$\delta w_{ij}(t)$  be the change in connectivity strength (CS)  $w_{ij}$  of an edge  $E_{ij}$  connected from node  $N_i$  to  $N_j$ .

Then following the principles of Hebbian learning and considering self-mortality of  $w_{ij}$ , we can write

$$\delta w_{ij}(t) = -\alpha w_{ij}(t) + S(n_i(t)) \cdot S(n_j(t)) \quad (18.16)$$

where

$$S(n_k) = 1 / [1 + \exp(-n_k)] \quad \text{for } k = \{i, j\} \quad (18.17)$$

and  $\alpha$  represents the forgetfulness (mortality) rate.

It may be noted that the encoding model (18.16) is similar with Kosko's model (18.2). But there is a significant difference between the recall models of Pal & Konar with that of Kosko. The recall model of Pal & Konar in point-wise notation is given below.

$$n_i(t+1) = \text{Max}[n_i(t), \text{Max}_k \{(n_k \text{ Min } w_{ki})\}] \quad (18.18)$$

The above equation states that the cognitive system will attempt to restore the last value of  $N_i$  through a memory refresh cycle. The fuzzy belief of node  $N_i$  thus will assume either of i) the last value or ii) the supremum of  $(n_k, w_{ki})$ , whichever is larger.

To study the stability of the cognitive map Pal and Konar [30] derived some interesting results presented below.

**Property 1:** For all  $t$ ,  $n_i(t+1)$  in expression (18.18) is bounded between 0 and 1.

*Proof:* Let us consider three possible ranges of  $w_{ki}$  : (i)  $w_{ki} < 0$ , (ii)  $0 < w_{ki} \leq 1$ , (iii)  $w_{ki} > 1$ .

*Case I:* When  $w_{ki}(t) < 0$ , for  $\exists(\forall)k$ , let us assume that  $\max_k \{\min(n_k(t), w_{ki}(t))\} < 0$ . However, since  $0 < n_i(0) < 1$ , recursive use of expression (18.18) reveals that  $\forall t$ ,  $0 < n_i(t+1) < 1$ .

*Case II:* When  $0 < w_{ki}(t) < 1$ ,  $\exists(\forall)k$ , the proof of property 1 is obvious.

*Case III:* When  $w_{ki}(t) > 1$ ,  $\exists(\forall)k$ , since  $0 < n_k(0) < 1$ ,  $0 < \min(n_k(0), w_{ki}(0)) < 1$ . So  $\max_k \{\min(n_k(0), w_{ki}(0))\} < 1$ . Again since  $0 < n_i(0) < 1$  by expression (18.18),  $0 < n_i(1) < 1$ . Property 1 thus can easily be proved by the method of induction.

Property 1, therefore, holds for all the above cases.

**Theorem 18.1:** The recall model given by expression (18.18) is unconditionally stable.

*Proof:* It is clear from expression (18.18) that  $n_i(t+1) \geq n_i(t)$  for all  $t$ . Since an oscillation requires both an increase and decrease in value and  $n_i(t+1)$  is never less than  $n_i(t)$ , for any  $t$ , therefore,  $n_i(t+1)$  cannot exhibit oscillatory time-response. Further, since  $0 < n_i(t+1) < 1$  for all  $t$ , vide property 1, and  $n_i(t)$  does not exhibit oscillation, so  $n_i(t)$  must reach steady state with value between 0 and 1.

Since the model represented by expression (18.18) need not require to satisfy any condition for its stability, the statement of the theorem follows.

Corollary 1 follows from the above theorem. In this corollary, we consider a node vector  $\mathbf{N}$ , the  $i$ -th component of which represents the fuzzy belief of node  $N_i$ .

**Corollary 18.1:** Given a node vector  $\mathbf{N}$  such that its  $i$ -th component corresponds to the belief of node  $N_i$  for all  $i$ . Also given an edge connectivity matrix  $\mathbf{W}$  whose  $(i, j)$ -th component  $w_{ij}$  denotes connectivity strength from node  $N_j$  to node  $N_i$ . The condition for steady-state for the node vector  $\mathbf{N}$  then satisfies the following inequality:

$$\mathbf{N}^* \geq \mathbf{W}^{*T} \circ \mathbf{N}^* \quad (18.19)$$

where  $\mathbf{N}^*$  and  $\mathbf{W}^*$  represent the steady-state values of  $\mathbf{N}$  and  $\mathbf{W}$  respectively.

*Proof:* The vector-matrix form corresponding to expression (18.18) is given by

$$\mathbf{N}(t+1) = \mathbf{N}(t) \vee (\mathbf{W}^T \circ \mathbf{N}(t)) \quad (18.20)$$

where  $\vee$  and  $\circ$  denote fuzzy or (max) and max-min composition operators respectively. The  $\vee$  operator between two vectors computes the component-wise maximum of the two vectors like addition of two column vectors.

Now to satisfy  $\mathbf{N}(t+1) = \mathbf{N}(t) = \mathbf{N}^*$  at time  $t=t^*$  we require to satisfy

$$\mathbf{N}^* \geq \mathbf{W}^{*T} \circ \mathbf{N}^*.$$

Thus the result follows.

**Theorem 18.2:** *The condition for stability, limit cycles and instability of the dynamics (18.16) depends on the choice of  $\alpha$  as prescribed below:*

*Stable: when  $0 < \alpha < 2$ ,*

*Limit cyclic: when  $\alpha=2$ , and*

*Unstable, when  $\alpha>2$ .*

*Proof:* Replacing  $\delta$  by  $(E - 1)$  in expression (18.16) we obtain:

$$(E - 1 + \alpha) w_{ij} = X \quad (18.21)$$

where

$$X = [1 / \{1 + \exp(-n_i)\}] [1 / \{1 + \exp(-n_j)\}]. \quad (18.22)$$

The auxiliary equation for expression (18.21) is given by

$$(E - 1 + \alpha) w_{ij} = 0 \quad (18.23)$$

$$\text{or, } E = 1 - \alpha. \quad (18.24)$$

Therefore the complementary function is given by

$$w_{ij}(t) = A (1 - \alpha)^t \quad (18.25)$$

where  $A$  is a constant to be determined from boundary conditions.

Let steady-state value of  $w_{ij}$  and  $X$  be denoted by  $w_{ij}^*$  and  $X^*$  respectively. Now, for computing  $w_{ij}^*$  we first need to compute  $X^*$ . The steady-state value of  $w_{ij}$  can now be obtained from the particular integral

$$w_{ij}(t) = X^* / (E - 1 + \alpha). \quad (18.26)$$

Since  $X^*$  is constant, therefore, we substitute  $E = 1$  in expression (18.26) to determine  $w_{ij}^*$ . Thus,

$$w_{ij}^* = X^*/\alpha. \quad (18.27)$$

The complete solution for (18.16) is given by

$$w_{ij}(t) = A (1 - \alpha)^t + X^*/\alpha. \quad (18.28)$$

Substituting  $t=0$  in (18.28) we obtain:

$$A = w_{ij}(0) - X^*/\alpha. \quad (18.29)$$

Combining (18.28) and (18.29) we obtain the complete solution for  $w_{ij}(t)$  given below:

$$w_{ij}(t) = [w_{ij}(0) - X^*/\alpha] (1 - \alpha)^t + X^*/\alpha. \quad (18.30)$$

The condition for stability, limit cycles and instability as stated in the theorem follows directly from (18.30).

**Theorem 18.3:** *The steady-state value of  $X$  always satisfies the following inequality:*

$$0.25 \leq X^* \leq 0.48. \quad (18.31)$$

*Proof:* From (18.22) we have

$$X^* = [1/\{1 + \exp(-n_i^*)\}] [1/\{1 + \exp(-n_j^*)\}], \quad (18.32)$$

where  $n_i^*$  and  $n_j^*$  denote the steady-state values of  $n_i$  and  $n_j$  respectively.

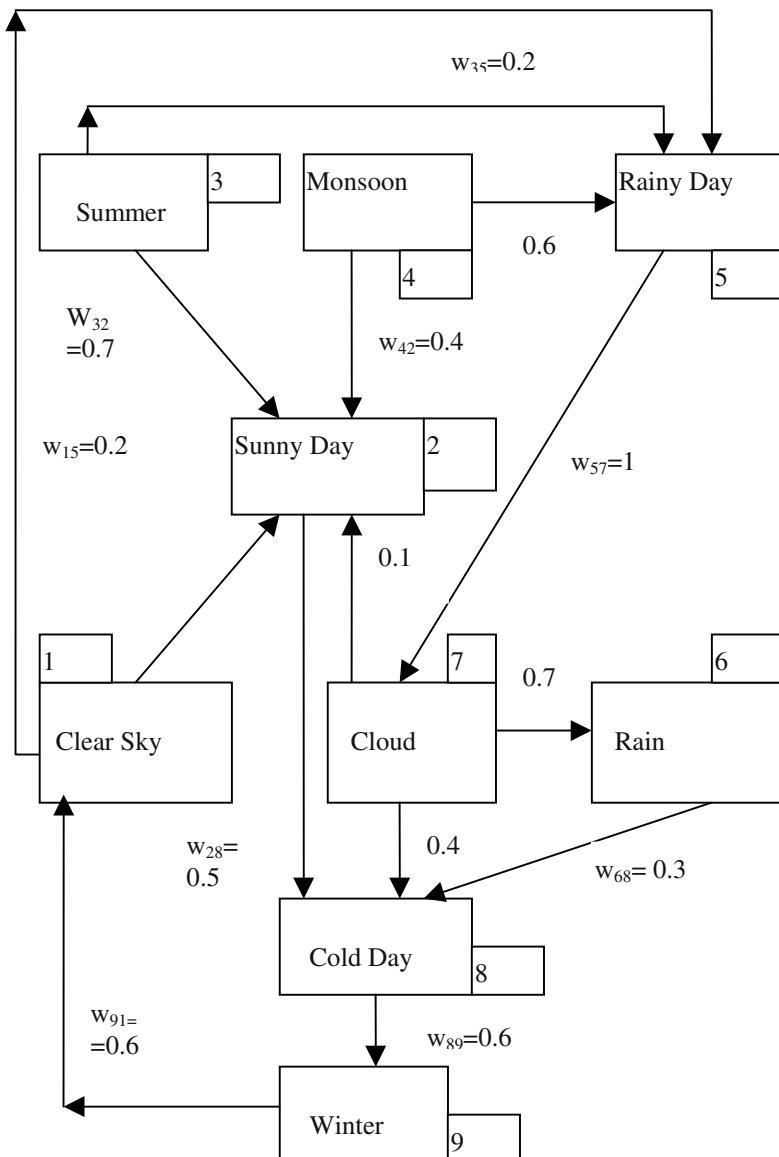
Since right hand side of (18.32) is monotonic for  $0 \leq n_i^*, n_j^* \leq 1$ , we determine the range of  $X^*$  by substituting  $n_i^* = n_j^* = 0$  and  $n_i^* = n_j^* = 1$  respectively in (18.32). Hence, the result follows.

**Corollary 18.2:** *The steady-state value of  $w_{ij}$  always satisfies the following inequality:  $0.25/\alpha \leq w_{ij}^* \leq 0.48/\alpha$ .*

*Proof:* Proof of the corollary directly follows from (18.27) and (18.31).

**Corollary 18.3:**  $w_{ij}^* > 0$  for all weights  $w_{ij}(0) \neq 0$  follows directly from Theorem 18.3.

The significance of the above results is that the cognitive learning system never destroys a weight of a link permanently. In other words, the cognitive learning model maintains stability of the structure of the cognitive map.

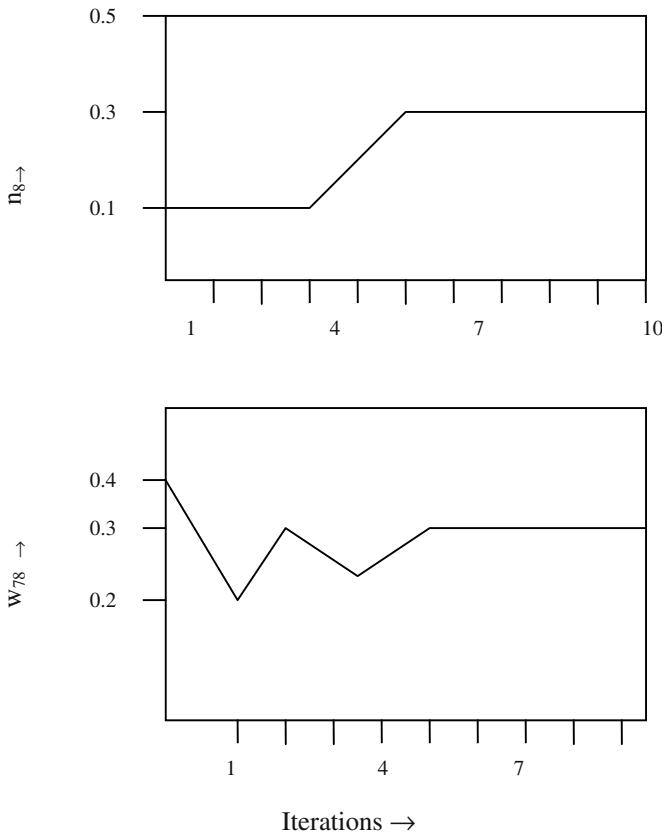


**Fig. 18.3:** Cognitive map of a typical weather forecasting system.

**Computer Simulation:** Pal and Konar [30] illustrated the behavior of their cognitive model with reference to a simulated weather forecasting system. To describe the construction of their cognitive map, let us consider one typical rule embedded in the map using nodes and arcs.

**Rule 1:** An increase in winter causes an increase in clear sky by an amount 0.8 with a CS = 0.6.

The above rule has been encoded by nodes 1 and 9 and the weight  $w_{91}$ . After the construction of the cognitive map is over, initial fuzzy beliefs and CS are mapped at appropriate nodes and arcs of the network.



**Fig. 18.4:** Typical responses of the cognitive map representing the weather forecasting problem.

The CS of the arcs then updated in parallel and with the new CS the fuzzy beliefs at all nodes are updated in parallel. This may be termed as one complete “encode-recall” cycle. A number of cycles are repeated until the steady-state condition is reached. The steady-state beliefs thus obtained represent a permanent non-volatile image of cognitive system’s memory.

The cognitive map presented in Fig. 18.3 has been simulated on a computer and the temporal variations of beliefs and CS of nodes and arcs are obtained for a selected value of  $\alpha$ . Plots of belief for one typical node (say node 9) and one weight (say  $w_{78}$ ) are given in Fig. 18.4 for a value of  $\alpha=1.6$  in the stable range.

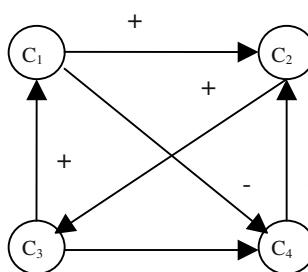
The cognitive model presented by Pal and Konar [ ] thus is found to be stable for  $0 < \alpha < 2$ . Further, the steady-state values of weights are never zero indicating the stability of the structure.

## 18.8 Conclusions

The chapter presented various models of fuzzy cognitive maps and their learning behavior. Most of these models employ unsupervised learning. Kosko’s model and its extension by Pal and Konar include Hebbian type encoding. The model by Zhang et al. is of completely different type and is used for reasoning using a specialized algebra, well known as NPN algebra. All the models introduced in the chapter provide scope for fusion of knowledge of multiple experts. The principle of knowledge fusion has been illustrated in problem 2 of the exercise.

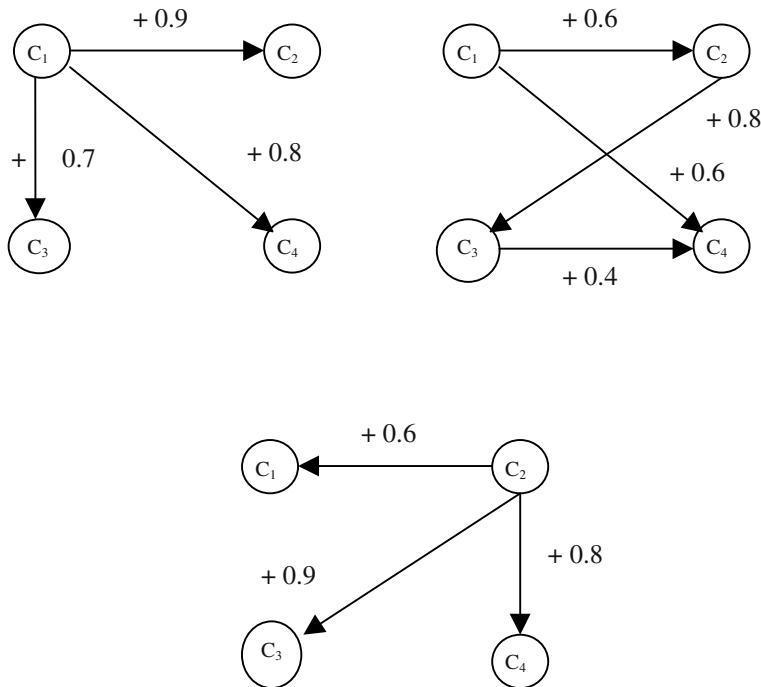
## Exercise

- Given a cognitive map in Fig. 18.5 below with signed edges. Construct an edge matrix  $E$ , whose elements  $e_{ij} \in \{-1, 0, 1\}$ . What is the causal effect of  $C_1$  and  $C_4$  jointly on others?



**Fig. 18.5:** An illustrative cognitive map.

2. Given 4 concepts  $C_1$ ,  $C_2$ ,  $C_3$  and  $C_4$  and three cognitive maps generated by three experts incorporating these concepts. How can you fuse the concepts of the experts?

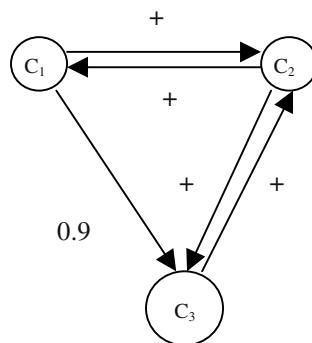


**Fig. 18.6:** Three cognitive maps with shared concepts.

[**Hints:** Convert the cognitive maps (a) through (c) into matrix representation and take their average]

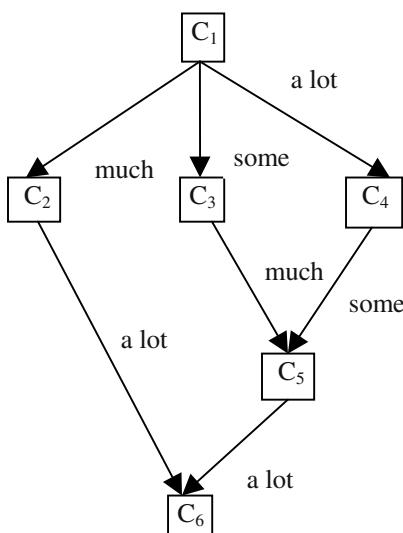
3. Given an FCM [20]:

- Find the edge matrix.
- With a threshold of  $1/2$ , check what state transition take place from the starting state  $[1 \ 1 \ 0]$ .
- Repeat (b) for initial state  $[0 \ 1 \ 0]$ .



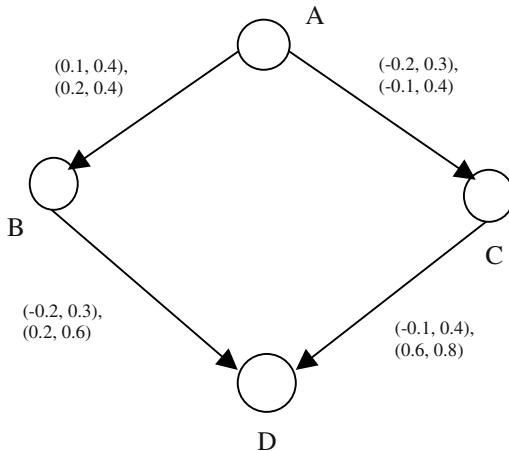
**Fig. 18.7:** An FCM.

4. Given a fuzzy cognitive map with fuzzy labels at the edges. Determine the total effect of  $C_1$  on  $C_6$ . Further given: {none  $\leq$  some  $\leq$  much  $\leq$  a lot}



**Fig. 18.7:** A cognitive map with fuzzy labels at the edges.

5. Given an FCM with 2 sets of fuzzy strengths of the edges. Find the composite fuzzy strength of the edges.



**Fig. 18.8:** An FCM with 2 sets of fuzzy strength of weights.

[**Hints:** Find average of the first labels and the second labels against each arc.]

6. Let  $X = \{x_1, x_2, x_3, x_4, x_5\}$  be a set and  $R$  be a fuzzy relation in  $X \times X$ . Given  $\mu_R(x_1, x_2) = 0.2$ ,  $\mu_R(x_2, x_4) = 0.6$ ,  $\mu_R(x_1, x_5) = 0.8$ ,  $\mu_R(x_5, x_4) = 0.9$ . Find  $\mu_R(x_1, x_4)$ .

[**Hints:**  $\mu_R(x_1, x_4) = \text{Max} [\{\mu_R(x_1, x_2) * \mu_R(x_2, x_4)\}, \{\mu_R(x_1, x_5) * \mu_R(x_5, x_4)\}]$ ]

7. Show that for any fuzzy relation  $R$ , if  $R^0 R o \dots o R$  (k-times) =  $R^k = I$  for integer  $k$ , then  $R^{n+k} = R^n$ , and consequently the fuzzy cognitive map will have limit cyclic behavior.

[**Hints:**  $R^{n+k} = R^n o R^k = R^n o I = R^n$ . Hence, the cognitive map will have sustained oscillation with period =  $k$ ]

8. Verify the result of Pal and Konar's model when applied to Fig. 18.3. Check that  $n_8$  has a steady state belief of 0.3.

## References

- [1] Atkinson, R. C. and Shiffrin, R. M., Human memory: A proposed system and its control process, In *The Psychology of Learning and Motivation: Advances in Research and Theory*, Spence, K. W. and Spence, J. T. (Eds.), vol. 2, Academic Press, NY, 1968.
- [2] Baddeley, A. D., "The fractionation of human memory," *Psychological Medicine*, vol.14, pp. 259-264, 1984.
- [3] Bharick, H. P., "Semantic memory content in permastore: Fifty years of memory for Spanish learned in school," *Journal of Experimental Psychology: General*, vol. 120, pp.20-33, 1984.
- [4] Biswas, B. and Konar, A., *A Neuro-Fuzzy Approach to Criminal Investigation*, Physica-Verlag, 2005 (to appear).
- [5] Biswas, B., Konar, A. and Mukherjee, A. K., "Image Matching with Fuzzy Moment Descriptors," *Engineering Applications of Artificial Intelligence*, Elsevier, vol. 14, pp. 43-49, 2001.
- [6] Chang, T. M., "Semantic memory: Facts and models," *Psychological Bulletin*, vol. 99, pp. 199-220, 1986.
- [7] Downs, R. M. and Davis, S., *Cognitive Maps and Spatial Behavior: Process and Products*, Aldine Publishing Co., 1973.
- [8] Duncan, E. M. and Bourg, T., "An examination of the effects of encoding and decision processes on the rate of mental rotation," *Journal of Mental Imagery*, vol. 7, pp. 33-56, 1983.
- [9] Goldenberg, G., Podreka, I., Steiner, M., Suess, E., Deecke, L. and Willmes, K., Pattern of regional cerebral blood flow related to visual and motor imagery, Results of Emission Computerized Tomography, In *Cognitive and Neuropsychological Approaches to Mental Imagery*, Denis, M, Engelkamp, J. and Richardson, J. T. E. (Eds.), Martinus Nijhoff Publishers, Dordrecht, The Netherlands, pp. 363-373, 1988.
- [10] Greeno, J. G., Process of understanding in problem solving, in *Cognitive Theory*, Castellan, N. J., Pisoni, Jr. D. B. and Potts, G. R. (Eds.), Hillsdale, Erlbaum, NJ, vol. 2, pp. 43-84,1977.
- [11] Gross, C. G. and Zeigler, H. P., *Readings in Physiological Psychology: Learning and Memory*, Harper & Row, NY, 1969.

- [12] Farah, M. J., "Is visual imagery really visual? Overlooked evidence from neuropsychology," *Psychological Review*, vol. 95, pp. 307-317, 1988.
- [13] Jolicoeur, P., "The time to name disoriented natural objects," *Memory and Cognition*, vol. 13, pp. 289-303, 1985.
- [14] Kintsch, W., "The role of knowledge in discourse comprehension: A construction-integration model," *Psychological Review*, vol. 95, pp. 163-182, 1988.
- [15] Kintsch, W. and Buschke, H., "Homophones and synonyms in short term memory," *Journal of Experimental Psychology*, vol. 80, pp. 403-407, 1985.
- [16] Konar, A. and Pal, S., Modeling cognition with fuzzy neural nets, In *Fuzzy Logic Theory: Techniques and Applications*, Leondes, C. T. (Ed.), Academic Press, NY, 1999.
- [17] Konar, A., *Artificial Intelligence and Soft Computing: Behavioral and Cognitive Modeling of the Human Brain*, CRC press, Boca Raton, Florida, 1999.
- [18] Konar, A., and Jain, L. C., *Cognitive Systems Engineering: A Distributed Computational Intelligence Approach*, Springer-Verlag, 2004 (to appear).
- [19] Kosko, B., "Fuzzy Cognitive Maps," *Int. J. of Man-Machine Studies*, vol. 24, pp. 65-75, 1986.
- [20] Kosko, B., *Fuzzy Engineering*, Prentice-Hall, NJ, ch. 15, pp. 499-528, 1997.
- [21] Kosslyn, S. M., Mental imagery, In *Visual Cognition and Action: An invitation to Cognitive Science*, Osherson, D. N. and Hollerback, J. M. (Eds.), vol. 2, pp. 73-97, 1990.
- [22] Kosslyn, S. M., "Aspects of cognitive neuroscience of mental imagery," *Science*, vol. 240, pp. 1621-1626, 1988.
- [23] Marr, D., *Vision*, Freeman, W. H., San Francisco, 1982.
- [24] Matlin, M. W., *Cognition*, Harcourt Brace Pub., Reprinted by Prism Books Pvt. Ltd., Bangalore, India, 1994.

- 
- [25] Milner, B., "Amnesia following operation on the temporal lobe," In *Amnesia Following Operation on the Temporal Lobes*, Whitty, C. W. M. and Zangwill O. L. (Eds.), Butterworth, London, pp. 109-133, 1966.
  - [26] McNamara, T. P., Ratcliff, R. and McKoon, G., "The mental representation of knowledge acquired from maps," *Journal of Experimental Psychology: Learning, Memory and Cognition*, pp. 723-732, 1984.
  - [27] Moar, I. and Bower, G. H., "Inconsistency in spatial knowledge," *Memory and Cognition*, vol. 11, pp. 107-113, 1983.
  - [28] Moyer, R. S. and Dumais, S. T., Mental comparison, In *The psychology of Learning and Motivation*, Bower, G. H. (Ed.), vol. 12, pp. 117-156, 1978.
  - [29] Paivio, A., On exploring visual knowledge, In *Visual Learning, Thinking and Communication*, Randhawa, B. S. and Coffman, W. E. (Eds.), Academic Press, NY, pp. 113-132, 1978.
  - [30] Pal, S. and Konar, A., "Cognitive reasoning with fuzzy neural nets," *IEEE Trans. on Systems, Man and Cybernetics*, Part - B, August, 1996.
  - [31] Peterson, M. A., Kihlstrom, J. F., Rose, P. M. and Glisky, M. L., "Mental images can be ambiguous: Reconstruals and reference-frame reversals," *Memory and Cognition*, vol. 20, pp. 107-123, 1992.
  - [32] Pylyshyn, Z. W., Imagery and Artificial Intelligence, In Minnesota Studies in the Philosophy of Science, *Perception and Cognition Issues in the Foundations of Psychology*, vol. 9, University of Minnesota Press, Minneapolis, pp. 19-56, 1978.
  - [33] Rumelhart, D. E., Mc Clelland, J. L. and the PDP research group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1 and 2, MIT Press, Cambridge, MA, 1986.
  - [34] Shepard, R. N. and Cooper, L.A. (Eds.), *Mental Images and Their Transformations*, MIT Press, Cambridge, MA, 1982.
  - [35] Shepherd, R. N. and Metzler, Z., "Mental rotation of three dimensional objects," *Science*, vol. 171, pp. 701-703, 1971.
  - [36] Tulving, E., "Multiple memory systems and consciousness," *Human Neurobiology*, vol. 6, pp. 67-80, 1987.

- 
- [37] Zhang, W. R., Chen, S. S. and Bezdek, J. C., "Pool2: A Generic System for Cognitive Map Development and Decision Analysis," *IEEE Trans. on Systems, Man and Cybernetics*, vol. 19, no. 1, pp. 31-39, 1989.
  - [38] Zhang, W. R., "Pool- A Semantic Model for Approximate Reasoning and its Application in Decision Support," *Management Information System*, vol. 3, no. 41, pp. 65-78, 1987.
  - [39] Zhang, W. R., "NPN Fuzzy Sets and NPN Qualitative Algebra: A computational Network for Bipolar Cognitive Modeling and Multi-agent Decision Analysis," *IEEE Trans. Systems, Man and Cybernetics*, vol. 26, no. 4, 1996.
  - [40] Zhang, W. R., Chen, S. S., Wang, W. and King, R., "A Cognitive Map Based Approach to the Co-ordination of Distributed Co-operative agents," *IEEE Trans. on Systems, Man and Cybernetics*, vol. 22, no. 1, pp. 103-114, 1992.

# 19

# Machine Learning Using Fuzzy Petri Nets

*The chapter presents a new model for unsupervised learning and reasoning on a special type of cognitive maps, realized with Petri nets. The unsupervised learning process in the present context adapts the weights of the directed arcs from transition to places in the Petri net. A Hebbian type learning algorithm with a natural decay in weights is employed here to study the dynamic behavior of the algorithm. The algorithm is conditionally stable for a suitable range of the mortality rate. A pre-trained network with stable weights may be used in reasoning phase for computing beliefs of the desired propositions from the supplied beliefs of the axioms (places with no input arcs). Because of the conditional stability of the learning algorithm, it may be employed in complex decision-making and learning such as automated car driving in an accident-prone environment. The chapter also presented a new scheme for knowledge refinement by adaptation of weights in a fuzzy Petri net using a different form of Hebbian learning.*

## 19.1 Introduction

Kosko's model [15-16] discussed in chapter 18 suffers from 2 major limitations. Firstly, the model is ineffective to describe many to one (or many) causal

relation. Secondly, the recall model gets trapped within limit cycles and thus is not applicable in real time decision-making problems. In this chapter we propose an extension of Kosko's model that capable of representing many to one (or many) causal relation by using Petri nets and the proposed recall model is free from limit cyclic behavior.

It may be added here that in the proposed model many to one (or many) causal relations are described as a joint occurrence of a set of antecedent clauses and one or more disjunctive consequent clauses. Further, instead of representing positive and negative causal relations separately, we can describe them by a uniform notion without attaching any "sign" label to the arcs in the fuzzy cognitive map. Readers at this stage may wonder: how is it possible? To answer this, suppose there is a negative causal relation:

$$P \rightarrow Q$$

between concepts P and Q, which means P causally decreases Q. We can represent the same causal relation as:

$$P \rightarrow \neg Q$$

which means P causally increases the dis-concept Q. The  $\neg$  symbol is a logical negation operator. Thus we can represent all negative causal relation as equivalent positive causal relations and attach no sign to the arcs to label the positive/ negative causality.

Another point that needs to be addressed before formally introducing the scheme is- why do we select fuzzy Petri nets (FPN) [1-14], [17], [20-24] to model the fuzzy cognitive maps? The answer to this is that FPN supports the necessary many to one (or many) causal relations and it has already proved itself successful as an important reasoning [9-14] and learning tool [20]. The principle of reasoning and learning that we shall introduce in the chapter, however, is different from the existing works.

The chapter is classified as follows. The encoding and recall model of the proposed cognitive map is introduced in section 19.2. The state-space representation of the model is given in section 19.3. An analysis of stability of the proposed model is performed in section 19.4. A computer simulation of an automated collision-free car driving system based on the proposed model is presented in section 19.5. Implication of the results of computer simulation is given in section 19.6. A new model of knowledge refinement and its stability analysis is presented in section 19.7. A case study to illustrate the concept of knowledge refinement is also included in the same section. Concluding remarks and comparison of the two models are given in section 19.8.

## 19.2 The Proposed Model for Cognitive Reasoning

The model for cognitive reasoning has two main components: (i) the encoding and (ii) the recall modules. The encoding is needed to store stable fuzzy weights at the arcs, connected between transition and places [17], of a FPN, while the recall is required for the purpose of deriving stable fuzzy inferences. Before presenting the model, we formally define an FPN [18].

**Definition 19.1:** An *FPN* is a 9-tuple, given by,

$$FPN = \langle P, Tr, T, D, I, O, Th, n, W \rangle \quad (19.1)$$

where

$P = (p_1, p_2, \dots, p_n)$  is a finite set of places ;

$Tr = (tr_1, tr_2, \dots, tr_m)$  is a finite set of transitions ;

$T = (t_1, t_2, \dots, t_m)$  is a set of tokens in the interval  $[0,1]$  associated with the transitions  $(tr_1, tr_2, \dots, tr_m)$  respectively;

$D = (d_1, d_2, \dots, d_n)$  is a finite set of propositions, each proposition  $d_k$  corresponds to place  $p_k$ ;

$P \cap Tr \cap D = \emptyset$ ; cardinality of  $(P) = \text{Cardinality of } (D)$ ;

$I: Tr \rightarrow P^\infty$  is the input function, representing a mapping from transitions to bags of (their input) places;

$O: Tr \rightarrow P^\infty$  is the output function, representing a mapping from transitions to bags of (their output) places;

$Th = (th_1, th_2, \dots, th_m)$  represents a set of threshold values in the interval  $[0,1]$  associated with transitions  $(tr_1, tr_2, \dots, tr_m)$  respectively;

$n: P \rightarrow [0,1]$  is an association function hereafter called fuzzy belief, representing a mapping from places to real values between 0 and 1;  $n(p_i) = n_b$ , say; and

$W = \{w_{ij}\}$  is the set of weights from  $j$ -th transition to  $i$ -th place, where  $i$  and  $j$  are integers.

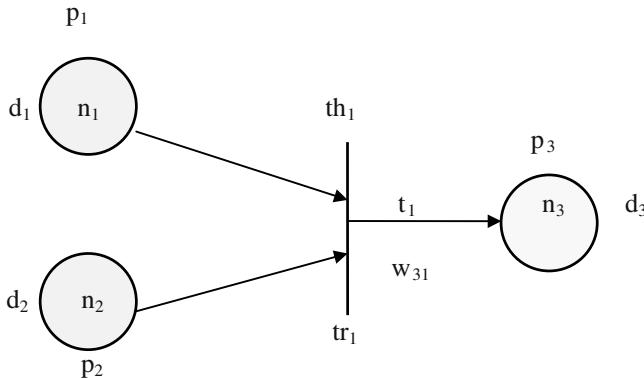
We now present an example to illustrate the above parameters in a typical FPN structure.

**Example 19.1:** The FPN in Fig. 19.1 represents the following causal rule (CR).

CR: (it-is-hot), (the-sky-is-cloudy)  $\rightarrow$  (it-will-rain).

The above rule means that the events (concepts): it-is-hot and the-sky-is-cloudy causally influence the event: it-will-rain. For all purposive action, this rule is identical with typical If-Then relation.

Here,  $P = \{p_1, p_2, p_3\}$ ,  $Tr = \{tr_1\}$ ,  $T = \{t_1\}$ ,  $D = \{d_1, d_2, d_3\}$  where  $d_1 = \text{it-is-hot}$ ,  $d_2 = \text{the-sky-is-cloudy}$  and  $d_3 = \text{it-will-rain}$ ,  $I(tr_1) = \{p_1, p_2\}$ ,  $O(tr_1) = \{p_3\}$ ,  $Th = \{th_1\}$ ,  $n = \{n_1, n_2, n_3\}$  and  $w = \{w_{31}\}$ .



**Fig. 19.1:** A fuzzy Petri net representing the given causal rule.

### 19.2.1 Encoding of Weights

The encoding model, represented by equation (19.2) and presented below, has been designed based on Hebbian learning concept [25], which states that *the strength of the weight of an arc should increase with the increasing signal levels at the adjacent nodes (here places and transitions)*.

$$\frac{d w_{ij}(t)}{dt} = -\alpha w_{ij}(t) + t_j(t) n_i(t) \quad (19.2)$$

where  $w_{ij}$  denotes the weight of the arc connected between transition  $tr_j$  and place  $p_i$ ,  $\alpha$  represents the forgetfulness (mortality) rate,  $t_j$  represents the token associated with transition  $tr_j$  (also called truth token) and  $n_i$  represents the fuzzy belief associated with proposition  $d_i$ . The  $-\alpha w_{ij}(t)$  term indicates a natural decay of the weight  $w_{ij}$  in the cognitive map. This has relevance with the natural tendency of forgetfulness of human beings.

The discrete version of expression (19.2) takes the form of expression (19.3)

$$w_{ij}(t+1) = (1 - \alpha) w_{ij}(t) + t_j(t) \cdot n_i(t) \quad (19.3)$$

### 19.2.2 The Recall Model

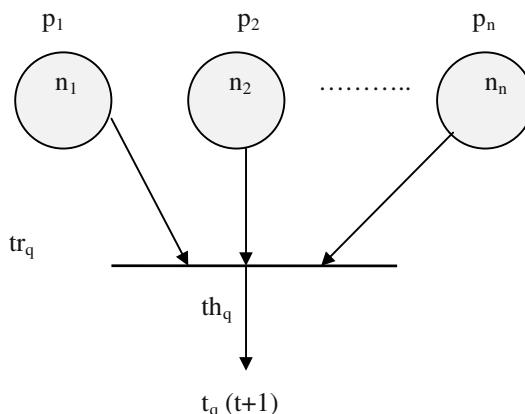
The recall process aims at determining fuzzy beliefs at the places. The recall model in the present context has 2-fold applications. First it is part of the

encoding cycle, and secondly it is an independent recall procedure. The natural question that automatically arises: why does the recall model appear as part of the encoding cycle? The answer to this is really interesting. As the weights are updated once by the recursive equation (19.3), some weights are strengthened, and some are weakened. Spontaneously then some concepts should be strengthened and some should be weakened. A little thinking reveals that similar *metamorphosis* of concepts always takes place in the psychological mind of the human beings. This justifies the inclusion of the recall model in the encoding phase. On the other hand, after the weights are stabilized, the recall/reasoning model is needed in its own course of action to derive new fuzzy inferences from the known beliefs of a set of axioms (places with no input arcs).

Before describing the recall of tokens at places, we first define the firing condition of the transition. *A transition fires if the ANDed value of fuzzy beliefs of its input places exceeds the threshold associated with it.* On firing of a transition, the ANDed (Min) value of belief of its input places is transmitted to its output [18]. This is described below by expression (19.4) (vide Fig.19.2):

$$t_q(t+1) = \left( \bigwedge_{k=1}^n n_k(t) \right) \wedge u\left[ \left( \bigwedge_{k=1}^n n_k(t) \right) - th_q \right] \quad (19.4)$$

where  $n_k(t)$  represents the fuzzy belief of place  $p_k$  which is an input place of transition  $tr_q$  and  $u$  is a unit step function. “ $\wedge$ ” denotes fuzzy AND (min) operator. In the present context, the first  $\wedge$  operator takes minimum of the beliefs  $n_k$  of the corresponding input places  $p_k$  in the FPN. The  $u$  operator checks the firing condition of the transition. If the transition fires, then  $u$  returns a one value, else it is zero. The second  $\wedge$  operator combines the first part of token computation with the third part of firing condition checking. The third  $\wedge$  operator is self-explanatory. The notation  $\exists k$  below the third  $\wedge$  operator represents that the transition  $tr_q$  possesses at most  $n$ -number of input places.

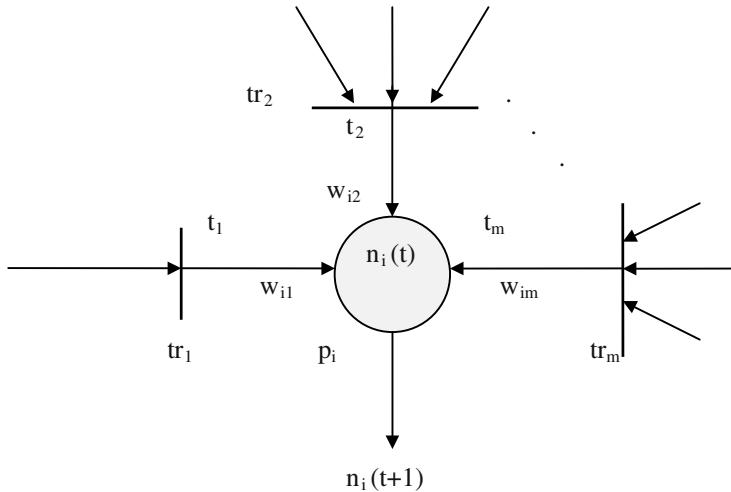


**Fig. 19.2:** An FPN representing connectivity from  $n$  places to transition  $tr_q$ .

Belief updating at place  $p_i$  is explained below by expression (19.5) (vide Fig.19.3):

$$n_i(t+1) = n_i(t) \vee (\bigvee_{j=1}^n (w_{ij}(t) \wedge t_j(t+1))) \quad (19.5)$$

where  $w_{ij}(t)$  denotes the weight of the arc connected between transition  $t_r$  and place  $p_i$ . Here,  $n_i(t+1)$  is computed by taking the maximum ( $\vee$ ) of the current belief  $n_i(t)$  and the influence from the other transitions that can directly affect the belief of place  $p_i$ .



**Fig. 19.3:** An FPN representing connectivity from  $m$  transitions to a place  $p_i$ .

It may be added here that during the recall cycle, fuzzy truth tokens (FTT) at all transitions are updated concurrently, and the results, thus obtained, are used for computing beliefs at all places in parallel by using expression (19.5). This is referred to as the **belief revision/recall cycle**.

### 19.3 State-Space Formulation

The state-space formulation is required for representing the scheme of cognitive reasoning by using simple arrays, instead of graphs for implementation. We will develop three expressions for updating FTT of transitions, fuzzy belief at places and weights in vector-matrix form with the help of two binary connectivity matrices  $P$  and  $Q$ , defined below.

**Definition 19.2:** A *place to transition connectivity (PTC) matrix Q* is a binary matrix whose elements  $q_{jk} = 1$ , if for places  $p_k$  and transition  $tr_b$ ,  $p_k \in I(tr_b)$ ;

otherwise  $q_{jk} = 0$ . If the FPN has  $n$  places and  $m$  transitions, then the  $Q$  matrix is of  $(m \times n)$  dimension [10].

**Definition 19.3:** A **transition to place connectivity (TPC) matrix  $P$**  is a binary matrix whose element  $p_{ij}=1$ , if for places  $p_i$  and transition  $tr_j$ ,  $p_i \in O$  ( $tr_j$ ); otherwise  $p_{ij}=0$ . With  $n$  places and  $m$  transitions in the FPN, the  $P$  matrix is of  $(n \times m)$  dimension [11].

### 19.3.1 State-Space Model for Belief Updating

The equation for belief updating at places is already presented in expression (5). For  $n$  places in the FPN, there will be  $n$  number of expressions analogous to (5). Further, if we have  $m$  number of transitions in the FPN, all of them may not be connected to place  $p_i$  like the configuration shown in Fig. 19.3. So, to represent connections from a few transitions to a place  $p_i$ , we use connectivity matrix  $W$ , and thus  $j$  is replaced by  $\exists j$  (some  $j$ ) in the range 1 to  $m$ . Thus we find the following expression.

$$n_i(t+1) = n_i(t) V \left[ \bigvee_{\exists j=1}^m (t_j(t+1) \wedge w_{ij}) \right] \quad (\text{revised 19.5})$$

For all  $n$  places, the expression presented above together takes the form of (19.6).

$$N(t+1) = N(t) V (W o T(t+1)) \quad (19.6)$$

where  $N$ = fuzzy belief vector of dimension  $(n \times 1)$ , the  $r$ -th component of which represents fuzzy belief of place  $p_r$ ,  $W$ = fuzzy weight matrix from places to their input transition of dimension  $(m \times n)$  and  $T$ = token vector for transitions with dimension  $(m \times 1)$ , such that the  $r$ -th component of  $T$  represents the fuzzy truth token of the  $r$ -th transition. The operator “ $o$ ” represents fuzzy max-min composition operator, which is carried out analogous to conventional matrix multiplication algebra with the replacement of algebraic multiplication by fuzzy AND (Min) and addition by fuzzy OR (Max) operator.

### 19.3.2 State-Space Model for FTT Updating of Transitions

Applying De Morgan’s law, the transition updating equation described by expression (19.4), can be re-modeled as follows:

$$t_q(t+1) = \left( \bigvee_{\exists k=1}^n (n_k) \right) \wedge u \left[ \left( \bigvee_{\exists k=1}^n (n_k) \right) - th_q \right] \quad (\text{revised 19.4})$$

where  $c$  above a parameter represents its one’s complementation. Since all places are not connected to the input of transition  $tr_q$ , therefore, to represent

connectivity from places to transitions we use the binary PTC matrix  $Q$ , with elements  $q_{qk} \in \{0,1\}$ . Thus we find:

$$t_q(t+1) = \underset{\forall k=1}{V} \underset{\forall k=1}{(n_k(t) \wedge q_{ik})} \wedge u[(\underset{\forall k=1}{V} (n_k(t) \wedge q_{ik})) - th_q] \quad (revised \ 19.4)$$

For  $m$  transitions as in Fig. 19.2, we will have  $m$  number of expressions as presented above. Combining all the  $m$  expressions we get the following state-space form for token updating at transitions.

$$T(t+1) = (Q \circ N(t)) \wedge U[(Q \circ N(t)) - Th] \quad (19.7)$$

where

$Th$  = threshold Vector of dimension  $(m \times 1)$ , the  $i$ -th component of which represents the threshold of transition  $tr_i$ ,

$U$ = Unit step vector of dimension  $(m \times 1)$ , and

$N$  = Complement of the belief vector, the  $i$ -th, component of which is the complement of the  $i$ -th component of  $N$ .

The other parameters in expression (19.7) have the same meaning as defined earlier.

### 19.3.3 State-Space Model for Weights

The weight updating equation given by expression (19.3) can be written in vector-matrix form as presented below:

$$W(t+1) = (1 - \alpha) W(t) + ((N(t) \cdot T^T(t)) \wedge P) \quad (19.8)$$

Here,  $W = [w_{ij}]$  is a weight matrix of dimension  $(n \times m)$ ,  $P$  = binary TPC matrix of dimension  $(n \times m)$  with element  $p_{ij} \in \{0,1\}$ , where  $p_{ij}$  represents connectivity from  $tr_j$  to  $p_i$ ,  $T$  over a vector denotes its transposition. The other parameters of expression (19.8) have been defined earlier.

## 19.4 Stability Analysis of the Cognitive Model

The results of stability analysis are presented in theorems 19.1 through 19.3.

**Theorem 19.1:**  $n_i(t)$  in expression (19.5) always converges for  $n_k(0)$  lying between 0 and 1, for all  $k$ .

*Proof:* Since  $n_k(0)$  for all  $k$  are bounded in  $[0, 1]$ , and expressions (19.4) and (19.5) involve AND (Min) and OR (Max) operators,  $n_i(t)$  at any time  $t$  remains

bounded in [0, 1]. Further, from expression (19.5) it is evident that  $n_i(t+1) \geq n_i(t)$ . Thus  $n_i(t+1)$  can never decrease with respect to  $n_i(t)$ , and thus  $n_i(t+1)$  cannot be oscillatory. Consequently,  $n_i(t+1)$  being bounded in [0, 1] and oscillation-free converges to stable point after some finite number of iterations. Hence, the theorem holds good.

**Theorem 19.2:** *The fuzzy truth tokens  $t_i(t)$  at transition  $tr_j$  for all  $j$  given by expression (4) converges to stable points for  $n_k(0)$  lying in [0, 1] for all  $k$ .*

*Proof:* Since fuzzy beliefs at places in the cognitive net converge to stable point, therefore, by expression (4) the proof of the theorem is obvious [9].

**Theorem 19.3:** *The weights in the encoding model given by expression (3) satisfy the following conditions.*

*stable when  $0 < \alpha < 2$ ,*  
*limit cycle when  $\alpha = 2$  and*  
*unstable when  $\alpha > 2$ .*

*Proof:* Rewriting expression (19.3) by extended difference operator E, we find

$$(E - 1 + \alpha) w_{ij}(t) = t_j(t) \cdot n_i(t) \quad (19.9)$$

Now, to prove the theorem we require to solve the above equation. The complementary function (CF) for the above equation is given in the expression

$$\text{CF: } w_{ij}(t) = A (1 - \alpha)^t \quad (19.10)$$

where A is a constant.

The particular integral (PI) for expression (19.9) is given by expression (19.11).

$$\text{PI: } w_{ij}(t) = \frac{t_j(t) \cdot n_i(t)}{E - 1 + \alpha} \quad (19.11)$$

Since PI represents the steady-state value of  $w_{ij}(t)$ , let us consider the steady state values of  $t_j(t)$  and  $n_i(t)$  in expression (19.8). Let  $t_j(t)$  as  $t \rightarrow \infty$  be called  $t_j^*$  and  $n_i(t)$  as  $t \rightarrow \infty$  be called  $n_i^*$ .

Replacing  $t_j(t)$  and  $n_i(t)$  be their steady-state values in expression (19.11), we get the numerator of expression (19.11) to be constant. Therefore, we set  $E=1$  in expression (19.11), which yields the resulting PI given by expression (19.12).

$$\text{PI: } w_{ij}^*(t) = \frac{t_j^* n_i^*}{\alpha} \quad (19.12)$$

The complete solution of equation (19.9) is given by (19.13).

$$w_{ij}(t) = A(1 - \alpha)^t + \frac{t_j^* n_i^*}{\alpha} \quad (19.13)$$

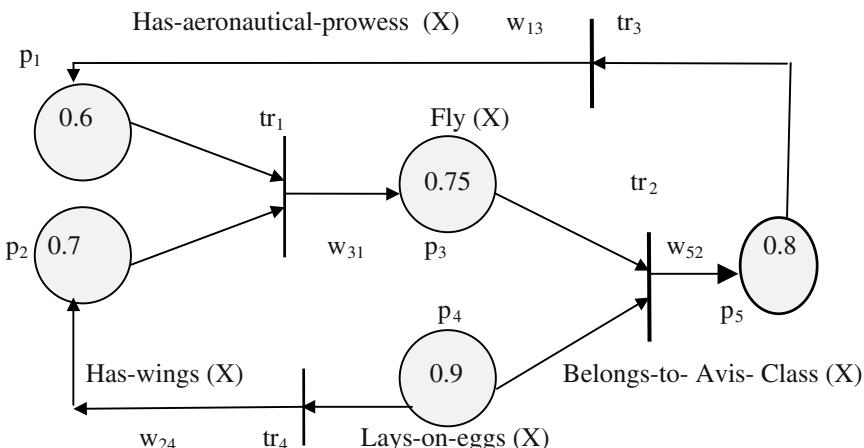
Substituting  $t = 0$  in expression (19.13) we find the value of  $A$ , which on further substitution in (19.13) yields the complete solution given by (19.14).

$$w_{ij}(t) = (w_{ij}(0) - t_j^* n_i^* / \alpha)(1 - \alpha)^t + \frac{t_j^* n_i^*}{\alpha} \quad (19.14)$$

The condition for stability, limit cycle and instability, now, directly follows from expression (19.14).

In the following example, we verify the condition for stability, limit cycles and instability of encoding process and unconditional stability of the recall process.

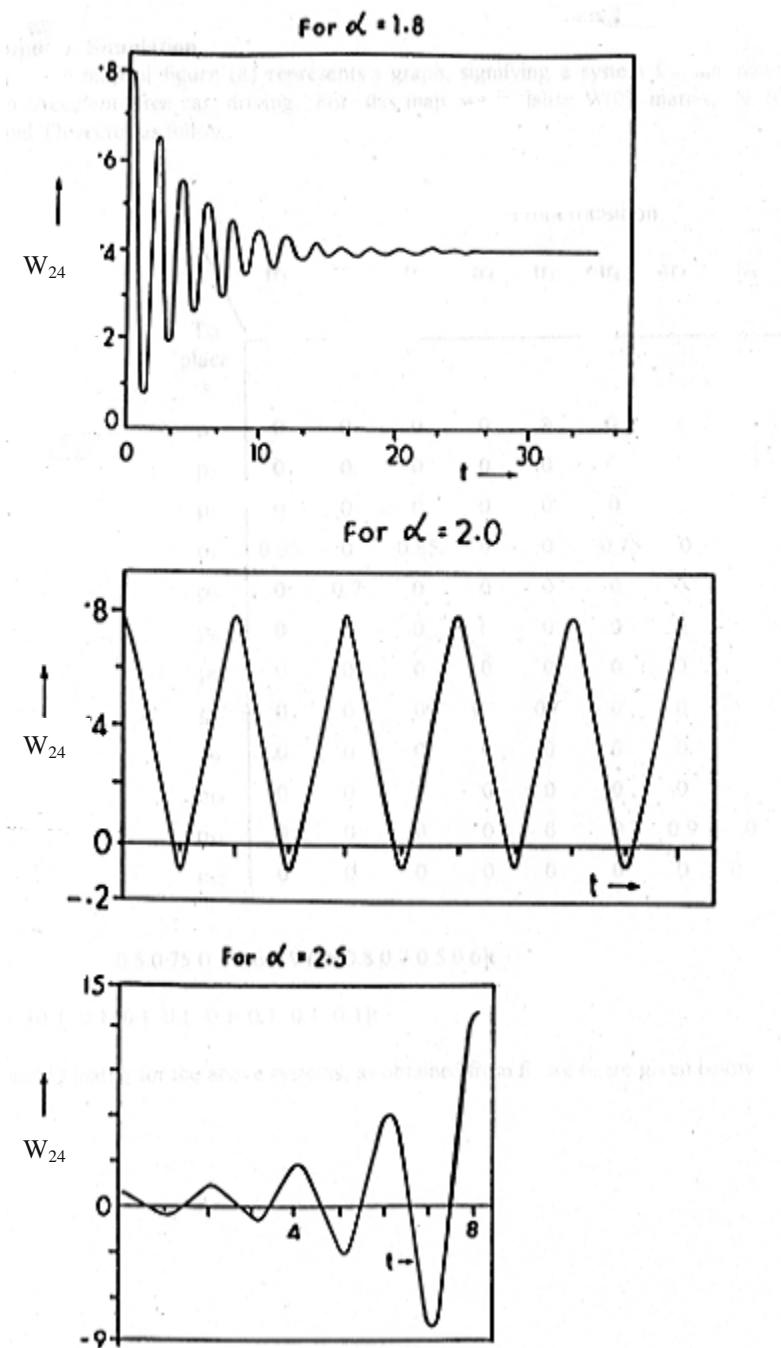
**Example 19.2:** Consider the cognitive map for bird (X) given in Fig. 19.4. The initial value of beliefs, weights and thresholds are shown in the figure itself. The encoding and recall models given in expressions (19.6), (19.7) and (19.8) are repeated until limit cycles, steady-state or instability is observed on the plots, vide Fig. 19.5. The results obtained in these figures are consistent with theorem 3.



Initial weights:  $w_{31}(0) = 0.85$ ,  $w_{52}(0) = 0.7$ ,  $w_{13}(0) = 0.85$ ,  $w_{24}(0) = 0.8$ .

Thresholds:  $th_i = 0$  for all  $i$ .

**Fig. 19.4:** A cognitive map for birds built with FPN.



**Fig.19.5:** Effect of  $\alpha$  on the dynamics of a sample weight  $w_{24}$  in Fig. 19.4.

## 19.5 Computer Simulation

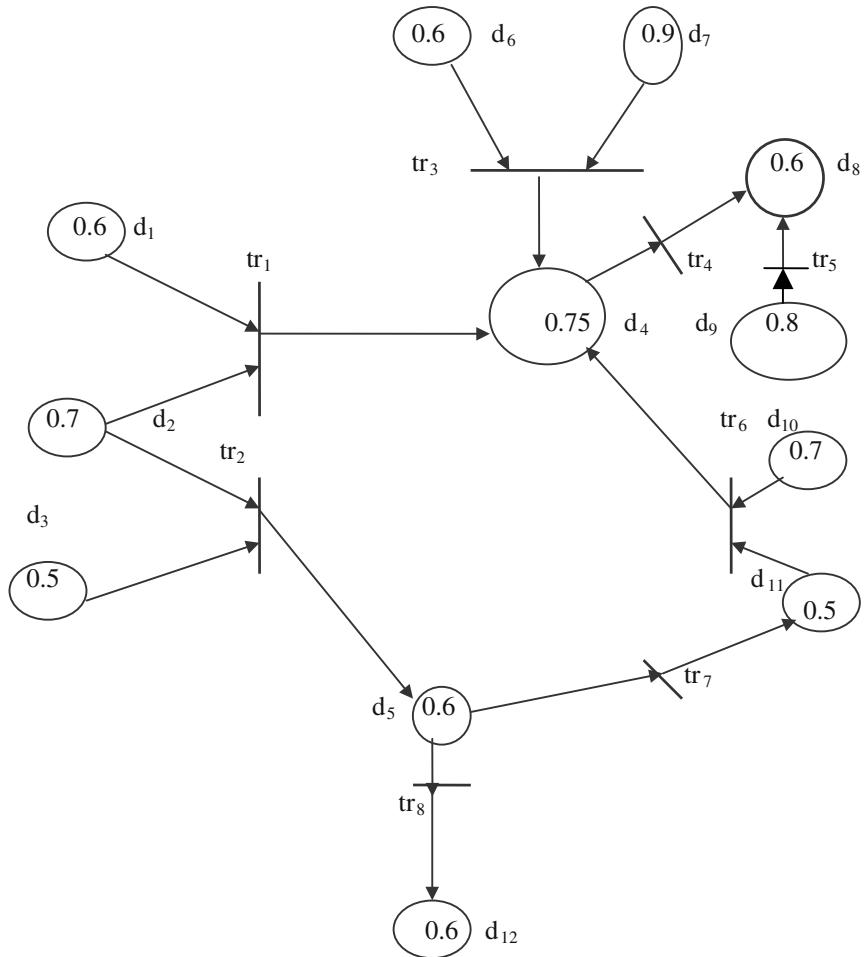
The cognitive map of Fig. 19.6 represents a graph, signifying a system for automated Collision/Accident-free car driving. For this map, we initialize W(0) matrix, N (0) vector and Th vector as follows.

W (0) =		From transition							
To place		tr <sub>1</sub>	tr <sub>2</sub>	tr <sub>3</sub>	tr <sub>4</sub>	tr <sub>5</sub>	tr <sub>6</sub>	tr <sub>7</sub>	tr <sub>8</sub>
p <sub>1</sub>		0	0	0	0	0	0	0	0
p <sub>2</sub>		0	0	0	0	0	0	0	0
p <sub>3</sub>		0	0	0	0	0	0	0	0
p <sub>4</sub>		0.95	0	0.85	0	0	0.75	0	0
p <sub>5</sub>		0	0.7	0	0	0	0	0	0
p <sub>6</sub>		0	0	0	0	0	0	0	0
p <sub>7</sub>		0	0	0	0	0	0	0	0
p <sub>8</sub>		0	0	0	0.8	0.4	0	0	0
p <sub>9</sub>		0	0	0	0	0	0	0	0
p <sub>10</sub>		0	0	0	0	0	0	0	0
p <sub>11</sub>		0	0	0	0	0	0	0.9	0
p <sub>12</sub>		0	0	0	0	0	0	0	0.85

$$N (0) = [0.6 \ 0.7 \ 0.5 \ 0.75 \ 0.6 \ 0.6 \ 0.9 \ 0.6 \ 0.8 \ 0.7 \ 0.5 \ 0.6]^T$$

$$Th = [0.1 \ 0.1 \ 0.1 \ 0.1 \ 0.1 \ 0.1 \ 0.1 \ 0.1 \ 0.1]^T$$

The P and Q matrix for the above systems, as obtained from Fig. 19.6, are given in page no. 534.



$d_1$ =Car behind side-car narrow in breadth,  $d_2$  = Side-car of the front car too close,  $d_3$  = Car behind side-car is wide,  $d_4$  = Front car speed decreases,  $d_5$ = Front car speed increases,  $d_6$  =Passer-by changes his/ her direction,  $d_7$ = Passer-by crosses the road,  $d_8$ = Rear car speed decreases,  $d_9$  =Front car changes direction,  $d_{10}$ = Rear car changes direction,  $d_{11}$ = Rear car speed increases,  $d_{12}$ = Rear car keeps safe distance with respect to front car.

**Fig. 19.6:** Fuzzy cognitive map, representing a car driver's problem.

		From transition							
		tr <sub>1</sub>	tr <sub>2</sub>	tr <sub>3</sub>	tr <sub>4</sub>	tr <sub>5</sub>	tr <sub>6</sub>	tr <sub>7</sub>	tr <sub>8</sub>
P =	To places	p <sub>1</sub>	0	0	0	0	0	0	0
	p <sub>2</sub>	0	0	0	0	0	0	0	0
	p <sub>3</sub>	0	0	0	0	0	0	0	0
	p <sub>4</sub>	1	0	1	0	0	1	0	0
	p <sub>5</sub>	0	1	0	0	0	0	0	0
	p <sub>6</sub>	0	0	0	0	0	0	0	0
	p <sub>7</sub>	0	0	0	0	0	0	0	0
	p <sub>8</sub>	0	0	0	1	1	0	0	0
	p <sub>9</sub>	0	0	0	0	0	0	0	0
	p <sub>10</sub>	0	0	0	0	0	0	0	0
	p <sub>11</sub>	0	0	0	0	0	0	1	0
	p <sub>12</sub>	0	0	0	0	0	0	0	1

		From place											
		p <sub>1</sub>	p <sub>2</sub>	p <sub>3</sub>	p <sub>4</sub>	p <sub>5</sub>	p <sub>6</sub>	p <sub>7</sub>	p <sub>8</sub>	p <sub>9</sub>	p <sub>10</sub>	p <sub>11</sub>	p <sub>12</sub>
Q =	To transitions	tr <sub>1</sub>	1	1	0	0	0	0	0	0	0	0	0
	tr <sub>2</sub>	0	1	1	0	0	0	0	0	0	0	0	0
	tr <sub>3</sub>	0	0	0	0	0	1	1	0	0	0	0	0
	tr <sub>4</sub>	0	0	0	1	0	0	0	0	0	0	0	0
	tr <sub>5</sub>	0	0	0	0	0	0	0	0	1	0	0	0
	tr <sub>6</sub>	0	0	0	0	0	0	0	0	0	1	1	0
	tr <sub>7</sub>	0	0	0	0	1	0	0	0	0	0	0	0
	tr <sub>8</sub>	0	0	0	0	1	0	0	0	0	0	0	0

The fuzzy encoding and recall equations given by expressions (19.8) and (19.7) & (19.6) are updated with  $\alpha = 1.8 (<2)$  in order until a convergence in weights is attained. The steady-state value of weights, thus obtained, is saved by the cognitive system for application during recognition phase. Its value, as obtained through simulation, is found to be  $W^*$ , where

$$w_{41}^* = 0.25, \quad w_{43}^* = 0.25, \quad w_{46}^* = 0.25, \quad w_{56}^* = 0.17, \quad w_{84}^* = 0.31, \\ w_{85}^* = 0.33, \quad w_{11,7}^* = 0.2, \quad w_{12,8}^* = 0.2, \text{ and } w_{ij} = 0 \text{ for all other values of } i, j.$$

Now, with a new  $N(0)$  vector, collected from a source, as given below and the  $W^*$  matrix, cognitive system can derive new steady-state inferences  $N^*$  by using expressions (19.7) and (19.6) only in order. The value of  $N(0)$  and  $N^*$  are presented below:

$$N(0) = [0.2 \ 0.3 \ 0.4 \ 0.0 \ 0.0 \ 0.3 \ 0.35 \ 0.0 \ 0.4 \ 0.3 \ 0.0 \ 0.0]^T$$

$$N^* = [0.2 \ 0.3 \ 0.4 \ 0.25 \ 0.17 \ 0.3 \ 0.35 \ 0.33 \ 0.4 \ 0.3 \ 0.17 \ 0.17]^T$$

Since among the concluding places  $\{p_4, p_5, p_8, p_{11}, p_{12}\}$ ,  $p_8$  has the highest steady-state belief ( $=0.33$ ), therefore  $d_8$  (RC-SD) has to be executed.

## 19.6 Implication of the Results

Stability analysis of a cognitive map, represented by fuzzy Petri net models has been accomplished in detail in this chapter. The result of the stability analysis envisages that the encoding model is conditionally stable ( $0 < \alpha < 2$ ), while the recall model is unconditionally stable. Further, the cognitive map never destroys any weights during the process of encoding. This is evident from the fact that when  $w_{ij}(0) \neq 0$ ,  $w_{ij}^*$  too is nonzero for any  $i, j$ . This is an important issue because it ensures that the cognitive map never destroys its weights (connectivity) completely.

## 19.7 Knowledge Refinement by Hebbian Learning

This section presents a new method for automated estimation of certainty factors of knowledge from the proven and historical databases of a typical reasoning system. Certainty factors, here, have been modeled by weights in a special type of recurrent fuzzy neural Petri net. The beliefs of the propositions, collected from the historical databases, are mapped at places of a fuzzy Petri net and the weights of directed arcs from transitions to places are updated synchronously following the Hebbian Learning principle until an equilibrium condition [15], [9] following which the weights no longer change further is reached. The model for weight adaptation has been chosen for maintaining consistency among the initial beliefs of the propositions and thus the derived steady-state weights represent a more accurate measure of certainty factors than those assigned by a human expert.

### 19.7.1 The Encoding Model

The process of encoding of weights consists of three basic steps, presented below:

**Step-I:** A transition  $tr_i$  is enabled if all its input places possess tokens. An enabled transition is firable. On firing of a transition  $tr_i$ , its FTT  $t_i$  is updated

using expression (19.15)[8], where places  $p_k \in I(\text{tr}_i)$ ,  $n_k$  is the belief of proposition mapped at place  $p_k$ , and  $\text{th}_i$  is the threshold of transition  $\text{tr}_i$ .

$$t_i(t+1) = (\bigwedge_{1 \leq k \leq n} n_k(t)) \wedge u [(\bigwedge_{1 \leq k \leq n} n_k(t)) - \text{th}_i] \quad (19.15)$$

Expression (19.15) reveals that if  $\bigwedge_{1 \leq k \leq n} n_k > \text{th}_i$ ,

$$\begin{aligned} t_i(t+1) &= \bigwedge_{1 \leq k \leq n} n_k(t) \\ &= 0, \text{ otherwise.} \end{aligned}$$

**Step-II:** After the FTTs at all the transitions are updated synchronously, we revise the fuzzy beliefs at all places concurrently. The fuzzy belief  $n_j$  at place  $p_j$  is updated using expression 19.16 (a), where  $p_i \in O(\text{tr}_j)$  and by using 19.16(b) when  $p_i$  is an axiom, having no input arc.

$$n_i(t+1) = \bigvee_{j=1}^m (t_j(t+1) \wedge w_{ij}(t)) \quad (19.16(a))$$

$$= n_i(t) , \text{ when } p_i \text{ is an axiom} \quad (19.16(b))$$

**Step-III:** Once the updating of fuzzy beliefs are over, the weights  $w_{ij}$  of the arc connected between transition  $\text{tr}_j$  and its output place  $p_i$  are updated following Hebbian learning [10] by expression (19.17).

$$w_{ij}(t+1) = t_j(t+1) \wedge n_i(t+1) \quad (19.17)$$

The above three step cycle for encoding is repeated until the weights become time-invariant. Such a time-invariant state is called equilibrium. The steady-state values of weights are saved for subsequent reasoning in analogous problems.

**Theorem 19.4:** *The encoding process of weights in a cognitive map realized with FPN is unconditionally stable.*

*Proof:* Since  $n_k(0)$  for all  $k$  is bounded in  $[0, 1]$ , and the encoding model (expressions (19.15-19.17)) involves only AND (min) and OR (max) operators,  $n_k(t)$  will remain bounded at any time  $t$ . Further, from expression 19.16(a) we have

$$n_i(t+1) = \bigvee_{j=1}^m (t_j(t+1) \wedge w_{ij}(t)) \quad (19.16(a) \text{ re-written})$$

$$= \bigvee_{j=1}^m \{t_j(t+1) \wedge t_j(t) \wedge n_i(t)\} \quad \text{by (19.17)}$$

$$\begin{aligned} m \\ = \vee_{j=1}^m [\{t_j(t+1) \wedge t_j(t)\}] \wedge n_i(t) \end{aligned}$$

Thus  $n_i(t+1) \leq n_i(t)$ . Consequently,  $n_i(t+1)$  can only decrease with respect to its last value, and thus it cannot have an oscillation. Therefore,  $n_i(t+1)$  being bounded and oscillation-free is stable. This holds for any  $i$ ,  $1 \leq i \leq n$ , where  $n$  is the number of places in the FPN.

If  $n_k(t+1)$  for all  $k$  is stable, then by (19.15)  $t_j(t+1)$  too is stable. Finally,  $n_i(t+1)$  and  $t_j(t+1)$  being stable,  $w_{ij}(y+1)$  by expression (19.17) is also stable. Hence, the encoding process of weights in a cognitive map by the proposed model is stable.

### 19.7.2 The Recall/ Reasoning Model

The reasoning model of a recurrent FPN has been reported elsewhere [8-9]. During the reasoning phase, we can use any of these models including the new model proposed below.

The reasoning / recall model in an FPN can be carried out in the same way as in the first two steps of the encoding model with the following exceptions.

- While initiating the reasoning process, the known fuzzy beliefs for the propositions of a problem are to be assigned to the appropriate places. It is to be noted that in the encoding model the fuzzy beliefs of propositions were submitted using proven case histories.
- The reasoning model should terminate when the fuzzy beliefs associated with all propositions reach steady-state values, i.e., when for all places,

$n_i^*(t+1) = n_i^*(t)$  at  $t = m \in \mathbb{N}$ . The steady-state beliefs thus obtained are used to interpret the results of typical analogous reasoning problems. The execution of the reasoning model is referred to as belief revision cycle [8].

**Theorem 19.5:** *The recall process in an FPN unconditionally converges to stable points in belief space.*

*Proof:* Directly follows from the first part of the proof of Theorem 19.4 showing that  $n_i(t+1)$  for all  $i$  is stable.

### 19.7.3 Case Study by Computer Simulation

In this study, we consider two proven case histories described by (Rule base I, Database I) and (Rule base II, Database II). The beliefs of each proposition in the FPNs (vide Fig. 19.7 and 19.8) for these two case histories are known. The encoding model for the cognitive map presented above has been used to estimate

the CFs of the rules in either cases. In case the estimated CF of a rule, obtained from two case histories differs, we take the average of the estimated values as its CF.

### Case History I

#### Rule base I :

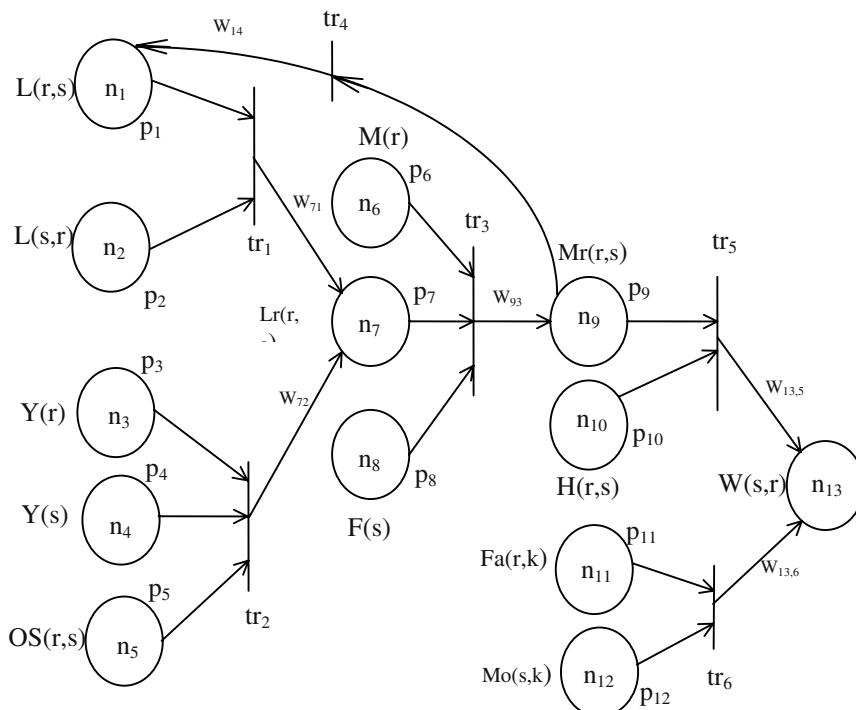
- PR1 : Loves(x,y), Loves(y,x) → Lover(x,y)
- PR2 : Young(x), Young(y),  
Opposite-Sex (x,y) → Lover(x,y)
- PR3 : Lover(x,y), Male(x), Female(y) → Marries(x,y)
- PR4 : Marries(x,y) → Loves(x,y)
- PR5 : Marries(x,y), Husband(x,y) → Wife(y,x)
- PR6 : Father(x,z), Mother(y,z) → Wife(y,x)

#### Database I :

Loves (ram,sita), Loves(sita,ram), Young(ram),  
Young(sita), Opposite-Sex(ram,sita),  
Male(ram), Female(sita), Husband(ram,sita),  
Father(ram, kush),  
Mother (sita, kush)

**Table 19.1:** Parameters of case history I

Initial Weights $w_{ij}$	$w_{71}=0.8, w_{72}=0.7, w_{93}=0.6,$ $w_{14}=0.9, w_{13,5}=0.8, w_{13,6}=0.5$
Initial Fuzzy Beliefs $n_i$	$n_1=0.2, n_2=0.8, n_3=0.75,$ $n_4=0.9,$ $n_5=0.6, n_6=0.75, n_7=0.35,$ $n_8=0.85, n_9=0.45, n_{10}=0.85,$ $n_{11}=0.7, n_{12}=0.65, n_{13}=0.$
Steady-state weights after 4 iterations	$w_{71}=0.35, w_{72}=0.60, w_{93}=0.35,$ $w_{14}=0.35, w_{13,5}=0.35,$ $w_{13,6}=0.50$
$th_j = 0$ for all transitions $tr_j$	



$L = \text{Loves}$ ,  $Y = \text{Young}$ ,  $OS = \text{Opposite-Sex}$ ,  $Lr = \text{Lover}$ ,  $M = \text{Male}$ ,  $F = \text{Female}$ ,

$Mr = \text{Married}$ ,  $H = \text{Husband}$ ,  $W = \text{Wife}$ ,  $Fa = \text{Father}$ ,  $Mo = \text{Mother}$

$r = \text{Ram}$ ,  $s = \text{Sita}$ ,  $k = \text{Kush}$

**Fig. 19.7:** A FPN with initially assigned known beliefs and random weights.

While reasoning in analogous problems, the rules may be assigned with the estimated CFs, as obtained from the known histories. In this example, case-III is a new test problem, whose knowledge base is the subset of the union of the knowledge bases for case-I and case-II. Thus the CFs of the rules are known. In case-III, the initial beliefs of the axioms only are presumed to be known. The aim is to estimate the steady-state belief of all propositions in the network. Since stability of the reasoning model is guaranteed, the belief revision process is continued until steady-state is reached. In fact steady state occurs in the reasoning model of case-III after 5 belief revision cycles. Once the steady-state condition is reached, the network may be used for generating new inferences.

The FPN, given in Fig. 19.7, has been formed using the above rule-base and database from a typical case history. The fuzzy beliefs of the places in Fig. 19.7

are found from proven historical database. The initial weights in the network are assigned arbitrarily and the model for encoding of weights is used for computing the steady-state value of weights (vide Table 19.1).

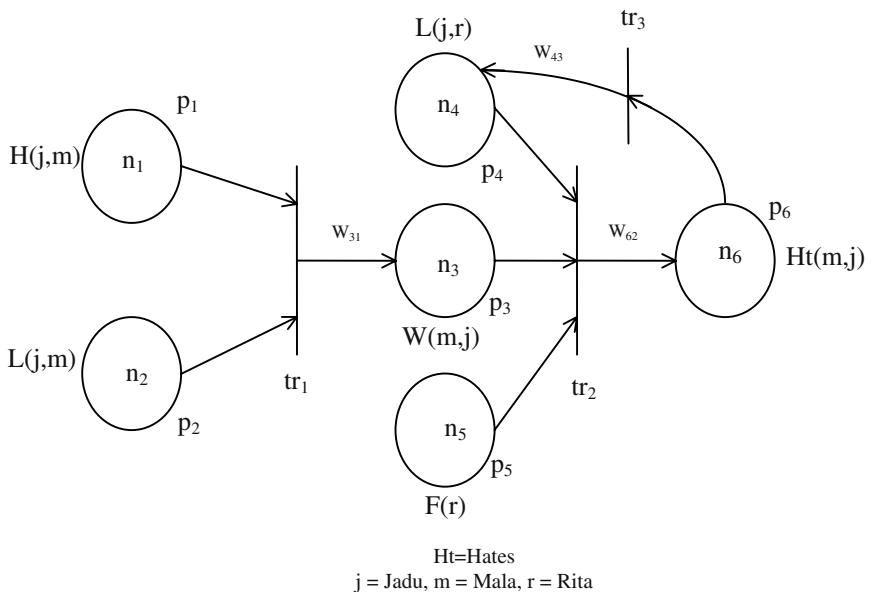
### Case History II

#### Rule base II :

- PR1 : Wife(y,x), Loves(x,z), Female(z)  $\rightarrow$  Hates(y,x)
- PR2 : Husband(x,y), Loves(x,y)  $\rightarrow$  Wife(y,x)
- PR3 : Hates(z,x)  $\rightarrow$  Loves(x,y)

#### Database II :

Husband(jadu,mala),  
Loves(jadu,mala),  
Loves(jadu,rita), Female(rita)



**Fig. 19.8:** A second FPN with known initial beliefs and random weights.

The FPN of Fig. 19.8 is formed with the rule base and database given above. The system parameters of the FPN of Fig. 19.8 are presented in Table- 19.2.

### The Current Reasoning Problem

Now, to solve a typical reasoning problem, whose knowledge and databases are presented herewith, we need to assign the derived weights from the last two case

histories. The reasoning model can be used in this example to compute the steady state belief of the proposition: Hates (lata, ashoke), with the given initial beliefs of all the propositions. Table 19.3 shows the system parameters of the FPN given in Fig. 19.9.

**Table 19.2:** Parameters of Case History 2

Initial Weights $w_{ij}$	$w_{31}=0.75, w_{62}=0.95, w_{43}=0.8$
Initial Fuzzy Beliefs $n_i$	$n_1=0.8, n_2=0.7, n_3=0.1, n_4=0.2, n_5=0.9, n_6=0.3$
Steady-state weights after 3 iterations	$w_{31}=0.7, w_{62}=0.10, w_{43}=0.10$
$th_j = 0$ for all transitions $tr_j$	

### Current Rule base:

- PR1 : Loves(x,y), Loves(y,x) → Lover(x,y)
- PR2 : Young(x), Young(y), OS(x,y) → Lover(x,y)
- PR3 : Lover(x,y), Male(x), Female(y) → Marries(x,y)
- PR4 : Marries(x,y) → Loves(x,y)
- PR5 : Marries(x,y), Husband(x,y) → Wife(y,x)
- PR6 : Father(x,z), Mother(y,z) → Wife(y,x)
- PR7 : Wife(y,x), Loves(x,z), Female(z) → Hates(y,x)
- PR8 : Hates(z,x) → Loves(x,y)

### Current Database:

Loves(ashoke,lata), Loves(lata,ashoke),  
 Young(asoke), Young(lata), Opposite-Sex(asoke, lata), Male(ashoke),  
 Female(lata), Husband(ashoke,lata),  
 Father(asoke, kamal), Mother(lata, kamal), Loves(ashoke,tina),  
 Female(tina)

### 19.7.4 Implication of the Results

The analysis of stability envisages that both the encoding and the recall model of the fuzzy cognitive map are unconditionally stable. The time required for convergence of the proposed model is proportional to the number of transitions on the largest path (cascaded set of arcs) [9] in the network. The model could be used for determination of CF of rules in a KB by maintaining consistency among the beliefs of the propositions of known case histories.

**Table 19.3:** Parameters of the current reasoning problem

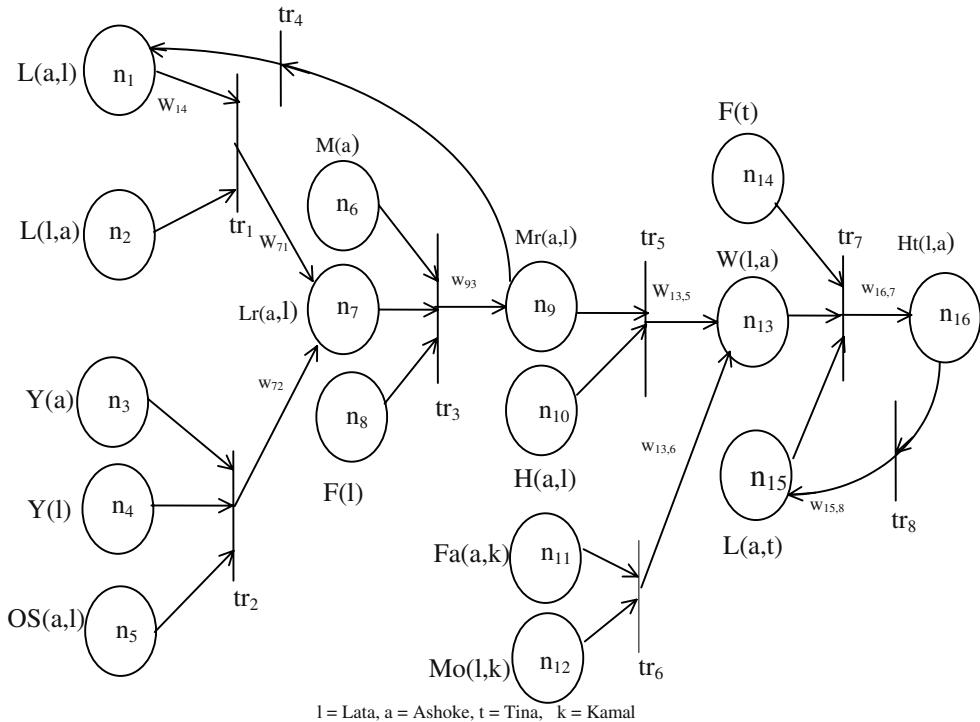
Initial weight $w_{ij}$ taken from the steady state CFs of corresponding rules from earlier case histories I & II shown in parenthesis	$w_{71}=w_{71}(I)=0.35, w_{72}=w_{72}(I)=0.60,$ $w_{93}=w_{93}(I)=0.35, w_{14}=w_{14}(I)=0.35,$ $w_{13,5}=w_{13,5}(I)=0.35, w_{13,6}=w_{13,6}(I)=0.50,$ $w_{16,7}=w_{62}(II)=0.10, w_{15,8}=w_{43}(II)=0.10$
Initial Fuzzy Beliefs $n_i$	$n_1=0.4, n_2=0.8, n_3=0.75, n_4=0.85,$ $n_5=0.65, n_6=0.9, n_7=0.3, n_8=0.7, n_9=0.3,$ $n_{10}=0.95, n_{11}=0.65, n_{12}=0.6, n_{13}=0.25,$ $n_{14}=0.55, n_{15}=0.35, n_{16}=0.40$
<b>Steady-state Belief at place <math>p_{16}</math> for proposition Hates(l,a)</b>	<b><math>n_{16}=0.10</math></b>
$th_j = 0$ for all transitions $tr_j$	

## 19.8 Conclusions

The paper developed two models of fuzzy cognitive maps that are free from the limitations of existing models. The first model employed Hebbian learning with an additional forgetfulness factor  $\alpha$ . The conditional convergence of the first model in encoding and unconditional convergence in recall phases has been proved. The second model has been proved to converge to stable points in both encoding and recall phases.

Both the models can be used for fusion of knowledge from multiple sources. In case, the first model is used for knowledge refinement, then  $\alpha$  for each FPN representing the case histories should be equal. Selection of  $\alpha$  also is an important problem. A large value of  $\alpha$  (<2) causes an overdamped behavior of weights. On the other hand, a very small value of  $\alpha$  (<0.2) requires significant

time for the oscillations of weights to settle down to stable values. Typically,  $\alpha$  should be chosen in the range [1, 1.2] to avoid quick memory refresh but stable learning.



**Fig. 19.9:** An FPN used for estimating the belief of  $Ht(l,a)$  with known initial belief and CFs.

The second model does not include  $\alpha$ , and thus can easily be employed for knowledge refinement from multiple sources. The inferences derived by the second model during the recall phase, however, are more approximate in comparison to those of the first model. In summary, the first model is a more accurate representation of the cognitive learning by human beings, and naturally it invites more difficulty in tuning  $\alpha$ , which does not arise in the second model.

## Exercise

1. Show that at steady state the model given by expression (19.6) satisfies the following inequality:

$$N^* \geq W^* o T^*$$

where the \* above a parameter denotes its steady state value.

**[Hints:** At steady state expression (19.6) can be re-written as

$$N^* = N^* \vee (W^* o T^*).$$

Now, the left hand side  $N^*$  can be equal to the  $N^*$  in the right hand side, if the  $N^*$  in the right hand side satisfies:

$$N^* \geq W^* o T^*.]$$

2. Show that at steady state the weight matrix in the encoding equation (19.8) is given by

$$W^* = (1/\alpha) [(N^* . (T^*)^T) \wedge P].$$

where the \* above a parameter denotes its steady state value.

**[Hints:** At steady state expression (19.8) takes the following form:

$$W^* = (1 - \alpha) W^* + [(N^* . (T^*)^T) \wedge P],$$

which after simplification yields the desired results.]

3. The solution of the following equation describes the steady state minimum value of weight matrix in the encoding cycle represented by the model (19.6-19.8).

$$W^* = (1/\alpha) [((W^* o T^*) . (T^*)^T) \wedge P]$$

**[Hints:** The result follows by combining the results of the last two problems.]

## References

- [1] Bugarin, A. J. and Barro, S., "Fuzzy reasoning supported by Petri nets," *IEEE Trans. on Fuzzy systems*, vol 2, no.2, pp. 135-150,1994.
- [2] Cao, T. and Sanderson, A.C .,"Task sequence planing using Fuzzy Petri nets," *IEEE Trans . on Systems, Man and Cybernetics*, vol.25, no.5, pp.755-769, May 1995.
- [3] Cao, T. and Sanderson, A. C., "A fuzzy Petri net approach to reasoning about uncertainty in robotic systems," *Proc. of IEEE Int. Conf. Robotics and Automation*, Atlanta, GA, pp. 317-322, 1993.
- [4] Cao, T., "Variable reasoning and Analysis about uncertainty with fuzzy Petri nets," *Lecture notes in Computer Scince 691*, Marson, M. A. (Ed.), Springer-Verlag, NY, pp. 126-145, 1993.
- [5] Cardoso, J., Valette, R., and Dubois, D., "Petri nets with uncertain markings," In *Advances in Petri nets*, Lecture notes in computer science, Rozenberg, G. (Ed.), Springer-Verlag, NY, vol.483, pp. 65-78, 1990.
- [6] Daltrini, A. and Gomide, F., "An Extension of fuzzy Petri nets and its applications," *IEEE Trans. on Systems, Man and Cybernetics*, 1993.
- [7] Daltrini, A., *Modeling and Knowledge processing based on the extended fuzzy Petri nets*, M. Sc. degree thesis, UNICAMP-FEE0DCA, May 1993.
- [8] Garg, M. L., Ashon, S. I., and Gupta, P. V., "A fuzzy Petri net for knowledge representation and reasoning", *Information processing letters*, vol.39, pp.165-171, 1991.
- [9] Konar, A., *Uncertainty Management in Expert System using Fuzzy Petri Nets*, Ph.D. dissertation, Jadavpur University, India, 1994.
- [10] Konar, A., *Artificial Intelligence and Soft Computing: Behavioral and Cognitive Modeling of the Human Brain*, CRC press, Boca Raton, 1999.
- [11] Konar A. and Mandal A. K., "Uncertainty management in Expert systems Using Fuzzy Petri nets," *IEEE Trans. on Knowledge and Data Engineering*, vol. 8, no. 1, pp. 96-105, February, 1996.
- [12] Konar A. and Mandal A.K., "Stability analysis of a non-monotonic Petri Nets for diagnostic systems using fuzzy logic," *33rd Midwest Symp. on Circuits, Systems and Computers*, Canada, 1991.

- [13] Konar, A. and Mandal, A.K., "Non-monotonic Reasoning in Expert Systems using Fuzzy Logic," *AMSE Publication*, 1992.
- [14] Konar, S., Konar, A. and Mandal, A. K., "Analysis of fuzzy Petri net models for reasoning with inexact data and knowledge-base," *Proc. of Int. Conf. on Control, Automation, Robotics and Computer Vision*, Singapore, 1994.
- [15] Kosko, B., "Fuzzy Cognitive Maps", *International Journal of Man-Machine Studies*, Vol.24, pp. 65-75, 1986.
- [16] Kosko, B., *Fuzzy Engineering*, Prentice-Hall, ch. 15, pp. 499-527, 1997.
- [17] Lipp, H. P. and Gunther, G., "A Fuzzy Petri net concept for complex decision making process in production control," *Proc. First European congress on fuzzy and intelligent technology (EUFIT '93)*, Aachen, Germany, vol I, pp. 290 – 294, 1993.
- [18] Looney, C. G., "Fuzzy Petri nets for rule-based decision making," *IEEE Trans. on Systems, Man, and Cybernetics*, vol.18, no.1, pp.178-183, 1988.
- [19] Pal, S. and Konar, A., " Cognitive reasoning using fuzzy neural nets," *IEEE Trans. on Systems, Man and Cybernetics* , Part B, vol. 26, no. 4, 1996.
- [20] Pedrycz, W. and Gomide, F., "A Generalized Fuzzy Petri Net model," *IEEE Trans. on Fuzzy systems*, vol. 2, no.4, pp. 295-301, Nov 1994.
- [21] Pedrycz, W., *Fuzzy sets Engineering*, NY, CRC press, Boca Raton, 1995.
- [22] Peters, J. F. and Sohi, N., "Coordination of multi-Agent systems with fuzzy clocks", *Concurrent Engineering Research and Applications*, vol. IV, no. 1, March 1996.
- [23] Scarpelli, H. and Gomide, F., "High level Fuzzy Petri nets and backward reasoning," In *Fuzzy Logic and Soft Computing*, Bouchon-Meunier, B., Yager, R. R. and Zadeh, L. A. (Eds.), World Scientific, 1995.
- [24] Scarpelli, H., Gomide, F. and Yager, R., "A reasoning algorithm for high level Fuzzy Petri nets," *IEEE Trans. on Fuzzy systems* , vol. 4 , no. 3, pp. 282-295, Aug. 1996.
- [25] Simpson, P. K., *Artificial Neural System: Foundation, Paradigm, Application and Implementation*, Pergamon Press, 1990.

# 20

# Computational Intelligence in Tele- Communication Networks

*The chapter discusses the scope of computational models of machine intelligence in tele-communication networks. It begins with a brief introduction to computer networks, and outlines two popular reference models of network architecture. The chapter then presents three interesting problems that the design engineers face in the network layer of the reference models. The problems are centered around network routing, congestion control and call admission control. Classical control and optimization techniques are not suitable for real time solution to these problems. Thus a computational intelligence approach to solve the problems in real time is proposed. In this chapter, routing, congestion control and call admission control have been taken care of by genetic algorithm, fuzzy logic and artificial neural networks respectively.*

## 20.1 Introduction

The last decade observed a massive change in the tele-communication systems. The concept of the “computer center” as a room with a large

computer, where users needed to visit to run their programs, has become obsolete. Nowadays users run their machines, interconnected through a network. Such interconnection help the users to share their program or data. It is needless to mention that connection of computers should not be always done through copper wires; fiber optics, microwave and communication satellites can also be used as part of a communication network.

Since the beginning of the computer network era, two popular reference models of network architecture are prevalent. They are called open systems interconnection (OSI) and transmission control protocol/ internet protocol (TCP/IP) reference models. The communication-scheme that these reference models employ are divided into several layers. Some of the commonly used layers in modern tele-communication systems that take the benefits of both the models are as follows.

- ◆ Application layers
- ◆ Transport layer
- ◆ Network layer
- ◆ Datalink layer
- ◆ Physical layer.

Details of network functions in these layers are available in any standard book on computer networks [11]. In this chapter we shall mainly discuss some common problems of the network layer and their solution by using computational intelligence techniques. The problems to be discussed in this chapter are briefly outlined below.

- ◆ **Routing:** A routing algorithm determines the routes from a source node to a given destination node for efficient communication in a computer network. Routing is essential for both circuit-switching networks and packet-switching networks. In circuit-switching networks such as telephone networks, a circuit is allocated between a source and a destination node before initiating communication, and the data are transmitted through the circuit. In packet-switching networks, on the other hand, data to be sent are decomposed into *packets*, each of which is independently transferred from a given source to a destination node.

Routing algorithms in circuit switching networks generate a route for the circuit based on the status of the network, such as load status of the link and the topological information. In case of packet switching, routing algorithm allocates a route to each packet according to the current status of the network.

- ◆ **Congestion control :** According to international tele-communication union (ITU) congestion in broadband integrated services digital networking (ISDN) is defined as follows:

*“Congestion is defined as a state of network elements (eg. Switches, concentrators, cross-connects and transmission links) in which the network is not able to meet the negotiated network performance objectives for the already established connections and/or for the new connection requests.”*

There exists a similar definition for packet-switching networks. In [10] congestion has been defined as a network state, in which performance of the network degrades due to saturation of network resources. Congestion control is needed to minimize the intensity, duration and spread of congestion. In this chapter we shall present a novel scheme for congestion control using the logic fuzzy sets.

- ◆ **Call admission control (CAC):** The call admission control problem refers to determining whether (or not) to admit a particular call into the network. A call is said to be admitted if the network has sufficient resources to guarantee both the quality of service (QoS) user requests and the QOS already promised to existing calls. The common resources to be allocated include bandwidth of the transmission link and buffer space in switches and routers. In this chapter we present neural and fuzzy models for the CAC problem.

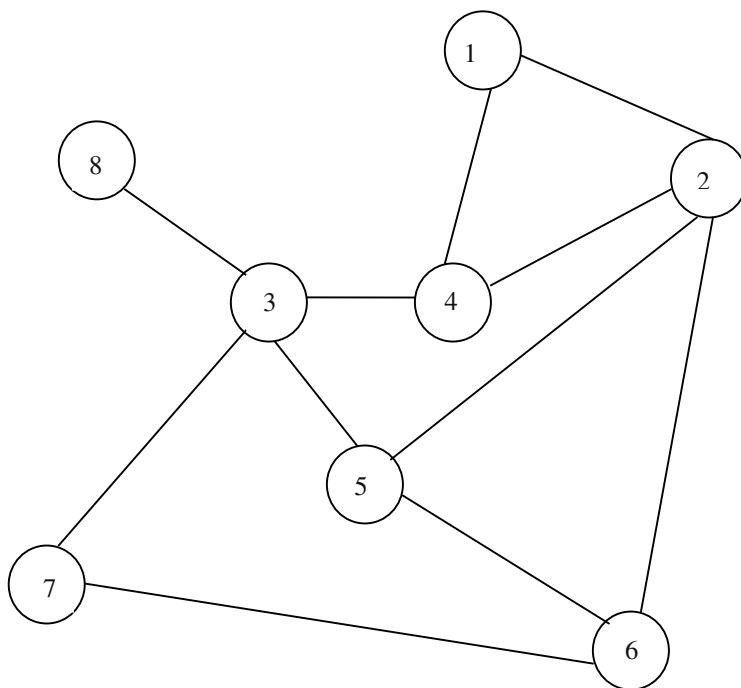
The chapter includes six main sections. In section 20.2 we present a scheme for network routing using genetic algorithm. Section 20.3 provides a new technique for congestion control using fuzzy logic. In section 20.4 the call admission control problem and its solution using neural nets have been addressed. Section 20.5 proposes a scheme for intelligent traffic control using computational intelligence models. Conclusions are listed in section 20.6.

## 20.2 Network Routing Using Genetic Algorithm

In this section, we briefly outline an adaptive network routing algorithm for applications in the internet. The algorithm maintains a limited number of alternative routes to each destination and observes their communication latency adaptively. For generating alternative routes *path-genetic operators* consisting of *path-crossover* and *path-mutation* are designed.

One typical GA-based routing scheme is illustrated in Fig. 20.1. Each node in Fig. 20.1 has its routing table consisting of lists of alternative routes for a given destination. It needs mention here that a routing table does not necessarily

contain route to all destinations in a network. It only contains routes to destinations to which packets are transferred frequently. The table also includes delay along the routes and its weight value. Delay of a route is evaluated by sending observation packets periodically along the route. The weight of a route is computed from its delay. The routes in routing tables are represented by node-IDs along them. For example, a route in Fig.20.1 from node 2 to destination node 1 is 2-4-1 (or 2 4 1) Alternative routes can be generated in routing tables by specialized *path-genetic operators*.



**Fig. 20.1:** Topology of a network.

**Table 20 .1:** Routing table for node 2

Destination node	Route	Delay	Weight
1	(2 1)	100	0.8
	(2 4 1)	180	0.2
5	(2 5)	120	0.4
	(2 6 5)	100	0.5
	(2 4 3 5)	220	0.1
8	(2 4 3 8)	320	1.0
7	(2 6 7)	240	0.5
	(2 5 3 7)	270	0.3
	(2 4 3 7)	300	0.2

### 20.2.1 Path -Genetic Operators

It has been mentioned that typically there exists two varieties of path-genetic operators: path-crossover and path-mutation. Path crossover operator exchanges sub-routes between pairs of routes. An outline of the path-crossover operation is presented below.

Let

$r_1$  and  $r_2$  be a pair of routes between a given source and destination nodes,

$N_c$  be a potential cross-site that includes common set of nodes of  $r_1$  and  $r_2$ ,

and  $n_c$  be a selected node from the given cross-site  $N_c$ .

Then the path-crossover operation between routes  $r_1$  and  $r_2$  is accomplished by the following steps.

- ◆ Identify  $N_c$  from  $r_1$  and  $r_2$  such that  $r_1 \cap r_2 = N_c$  and for any source and destination node  $s$  and  $d$ ,  $s \notin N_c$  and  $d \notin N_c$ .
- ◆ Select a node  $n_c$  from  $N_c$  .
- ◆ Cross-over the routes  $r_1$  and  $r_2$  by exchanging all the nodes after the cross-over point  $n_c$ .

**Example 20.1:** Let the pair of routes be

$$r_1 = (2 \ 4 \ 5 \ 7 \ 8 \ 11 \ 15 \ 17 \ 20),$$

and  $r_2 = (2 \ 3 \ 4 \ 6 \ 9 \ 11 \ 12 \ 14 \ 17 \ 19 \ 20).$

Here,  $N_c = (r_1 \cap r_2)$

$$= (4 \ 11)$$

and  $s \notin N_c$  and  $d \notin N_c.$

Let  $n_c$  be selected randomly as 11. Then, crossover of  $r_1$  and  $r_2$  yields  $r'_1$  and  $r'_2$  where

$$r'_1 = (2 \ 4 \ 5 \ 7 \ 8 \ 11 \ 12 \ 14 \ 17 \ 19 \ 20)$$

$$r'_2 = (2 \ 3 \ 4 \ 6 \ 9 \ 11 \ 15 \ 17 \ 20)$$

It is important to note that on occasions  $N_c$  is a null set, and no crossover is performed when  $N_c$  is null.

Path-mutation operation makes a slight change to a route by perturbation. The following sequence of steps needs to be performed for the path-mutation operation.

- ◆ Select a node  $n_m$  from a given route  $r$ , such that  $n_m$  is not the beginning or the end symbol of  $r$ .
- ◆ Identify a node  $n'_m$  from the neighbors of  $n_m$ .
- ◆ Let  $s$  and  $d$  be the source and destination nodes. Select a path  $p_1$  from  $s$  to  $n'_m$  and  $p_2$  from  $n'_m$  to  $d$ , such that both these paths are shortest.
- ◆ If the resulting path  $p_1$  or  $p_2$  already exist, then no mutation is performed. Otherwise, we obtain a mutated path  $r' = p_1 + p_2$ .

The following example illustrates the phenomenon of mutation.

**Example 20.2:** Let  $r$  be a route given by  $r = (2 \ 4 \ 5 \ 7 \ 8 \ 11 \ 15 \ 17 \ 20)$ . Suppose, we randomly select node 8, and assume that node 9, a neighbor of node 8 is selected. Next we connect node 2 to node 9 and node 9 to node 20 by using Dijkstra's algorithm [4] and generate the offspring:

$$R' = (2 \ 4 \ 6 \ [9] \ 10 \ 19 \ 20)$$

### 20.2.2 Fitness Evaluation

A fitness value of a route is computed from communication latency along routes. To determine the communication delay along a route, a delay query packet is sent to the destination and then sent back. The delay in a route is evaluated by averaging the delays for forward transfer and return of packets.

Let

$d_i$  be the delay along route  $i$ ,  $S$  be a set of route to the same destination.

Then we normalize  $d_i$  by using expression (20.1).

$$\eta_i = \frac{d_i}{\sum_{j \in S} d_j} \quad (20.1)$$

Next we calculate weight  $W_i$  of route by the following expression:

$$W_i = \frac{(1 / \eta_i)}{\sum_{j \in S} (1 / \eta_j)} \quad (20.2)$$

Thus, sum of the weights for routes to a given destination is one.

### 20.2.3 The Genetic Based Routing Algorithm

Each node in the network performs the genetic based algorithm independently according to the following sequence.

- ◆ On creation of a packet at a node, the node determines a route of the packet based on its routing table. Packets received at a node  $n$  are also forwarded to their respective destinations by node  $n$ . The routing table is initially empty. If a route for the destination of a packet is not available, a default route is generated using Dijkstra's shortest-path algorithm.
- ◆ After transfer of a specified number of packets along the route, a delay query packet is sent to determine the communication latency of the route.

On arrival of the packet at the destination, another packet is sent back to notify its answer. Communication latency of the route is then evaluated by averaging the time to send the query packet and its answer.

- ◆ Next the weights of routes to the same destination are evaluated by using expression (20.2). After every evaluation of weights, genetic operators are invoked at a specified probability to create alternative routes in the routing table.
- ◆ If the size of a routing table exceeds its limit, selection operators are employed to reduce its size. Two selection operators such as i) local and ii) global may be used. The local selection operation deletes a route with the smallest weight among routes of a given destination. The global selection operation on the other hand deletes all the routes of a destination for which the sum of frequencies of data transfer is the smallest among all the destinations is a routing table .

The genetic-based routing algorithm attempts to maintain a set of well – performed alternative routes in a routing table, each of which is generated adaptively by genetic operators. The algorithm is capable of determining the entire route of a packet in its source node. This is usually referred to as *source routing*. Conventional routing algorithms in the internet only determine next hop of packets of each node. Thus genetic-based routing algorithm will find extensive applications in the internet routing.

## 20.3 Computational Intelligence in Network Congestion control

Congestion in a network generally takes place because of saturation of network resources such as communication links, buffers, network switches etc. As a specific instance, suppose that a communication link delivers packets to a queue at a higher rate than the service rate of the queue. Then the queue size will gradually increase, and because of a finite length of the queue not only the packets experience delays, but losses also occur during data transmission. Thus it is evident that network resources should be managed properly for sharing among competing users. Congestion takes place for poor management of network resources.

The effect of network congestion is degradation in long delays in delivery of message and losses with buffer overflow. Consequently, there is degradation in the quality of delivered service with the need for retransmission of packets. In the event of retransmission of packets, there is a drop in the throughput, and

when a substantial part of network traffic is due to retransmission, the network throughput collapses inadvertently.

A “good” congestion control system should be preventive, if possible. If not, it should react quickly and minimize the spread of congestion and its duration. In fact elimination of congestion is impossible in practice, as it demands infinite resources. For example, assuring zero waiting time means increasing the service rate or the number of servers to infinity—which is not a feasible solution. Thus some deterioration of performance may be allowed, but some design strategies must be chosen so as to avoid congestion from being intolerable.

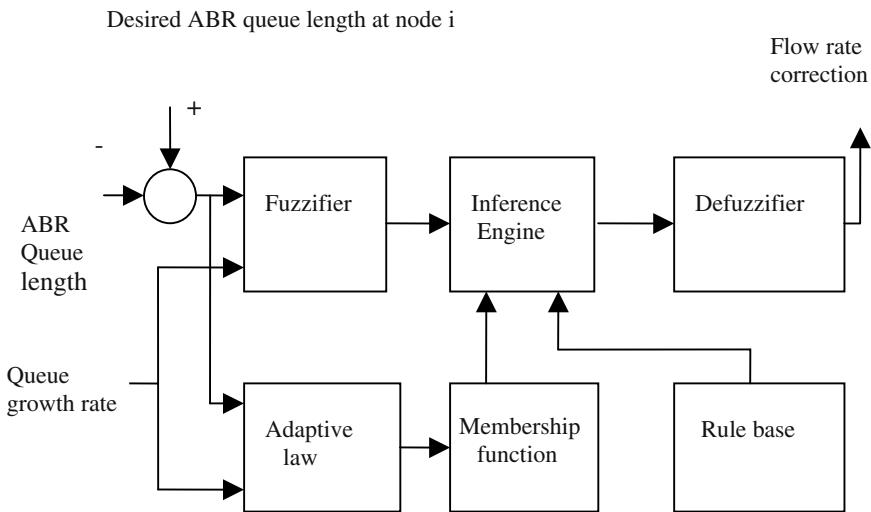
Before presenting the schemes for congestion control, let us have a look at how we can sense or measure congestion. Congestion can be sensed (or predicted) by

1. packet loss sensed by
  - the queue as an overflow
  - sender for lack of acknowledgement (timeout mechanism)
2. packet delay
  - Inferred by the queue size.
  - Acknowledged to the user using time stamps in the packet header.
3. loss of throughput
  - Observed by the sender queue size.

### 20.3.1 Fuzzy Congestion Controller

A fuzzy congestion controller usually accepts queue length, queue length change rate and overall cell-loss probability for all traffic using the same queue as the inputs. The fuzzy congestion controller generates a signed control action as output. A negative value of control action denotes a certain degree of congestion. Under this case a new call has little chance of being accepted. On the other hand, a positive control action indicates that the system is free from congestion to a certain degree. Under such situation new calls have a good chance of entering the network.

A simple scheme of fuzzy congestion controller is shown in Fig. 20.2. The system inputs ABR queue length and queue growth rate, and outputs controlled flow rate.



**Fig. 20.2:** A schematic diagram of a fuzzy congestion controller of node i.

A Mamdani-type controller design is needed to realize the scheme presented in Fig. 20.2. Here, available bit rate (ABR) queue length and queue growth rate are modeled as input fuzzy linguistic variables and ABR-QUEUE-LENGTH and QUEUE-GROWTH-RATE are two fuzzy universe of discourses. TOO-SHORT, ACCEPTABLE and TOO-HIGH are fuzzy subsets of the universe ABR-QUEUE-LENGTH. Similarly, INCREASING-SLOWLY, DECREASING-SLOWLY, DECREASING-FAST, INCREASING-FAST, NOT-CHANGING are fuzzy subsets of the Universe QUEUE-GROWTH-RATE. In the proposed scheme, flow rate is modeled as the output linguistic variable, and thus we define FLOW RATE as a Universe of discourse. INCREASES-SHARPLY, INCREASES-MODERATELY, NOT-CHANGING, DECREASES-SHARPLY are fuzzy subsets of the universe FLOW-RATE. Table 20.2 below provides a list of fuzzy production rules used for the controller design [9].

**Table 20.2:** Fuzzy linguistic rules for congestion control

Input linguistic variable		Output linguistic Variable
ABR queue length	Queue growth rate	Flow rate
1. TOO-SHORT	DECREASING-FAST	INCREASES-SHARPLY
2. -DO-	DECREASING-SLOWLY	INCREASES-MODERATELY
3. -DO-	NOT-CHANGING	INCREASES-MODERATELY
4. -DO-	INCREASING-SLOWLY	DECREASES-MODERATELY
5. -DO-	INCREASING-FAST	DECREASES-MODERATELY
6. ACCEPTABLE	DECREASING-FAST	INCREASES-MODERATELY
7. -DO-	DECREASING-SLOWLY	INCREASES-MODERATELY
8. -DO-	NOT-CHANGING	NOT-CHANGING
9. -DO-	INCREASING-SLOWLY	DECREASES-MODERATELY
10. -DO-	INCREASING-FAST	DECREASES-MODERATELY
11. TOO-HIGH	DECREASING-FAST	NOT-CHANGING
12. -DO-	DECREASING-SLOWLY	NOT-CHANGING
13. -DO-	NOT-CHANGING	DECREASES-MODERATELY
14. -DO-	INCREASES-SLOWLY	DECREASES-SHARPLY
15. -DO-	INCREASES-FAST	DECREASES-SHARPLY

The rules presented in Table 20.2 have two components: antecedent part and consequent part. To illustrate as to how to read the rules from the table, let us consider the first rule, stated below:

*IF (ABR queue length is TOO- SHORT) &  
(queue growth rate is DECREASING-FAST)  
THEN ( flow rate INCREASES -SHARPLY)*

Other rules presented in Table 20.2 may be read in a similar manner. Measurements in the present context being unique, the fuzzy subsets: ABR-QUEUE-LENGTH and QUEUE-GROWTH-RATE are treated as singleton sets.

Computation of the membership distribution of flow rate thus can easily be accomplished by Mamdani-type controller model, as presented below.

Let the j-th rule be defined as

*IF       (ABR queue length is  $A_1^j$ ) and  
(queue growth rate is  $A_2^j$ )  
THEN (flow rate is  $B_1^j$ ).*

where  $A_1^j$ ,  $A_2^j$  and  $B_1^j$  are three fuzzy sets, such that

$$A_1^j \subseteq \text{ABR-QUEUE-LENGTH},$$

$$A_2^j \subseteq \text{QUEUE-GROWTH-RATE},$$

and  $B_1^j \subseteq \text{FLOW-RATE}$ .

Then,

$$\begin{aligned} & \mu_{B_1^j}' \text{ (flow rate)} \\ &= \text{Max} [ \text{Min} \{ (\mu_{A_1^j}' \text{ (ABR queue length)} \cap \mu_{A_2^j}' \text{ (queue growth rate)}) \}] \\ & \qquad \qquad \qquad \mu R^j \text{ (ABR queue length, queue growth rate)} \} ] \end{aligned} \quad (20.3)$$

where

$$A_1'^j \approx A_1^j, A_2'^j \approx A_2^j \text{ and } B_j' \approx B_j.$$

$$\begin{aligned} & \mu R^j \text{ (ABR queue length, queue growth rate)} \\ = & \text{ Min } [\mu A_1^j \text{ (ABR queue length)}, \mu A_2^j \text{ (queue growth rate)}] \quad (20.4) \end{aligned}$$

for ABR queue length  $\in$  ABR-QUEUE-LENGTH.

and queue growth rate  $\in$  QUEUE-GROWTH-RATE.

In the present context we have  $1 \leq j \leq 15$ .

Thus, we compute:

$$\begin{aligned} & \mu_B' \text{ (flow rate)} \\ = & \text{ Max } [\mu_B^{j_j}] \quad (20.5) \\ & 1 \leq j \leq 15 \end{aligned}$$

For computing flow rate, we may use the following center of gravity defuzzification principle.

$$\text{Flow rate} = \frac{\sum_u u \times \mu_B'(u)}{\sum_u \mu_B'(u)} \quad (20.6)$$

where  $u \in$  FLOW-RATE.

One important issues that we did not discuss so far is the scope of tuning the membership curves in Fig 20.2. Generally, Gaussian or triangle-shaped curves are used in typical fuzzy systems. Such curves are described mathematically by its parameter set. For example, a Gaussian curve is described by its mean value  $m$  and variance  $\sigma^2$ . Adaptation of membership function thus means changing mean  $m$  and variance  $\sigma^2$  of the Gaussian curves. An adaptive law is formulated in Fig.20.2 to adjust mean  $m$  and variance  $\sigma^2$  of Gaussian membership curves from the measured value of ABR queue length error and queue growth rate. The tuning can be done with different objectives in mind. One typical objective, for instance, is the maximization of throughput with minimization of end-to-end delay experienced by the users.

There exist neural net-based methods [2], [12] and evolutionary computing techniques [7] for congestion control, and details of them are available in [10].

## 20.4 Computational Intelligence in Handling the Call Admission Control Problem

The call admission control problem, in general, refers to determining whether or not to admit a particular call into the network. A call is admitted if the network includes sufficient resources to guarantee i) the QoS the user demands and ii) the QoS already promised to existing calls. Common resources to be allocated include bandwidth of transmission links and buffer space of switches and routers. A user requesting a network connection submits two sets of parameters, such as i) traffic descriptions and ii) QoS requirements. The traffic descriptions can be any set of parameters like peak rate, average rate, maximum burst size, correlation measure, application type (such as MPEG Video) etc. In many applications, a user does not provide the traffic descriptor, but the network itself measures the traffic behavior and provides the descriptors. Network supplied traffic descriptors include counts of cell or packet arrivals, power spectral density parameters, correlation measures and entropy. The quality of service requirements, on the other hand, includes packet or cell loss rate, maximum or average delay and delay variation.

The general problem of admitting calls into the network consists of two parts. First a candidate path between the caller and the called is to be established; this is referred to as the *routing problem* in the literature [8]. Calls are rejected if no such path can be identified. After identification of the candidate path, each node along the path decides individually if it can accept the call. The call is accepted if each node on the path decides that the resources in the network are sufficient to guarantee the requested QoS of both the existing calls and the new call.

There exist quite a large number of approaches for call admission control. In this section, we present a neural approach to call admission control. But why neural networks are given so much priority for call admission, the following points give an overview to them.

- ◆ Neural nets do not require an accurate mathematical model of either the traffic or the system. Since traffic data may vary widely, neural network models are appropriate for call admission control problem.
- ◆ For a large number of training instance, a neural net will generalize accurately, and is expected to produce accurate outputs for inputs- not in the training set.
- ◆ Neural nets are adaptable as they can be re-trained in real time using the latest measurements.

### 20.4.1 Call Admission Control Using Neural Networks

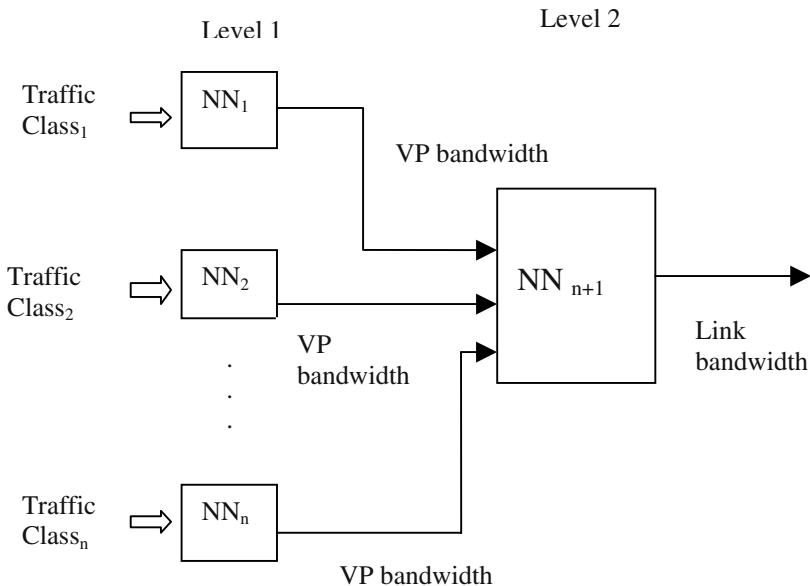
A neural net, suppose has n-inputs:  $u_1, u_2, \dots, u_n$ , and a single output  $y = f(u_1, u_2, \dots, u_n)$ , where  $f(\cdot)$  denotes a input-output mapping. The output  $y$  in the present context can be a QoS estimate, which can be an estimate of the buffer delay, the buffer's loss rate, or the amount of bandwidth needed on the link to carry all the calls whose traffic enters the FIFO buffer. The model we mentioned above having a single output, representing a particular QoS estimate, ensures servicing of only one QoS parameter. To ensure servicing of m number of QoS parameters, we should arrange for m-outputs of the neural networks, one for one QoS parameter.

An alternative version of the above model offers only binary output, representing accept/reject decision of a call. Thus the neural net classifies a call into one of two decision spaces. In the case of a single output representing a QoS estimate, a call is admitted if the QoS estimate for the new candidate aggregate stream is below the most stringent QoS requirement for all calls in the stream. For multiple output call admission control system, the QoS estimate are compared to a threshold of  $\frac{1}{2}$ , where levels 0 and 1 respectively stand for accept and reject status of the call.

Let us now consider a modular design in which multiple neural nets are organized in a hierarchical fashion. Modularity, in general, is needed to solve a complex task by decomposing it into similar sub-tasks and then combining the individual solutions. One typical scheme of call admission control scheme using modular neural nets is presented in Fig 20.3. Here, the neural nets in level 1 determine the amount of multiplexing gain for mixing calls from the same class onto a single VP. The neural net in level 2 on the other hand determines the amount of multiplexing gain for mixing VPs onto a link.

Modular networks have several advantages over a single neutral network. First modular networks usually are trained with few instances and thus the training is relatively faster in comparison to that of single net. Secondly, representation of inputs data developed for a modular net is easier to understand than the cases in ordinary neural net. Third important issue in modular neural net is that it provides a better fit to a discontinuous input-output mapping.

The neural net presented in Fig. 20.3 has  $(n + 1)$  modules. Each module in the proposed neural net is a 3-layered feed forward topology with one input, one output and one hidden layer. First two layers of the neural nets contain a linear activation function with a sigmoid type synaptic non-linearity. The last layer includes a linear activation function only. Generally, back-propagation learning is employed to train the modular neural nets.



**Fig. 20.3:** Sample modular neural net design for call admission control.

#### 20.4.2 Input/ Outputs of Neural Nets Used in Call Admission Control

Researchers have considered various input/output parameters of the neural net used in call admission control. In this section we explore the possibilities of different input/output parameters of the proposed neural model.

##### Possible Inputs

- ◆ **Number of calls per traffic class:** One possible input is no. of calls per traffic class. To be specific, let  $s = (s_k)$ , where  $s_k$ , the k-th component gives the number of calls in the stream that belong to the k-th class, be the input vector of the neural net. The vector  $s$  is called the call vector. The advantage of this approach is that the users need not supply any traffic descriptors at all, and the decision boundary is determined by the input [5], [6].
- ◆ **Counts of arrivals:** Researchers in [12] have considered online traffic measurements as inputs of the neural nets used in handling the call admission control problem. In fact Youssef et al. in [12] determined the number of call arrivals  $N_s(q)$  in each interval  $q$  for each data stream  $s$ . The

neural nets employed in handling the call admission control problem then may be supplied with the number of cell arrivals over consecutive intervals as inputs, and then trained to capture the correlation among consecutive cell arrivals. The main advantage of this input is that the count of arrivals entirely characterizes the input stream, and users need not submit any traffic descriptors.

- ◆ **Variance of Counts:** Variance of counts (VOC) has been considered as the input of neural nets used in call admission control. To calculate VOCs the total time is divided into intervals of equal length. Let  $N_s(q)$  be the number of cells or packets arriving at interval  $q$  for stream S. Let  $N_s(q, h)$  be the number of cells arriving in the interval  $q$  to  $(q + h)$ . The variance of count, for an interval  $m$  is defined by

$$\text{VOCs } (m) = \frac{\text{var} \{ N_s (q+1, q+m) \}}{m}. \quad (20.7)$$

Further, the mean of  $N_s(q)$  denoted by  $\lambda_s$  is given by

$$\lambda_s = \frac{\sum_{i=1}^n N_s(q+i)}{m} \quad (20.8)$$

where  $m = 1, 2, 4, \dots, 2^m$  and  $(M+1)$  is the number of VOCs used.

$\lambda_s$  and VOCs  $(m)$  for  $m = 1, 2, 4, \dots, 2^m$  thus may be supplied to the input of the neural model. The main advantage of VOC as neural net input lies in VOC's second order statistics. Secondly, VOCs are additive. Thirdly, the method is independent of traffic class.

- ◆ **Power spectral density:** The input parameters we discussed so far are all time domain information. Another type of input, capable of capturing correlation and burstiness properties of traffic stream is the power-spectral density (PSD) function in the frequency domain. The PSD is defined as the Fourier transform of the auto correlation function of the input process. In [1], a traffic source is characterized by 3 power-spectrums: the d.c. component ( $u$ ), the half-power bandwidth ( $v$ ), and the average power ( $w$ ). As  $u$  increases, the traffic load increases, as  $v$  decreases, the input power in the low frequency band increases; and as  $w$  increases, the variance of the input rate increases. In [1] a call admission controller is designed where the user can input three simple parameters: its peak rate, mean rate and peak

call rate duration. The controller applies fast Fourier transforms to these inputs and outputs, the PSDs  $u$ ,  $v$  and  $w$  are then submitted to the input of a neural net.

## Possible Outputs

Neural nets used for call admission control possess any one of the outputs.

- ◆ **Accept/Reject Decision:** This output helps the neural net to learn the boundaries between feasible and infeasible performance regions of a given input space.
- ◆ **Loss rate:** With this output the neural net predicts the average buffer overflow rate. Because of a wide range in loss rate, it is usually represented in log scale.
- ◆ **Delay:** Neural networks used in call admission control may predict the average buffer delay or average buffer occupancy as output.
- ◆ **Jitter:** With jitter as output, neural net involved in call admission control predicts the variation in buffer delay.
- ◆ **Bandwidth:** When bandwidth is used as an output parameter, neural nets involved in call admission control predicts the amount of bandwidth needed to achieve a specific QoS for the given input stream.
- ◆ **P-th percentile delay:** A neural net generating p-th percentile delay as output, predicts the value of delay  $D$ , such that the probability that a call or packet experiences a delay less than or equal to is  $p\%$ . Typical value of percentile chosen is around 90%. All the calls in this system need to have the same percentile requirement.

**Example 20.3:** Consider the problem of designing a neural net based call admission control with two services. The first service ensures QoS guarantees to users on bandwidth and delay parameters. In other words, it guarantees minimum bandwidth  $B_1$  and maximum delay  $D_1$ . The second service ensures QoS guarantees to users on bandwidth delay and jitter. In other words, it guarantees minimum bandwidth  $B_2$ , maximum delay  $D_2$  and maximum jitter  $J_2$ . Assuming that there are 6 traffic classes, and the call vector is applied at the neural net input.

- (a) State the number of inputs and outputs of the neural net. Also mention the input/ output parameters.
- (b) Describe the call admission control decision based on these outputs.

**Answer to part (a):** To estimate the delay of service class 1, bandwidth of service class 1, delay of service class 2, bandwidth of service class 2 and jitter of service class 2, we need 5 outputs. Here, number of classes being net should have two components, depicting the number of calls in the stream that belong to the k-th traffic class  $s_k$ , for  $k=1,2$ .

**Answer to part (b):** We symbolize  $D_i$ ,  $B_i$  and  $J_i$  as the delay, bandwidth and jitter of class i. In the present context, the output parameters are  $D_1$ ,  $B_1$ ,  $D_2$ ,  $B_2$  and  $J_2$ . The call admission decision is as follows.

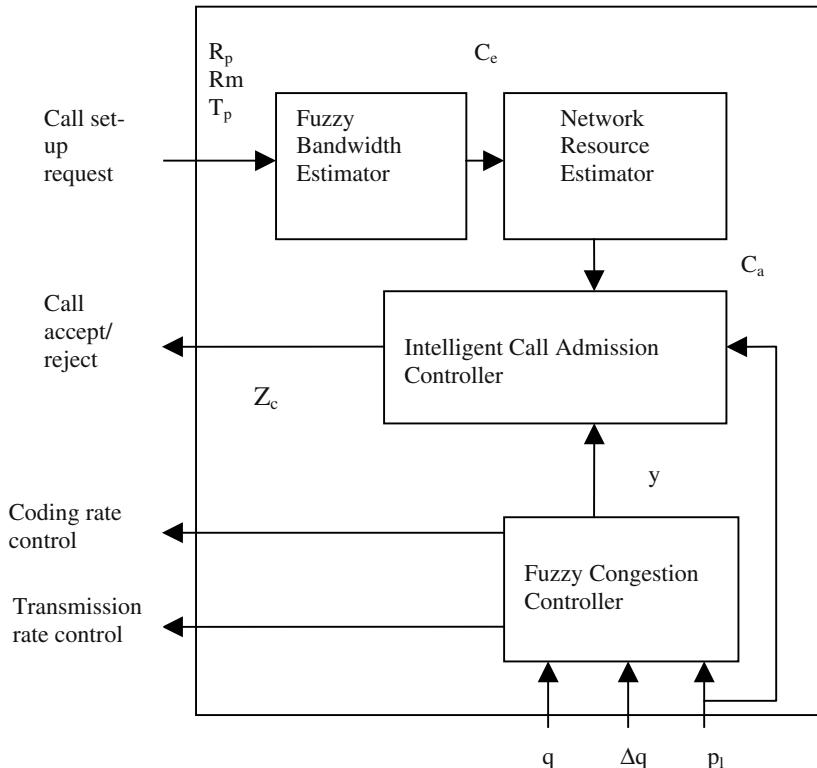
Suppose the requesting stream asks for service type 1. Then the call is admitted if potential new aggregated stream receiving service type1 has estimated delay less than  $D_1$  and estimated bandwidth less than  $B_1$ . Similarly, if the requesting stream demands service type 2, then the call is admitted if the new aggregate traffic receiving service type 2 has estimated delay, bandwidth and jitter less than  $D_2$ ,  $B_2$  and  $J_2$  respectively.

## 20.5 Intelligent Traffic Control

In this section we present a novel scheme for intelligent traffic control following the reference [10]. The scheme includes a fuzzy congestion controller, intelligent call admission controller and fuzzy bandwidth estimator as the basic building blocks (Fig. 20.4).

The fuzzy congestion controller in the present context has three inputs: the queue length  $q$ , the change rate of the queue length  $\Delta q$ , and the cell loss ratio  $p_1$ . Fuzzy congestion controller outputs a parameter  $y$  indicative of congestion in the network. The fuzzy bandwidth estimator determines the required capacity  $C_e$  for a new connection from its traffic description parameter such as peak cell rate  $R_p$ , sustainable cell rate  $R_m$  and peak cell rate duration  $T_p$ .

The network resource estimator performs the accounting of system resource usage. If a new connection with bandwidth  $C_e$  is accepted,  $C_a$  is updated by the rule:  $C_a \leftarrow C_a - C_e$ . On the contrary, if an existing connection with bandwidth  $C_e$  is disconnected,  $C_a$  is updated by the rule:  $C_a \leftarrow C_a + C_e$ . The call admission controller sends a decision signal  $Z_e$  back to the new connection to indicate acceptance/ rejection of the new connection request.



**Fig. 20.4:** A schematic diagram of an intelligent traffic controller.

## 20.6 Conclusions

The chapter addressed three important problems in computer networks namely congestion control, call admission control and routing. Classical control theory and optimization techniques are not suitable for handling these problems in real time. Consequently, the chapter handles the said problems by computational models of machine intelligence.

Genetic algorithms with new type of crossover and mutation operations have been employed in finding optimal path in routing problems. The congestion control problem has been formulated by fuzzy logic, and Mamdani's method has

been applied to solve the problem. The call admission control problem had been solved using multi-layered back-propagation algorithm. The chapter ended with a discussion on the architecture of a intelligent traffic controller that embodies a fuzzy congestion controller, a fuzzy bandwidth estimator and an intelligent call admission controller.

## Exercise

- Given a pair of routes:

$$r_1 = (2 \ 4 \ 5 \ 3 \ 8),$$

$$\text{and } r_2 = (2 \ 5 \ 6 \ 7 \ 3 \ 8),$$

determine  $N_c$ . Select a suitable  $n_c$ , and hence crossover  $r_1$  and  $r_2$  to yield  $r'_1$  and  $r'_2$ . What are  $r'_1$  and  $r'_2$ ?

[Answer:  $N_c = (5, 3)$ .

With  $n_c = 5$ ,  $r'_1 = (2 \ 4 \ 5 \ 6 \ 7 \ 3 \ 8)$  and

$$r'_2 = (2 \ 5 \ 3 \ 8).$$

With  $n_c = 3$ ,  $r'_1 = (2 \ 4 \ 5 \ 3 \ 8)$  and

$$r'_2 = (2 \ 5 \ 6 \ 7 \ 3 \ 8).$$

Note that in the second case  $r'_1 = r_1$  and  $r'_2 = r_2$ .]

- Consider the following paths between two nodes 2 and 5 with respective delay

Path	delay
i) (2 5)	120
ii) (2 6 5)	100
iii) (2 4 3 5)	220

Determine the weight for the path (2 5)

[Hints:  $\eta_1 = \frac{120}{120 + 100 + 220} = \frac{12}{44}$

$$\eta_2 = \frac{100}{120 + 100 + 220} = \frac{10}{44}$$

$$\eta_3 = \frac{220}{120 + 100 + 220} = \frac{22}{44}$$

$$w_1 = \frac{1/\eta_1}{1/\eta_1 + 1/\eta_2 + 1/\eta_3}$$

$$= \frac{(44/12)}{(44/12) + (44/10) + (44/22)}$$

$$= \frac{(44)(0.083)}{(44)[0.083 + 0.1 + 0.045]}$$

$$= \frac{0.083}{0.228} = 0.36 .]$$

3. Suppose the 4 paths mentioned in question 2, are submitted to the local selection pool for deletion of the route with the smallest weight among all the routes leading to destination node 5. Which of the routes (2 5), (2 6 5) and (2 4 3 5) is to be deleted?

[**Hints:** let the weights of the proposed 3 paths be  $w_1$ ,  $w_2$  and  $w_3$  respectively. Since  $w_1 : w_2 : w_3 = 1/\eta_1 : 1/\eta_2 : 1/\eta_3$ , reject the node with the smallest weight measure. Here, weight  $w_3$  will be rejected].

4. Given a fuzzy production rule:

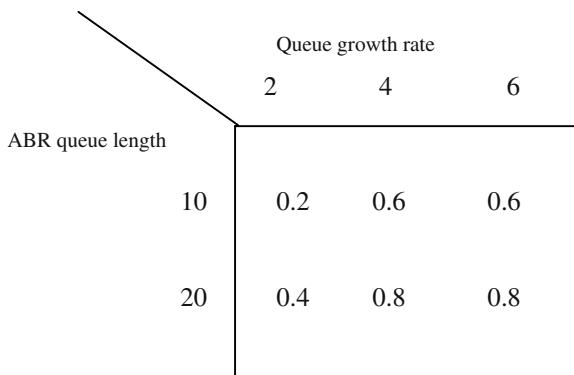
IF (ABR queue length is TOO-SHORT) and  
 (queue growth rate is DECREASING-FAST)  
 THEN (flow rate INCREASES-SHARPLY)..

Let the measured value of ABR queue length = 20, queue growth rate = 2/sec, then determine the flow rate with the following membership functions:

$$\mu_{\text{TOO-SHORT}}(\text{ABR queue length})|_{\text{ABR queue length} = 20} = 0.6,$$

$$\mu_{\text{DECREASING-FAST}}(\text{queue growth rate})|_{\text{queue growth rate} = 2/\text{s}} = 0.8,$$

$$\mu_R(\text{ABR queue length}, \text{queue growth rate})$$



Compute  $\mu_{\text{INCREASES-SHARPLY}}(\text{flow rate})$ .

[Hints:  $\mu_{\text{INCREASES-SHARPLY}}(\text{flow rate})$

$$= \text{Max} [\text{Min}\{\text{Min}\{\mu_{\text{TOO-SHORT}}(\text{ABR queue length}), \\ \forall q, g$$

$$\mu_{\text{DECREASE-FAST}}(\text{queue growth rate})\}, \mu_R],$$

where  $q = \text{ABR queue length}$  and  $g = \text{queue growth rate}$ .

$$= \text{Max} [\text{Min} \{ \text{Min} (0.6, 0.8), 0.2 \}]$$

$$= 0.2].$$

It is to be noted that only one element of the relational matrix is used in the above computation. Naturally, the question arises why? This in fact is because of the selected reading of ABR queue length and queue growth rate.

## References

- [1] Chang, C. J., Lin, S. Y., Cheng, R.-G. and Shine, Y. R., "PSD-based neural net connection admission control," *IEEE Infocom. Proceedings*, April 1997.
- [2] Chen, X. and Leslie, I. M., "A neural network approach towards adaptive congestion control in broad band ATM networks," *IEEE Global Telecommunication Conf. GLOBECOM'91*, pp. 115-119, 1991.
- [3] Cheng, R.-G. and Chang, C.-J., CAC and Computational Intelligence, In *Computational Intelligence in Tele-communication Networks*, Pedrycz, W. and Vasilakas, A. (Eds.), CRC Press, Florida, 2001.
- [4] Dijkstra, E. W., "A note on two problems in connection with graphs," *Numerische Mathematik*, vol. 1, pp. 269-271, 1959.
- [5] Hirmatsu, A., "ATM call admission control using a neural network trained with virtual output buffer method," *IEEE Int. Conf. Neural Networks*, vol. 6, 1994.
- [6] Hirmatsu, A., "Training techniques for neural network applications in ATM," *IEEE Commu. Magazine*, October 1995.
- [7] Jagielski, R. and Sekercioglu, A., "Dynamic forecast of boundaries of queue in ATM networks," *Proc. of Australian Telecommunication Networks Applications Conf.*, Melbourne, Australia, Dec. 1996.
- [8] Ogier, R., Plotkin, N. T. and Khan, I., "Neural network methods with traffic descriptor compression for call admission control," *IEEE Infocom. Proceedings*, March 1996.
- [9] Pitsillides, A. and Sekercioglu, A., *Intelligent congestion control framework for Integrated Services Communication Networks*, Technical Report TR-99-1, Dept. of Computer Sc., University of Cyprus, Nicosia, Cyprus, Jan. 1999.
- [10] Pedrycz, W. and Vasilakos, A., *Computational Intelligence in Telecommunication Networks*, CRC Press, Florida, 2001.
- [11] Tanenbaum, A. S., *Computer Networks*, Prentice-Hall, 1988.
- [12] Youssef, S., Habib, I. And Saadawi, T., "A neurocomputing controller for bandwidth allocation in ATM networks," *IEEE J. Selected Areas in Communications*, Feb. 1997.

# 21

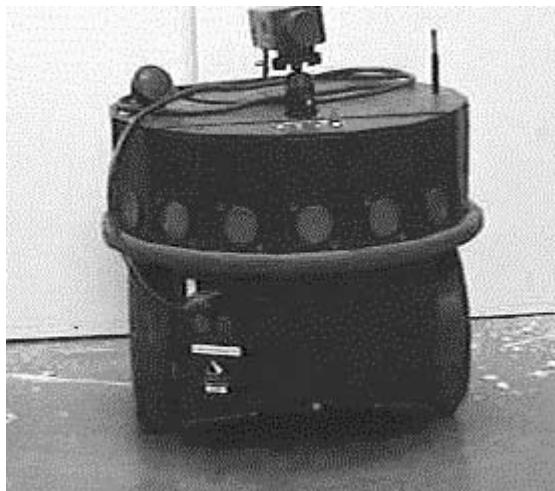
# Computational Intelligence in Mobile Robotics

*The chapter deals with mobile robots and its engineering applications. It begins with a brief introduction to the anatomy of mobile robots, and explores the scope of intelligent models in building automation for the robots. The chapter includes a comparative study of different neural topologies in path-planning application of the robots. It also outlines image segmentation and localization of a moving target in connection with the discussion on target-tracking application of the robots. The scope of extended Kalman filter in the proposed application has also been studied in detail.*

## 21.1 Introduction

A robot is a controlled manipulator capable of performing complex tasks and decision making like the human beings [28]. A mobile robot can displace itself in its workspace through locomotion or flying. Mobile robots designed for factory automation usually have a mobile platform that runs on wheels [29]. A low cost mobile robot [1], [3], [8], [9], [21], [34], [37], [38], [44] contains

ultrasonic/ laser source-detector assemblies fixed around its physical structure to sense the location of objects in its world map [29]. These transducers work on the principle of time of flight (TOF) measurement [21]. Sophisticated mobile robots [6], [33], [49], [52] include video cameras mounted on a pan and tilt platform to take snaps of the objects around it in different camera orientations. Such robots essentially have a built-in frame grabber to hold the incoming frames and a computer housed inside the robot is used to process the images in real time. An optical encoder disk connected with the rotating shaft of the wheel is employed to measure the speed of low cost mobile robots. A measure of position of the robot with reference to its initial (home) position is determined by integrating the speed over time. Modern mobile robots [11], [14], [25] also have a radio communication link with their desktop computers to share the major computational load of the robot.



**Fig. 21. 1:** Photograph of a Nomad Super Scout II mobile robot.

The work presented in this chapter is based on the experiments performed on Nomad Super Scout II mobile robots [33], [52] in Robotics and Vision laboratory of Jadavpur University. Manufactured by Nomadic Technologies Inc., USA the model super scout II comprises of 16 ultrasonic source-detector assemblies mounted around the robot's cylindrical structure (vide Fig. 21.1). It also contains one video camera with accessories, a built-in Pentium II mother - board and a radio communication link with its desktop computer through radio-modems located both at the robot and the desktop computer end. The robot has 2 independently controlled stepper motor driven side wheels and one caster wheel used for mechanical balancing of the system. The direction of motion of the robot is controlled by adjustment of the differential speed of its side wheels. For instance when the speed of both the side wheels are set equal, the robot moves

straight. When there is a difference in speed it moves on a circular trajectory. In order to roll the robot around its z-axis the speed of the 2 wheels should be set equal in magnitude but opposite in sign. The robot is configured as a client and the desktop computer as the server under a Linux environment. A number of clients may be added in the same radio-LAN as Ethernet devices. Typical TCP/IP communication protocols are used for communication of packets from the client to the server and vice-versa. The client transfers packets of range data obtained by its sonar transducer system and video frames grabbed by its camera to the desktop server. The server in turn generates control commands for motion of the robot and transfers the same to the client for its execution.

The chapter is classified as follows. In section 21.2 of the chapter, we compare the relative merits of 3 different neural algorithms in path planning application of a mobile robot for a given obstacle-map. Section 21.3 addresses the issues of image segmentation and object localization from video frames grabbed by the robot. The methodology presented in section 21.2 and 21.3 has successfully been realized in target tracking problem addressed in section 21.4. Extended Kalman filter has been employed for the prediction of the target robot's current position in the tracker robot's workspace. Conclusions are listed in section 21.5.

## 21.2 Path Planning of A Mobile Robot Using Neural Nets

The fundamental constituents of a biological nervous system are single cellular neurons, capable of processing and transmitting signals to other neurons through a nerve fiber. An artificial neural network (ANN), which is an electrical analogue of the biological nervous system, also consists of basic building blocks called neurons. Neurons in an ANN usually have small signal processing capability, and they individually or together can classify patterns into clusters. The mathematical structure and the topology of the ANN differ significantly depending on the type of their applications. In this section, the path-planning problem of a mobile robot using 3 different topologies of neural net will be addressed.

Given a starting position and a goal position in the robot's world map, the path-planning problem is concerned with determining a trajectory of the robot that does not hit an obstacle in the map. Further most of the path-planners employ constraints, such as the shortest time or the shortest path, for identifying the path uniquely. Thus, the path-planning problem, in general, is a constraint optimization problem, and no time-optimal solution to it is known till date. The time-complexity of the problem grows exponentially with an increase in the number of obstacles in the world map, and thus it is a *NP-complete* [4] problem. A number of techniques of path-planning using fuzzy logic [27], genetic

algorithm [14] and neural nets [4], [26], [47] are available in the current literature on mobile robotics [36], [45-46], [53-55] but none of these compared the performance of the algorithms on practical robots. This chapter provides a comparative study of path-planning algorithms by 3 classical neural nets with respect to a set of benchmark obstacle maps. The ANN used for this study include the back-propagation (BP) neural net [50], the radial basis function (RBF) neural net [41] and the self-organizing feature map (SOFM) neural net [24]. A performance measuring function is then used to compare the performance of the 3 algorithms.

### 21.2.1 Generation of Training Instances

The neural net based algorithms for path planning have been tested on the Nomad Super Scout II robot. The robot, as already stated, is of cylindrical shape with 16 sonar transducers mounted on its structure in a circular array with a spacing of 21.5 degrees. The sonar radiating in the heading direction of the robot is called sonar 0 (or  $S_0$  for brevity); the other sonar transducers counted in the counter-clockwise direction starting from sonar 0 are numbered 1 to 15 in an increasing order. In the path planning experiments, only the front 7 sonar transducers numbered 0, 1, 2, 3, 15, 14, 13 have been used. A schematic diagram of the sonar positions in the circular array is presented in fig. 21.2.

One important consideration in path planning problem is the computation of the goal angle. Usually, the goal angle is computed with reference to an axis of the robot. In the present experiment, the direction of radiation of sonar 12, for instance, has been considered as the reference axis. Readers unfamiliar with mobile robots may have a question: why the goal angle is not referenced with respect to an axis of the static reference frame? An answer to this is that the knowledge of the goal direction with reference to an axis of the robot is needed to orient its heading direction towards the goal.

Given the  $(x, y)$  co-ordinate of the goal position with reference to an absolute (static) reference frame, the goal angle ( $\theta_d$ ) [3] measured with respect to the radiating direction of sonar 12 of the robot is formally defined below.

**Definition 21. 1:** *The goal angle ( $\theta_d$ ), estimated at time  $t$  is defined as*

$$\theta_d(t) = \theta_{x\text{-axis}}(t) + \beta - \theta_{\text{cum}}(t) \quad (21.1)$$

where

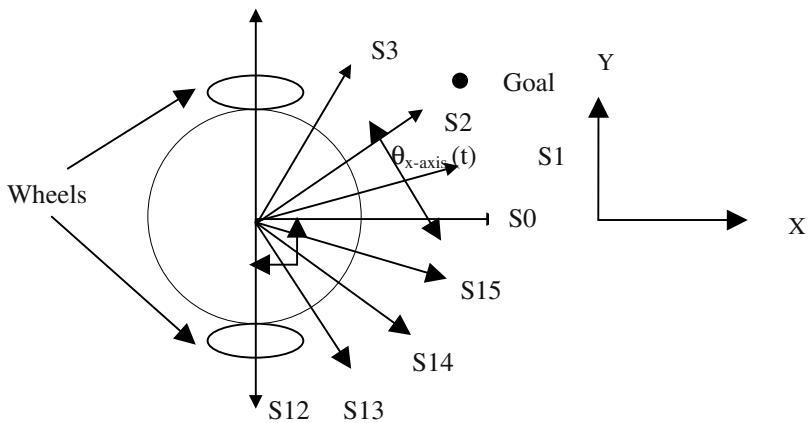
$\theta_{x\text{-axis}}(t)$  represents the goal angle with respect to the  $x$ -axis of the static reference frame which is the direction of the sonar 0 at the time of power-on of the robot;

$\beta = 90^\circ$  corresponds to the angular lead of the x-axis of the static reference frame with respect to Sonar12 ( $S_{12}$ ) direction during power on (at time  $t=0$ );

$\theta_{\text{cum}}(t)$  is the total angular shift so far undergone by the robot. It is formally defined as

$$\theta_{\text{cum}}(t) = \theta_{\text{cum}}(t-1) + \theta_{\text{current shift}} \quad (21.2)$$

where  $\theta_{\text{current shift}}$  is the angular shift of the robot at time  $t$  and is taken positive in counter-clockwise direction.



**Fig. 21.2:** The front 7 sonar assemblies of the robot and their orientation with reference to X-Y axes of the static reference frame.  $S_0, S_1, S_2$  etc. denote the sonar transducers, and the arrows associated with them denote the respective direction of radiation from these transducers.

**Example 21.1:** This example illustrates the computation of the goal angle of a robot following definition 1. Suppose, just after power-on ( $t=0$ ) the heading direction of the robot is along the x-axis of the static reference frame (vide Fig. 21.3). The robot then oriented itself like Fig. 21.4 by turning by an angle 21.5 degree clockwise around its own axis. The  $S_{12}$  axis, which was along the y-axis of the static reference frame at power-on, now has advanced by a clockwise rotation of  $22.5^\circ$ . Thus,

$$\theta_{\text{current-shift}} = -22.5^\circ \text{ and } \theta_{\text{cum}}(0) = 0^\circ, \text{ and by expression (21.2)}$$

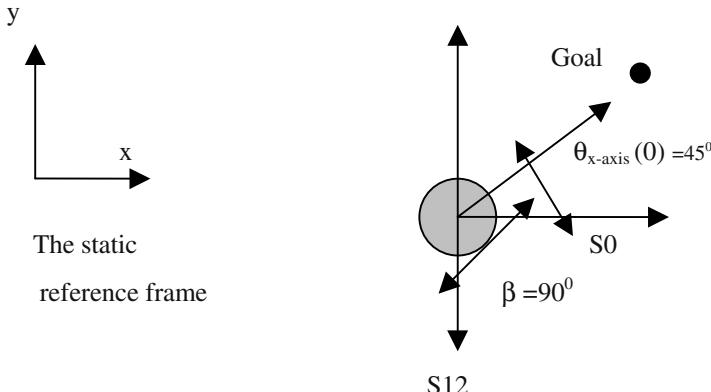
$$\theta_{\text{cum}}(1) = 0^\circ + (-22.5^\circ) = -22.5^\circ.$$

Suppose, given that  $\theta_{x\text{-axis}}(0) = +45^\circ$  we by expression (21.1) then obtain:

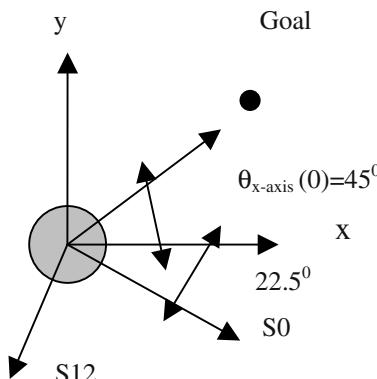
$$\theta_d(1) = 45^\circ + 90^\circ - (-22.5^\circ) = 157.5^\circ.$$

Thus, the goal angle with respect to the  $S_{12}$  axis of the robot in Fig. 21.4 is  $157.5^\circ$ .

Prior to employing neural nets in path planning, we need to train the network. Training in the present context refers to determining the next position/angular orientation of the robot in a given obstacle map for a pre-defined position of the goal in the map. For the sake of implementation, the training instances are represented by vectors of 10 components (vide Fig. 21.5). The first component is the goal angle  $\theta_d$ , the next 7 fields denote ranges  $R_i$  in the radiating direction of sonar  $S_i$ , for some integer value  $i$ , and the last 2 fields respectively denote the amplitude and orientation (clockwise/ counter-clockwise) of the necessary angular rotation of the robot. For clockwise rotation,  $d$  is set to 1, and it is set to 0 otherwise.



**Fig. 21.3:** The orientation of the robot, represented by the dark circle, just after power-on at  $t = 0$ .  $\theta_{x\text{-axis}}(0) = 45^\circ$  and  $\beta = 90^\circ$ . The  $S_0$  and  $S_{12}$  axes are parallel to the x-axis and the negative y-axis of the static reference frame.



**Fig. 21.4:** Computation of  $\theta_{x\text{-axis}}(1)$  after the robot has turned by a clockwise angle of  $22.5^\circ$ .

$\theta_d$	$R_0$	$R_1$	$R_{15}$	$R_2$	$R_{14}$	$R_3$	$R_{13}$	$\theta_m$	$d$
------------	-------	-------	----------	-------	----------	-------	----------	------------	-----

**Fig. 21.5:** The generalized training vector used in all 3 types of neural nets.

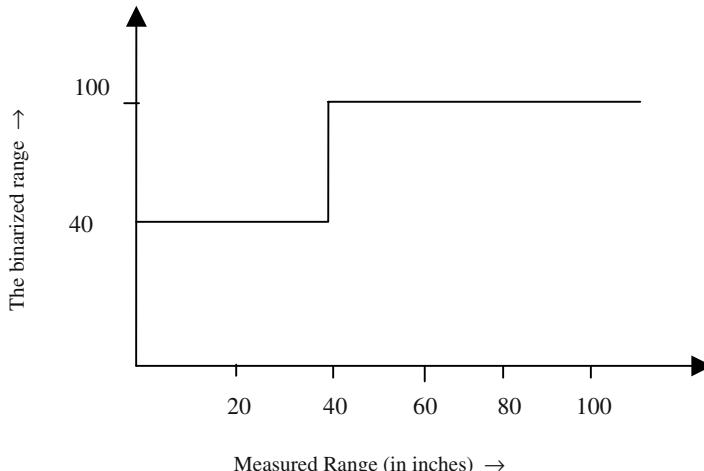
For the Super Scout II robot, range is a integer value between 0 to 255 inches. Since the training vector has 7 range fields, the number of combinations of these 7 range fields is  $256^7=2^{56}$ . Training a typical neural net by the Back-propagation algorithm, for instance, with so large number of training instances needs hundreds of hours even on a high-speed workstation. Alternatively, we can reduce the number of training instances through binarization of the range fields into 2 distinct levels, say 40 and 100 inches depending on a measured threshold range,  $R_{th}$ . Fig. 21.6 describes the binarization process of range with a  $R_{th} = 40$  inches. Any range value  $R_i$  less than or equal to  $R_{th}$  will be assigned a value 40 and 100 otherwise. The binarization process thus reduces the possible values of a range variable into 2 discrete values indicating that the sonar is blocked (when range  $\leq 40$  inches) and free (when range  $> 40$  inches).

$\theta_d$  and  $\theta_m$  are also discretized into 7 possible values. Thus the total number of training instances is reduced to  $7 \times 2^7 \times 7 \times 2 = 12544$ , where the first and the third 7 corresponds to  $\theta_d$  and  $\theta_m$  respectively,  $2^7$  corresponds to the 7 ranges and the last 2 for direction of rotation  $d$ . It is needless to mention that the figure 12544 also is very high, and so we selected judiciously only 650 out of these 12544 instances for training the neural nets. Further, to keep the measure of the R.M.S value of the training error within a permissible limit of 0.001, the 650 instances were further sub-divided into two halves. The first half comprising of 325 instances corresponds to training with only one large obstacle in the map. The second half containing the remaining 325 instances corresponds to training with multiple large sized obstacles in the map. The neural nets to be used for path planning thus should be trained with 2 sets of training instances. Training with 325 instances can keep the R.M.S value of training error within a permissible limit of 0.001. The following principles have been employed in the construction of training instances.

### Principles Employed for the Construction of the Training Set

The primary aim of the navigational planning is to move the robot towards the goal point, without colliding with any obstacles preferably by the shortest route. The following principles [3] have been followed in designing the training instances to achieve the objective.

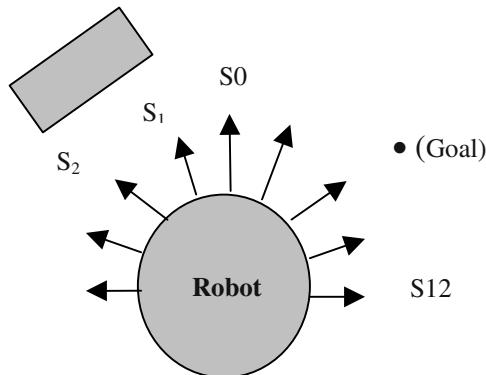
- i) The robot should rotate around its axis to keep its front 3 sonars S<sub>0</sub>, S<sub>1</sub>, S<sub>15</sub> free and should translate only if there is no obstacle in front of the said 3 sonars.
- ii) The robot should attempt to align its heading direction S<sub>0</sub> with the goal direction, while satisfying condition i). Thus in all the cases, angular deviations from the goal are to be minimized, if it is found impossible to make it zero.
- iii) If the situation allows the robot to choose between either of 2 angles of rotation, then the robot should rotate by the smaller angle around its axis, while in both the cases the front 3 sonars should remain free and the angular deviations from the goal is same.
- iv) If the angle to be turned by the robot to get itself aligned with the goal direction is same in both directions and the front 3 sonars remain free in either direction, then the robot should arbitrarily set a bias on the clockwise direction.
- v) If the heading direction of the robot is towards the goal (i.e. sonar 0 faces the goal direction) and the front 3 sonars are free, it will translate towards the goal.



**Fig.21.6:** The binarization process of range measured by sonar transducers.

Examples below illustrate the construction of training instances following the above principles. Examples 21.2 and 21.3 illustrate the cases with a single and multiple obstacles in the map respectively.

**Example 21.2:** Consider the world map shown in Fig. 21.7. Here, the goal is at an angle  $45^\circ$  with respect to the  $S_{12}$  axis of the robot and the sonar 1 and 2 are facing the large rectangular obstacle. Thus the range fields  $R_1$  and  $R_2$ , shown in Fig. 21.5, are 40 inches, while the other 5 range values should be 100 inches. Under the given situation (Fig. 21.7), the robot should turn by a clockwise angle of  $45^\circ$  to align its heading direction  $S_0$  towards the goal. So,  $\theta_m$  is  $45^\circ$  and the direction field  $d$  is set to 1 to represent the counter-clockwise rotation of the robot. The training instance corresponding to the said world map is given in Fig. 21.8 for convenience.

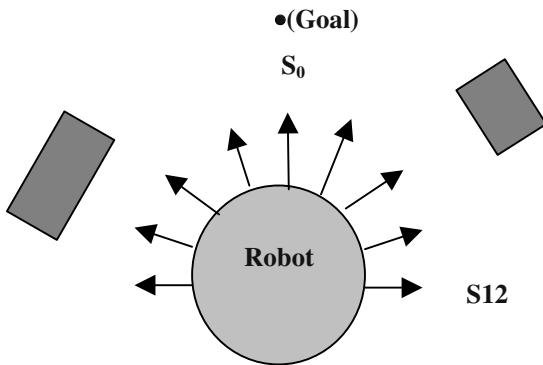


**Fig. 21.7:** A typical world map of the robot with only one obstacle used to illustrate the construction of a training instance.

45	100	40	100	40	100	100	100	45	1
----	-----	----	-----	----	-----	-----	-----	----	---

**Fig. 21.8:** The training vector corresponding to the world map depicted in Fig. 21.7.

**Example 21.3:** In this example we consider two large sized obstacles in the robot's world map (vide Fig. 21.9). Here, the goal is at an angle of  $90^\circ$  with respect to the sonar 12 i.e. is along sonar 0. The sonar readings show that there is no obstacle along the goal direction. Consequently, the robot should translate in the forward direction to satisfying the proposed principles. The training instance corresponding to the map of Fig. 21.9 is presented in Fig. 21.10.



**Fig. 21.9:** Illustrating the construction of a training instance with 2 obstacles in the map.

90°	100	100	100	40	40	40	60	0°	x
-----	-----	-----	-----	----	----	----	----	----	---

**Fig. 21.10:** The constructed training instance for the map shown in Fig. 21.9. The last field here presumes a don't care (x) value.

### 21.2.2 Configuring the Neural Nets

In this section, 3 classical Neural Net topologies will be discussed briefly in connection with their applications in the path-planning problem.

**Configuring the 3-layered feed-forward neural net for BP learning:** A 3-layered feed-forward neural net with 8 neurons at the input layer, 2 at the output layer and 8 neurons at the hidden layer has been considered for the proposed path-planning application. The first 8 fields of the training instance (vide Fig. 21.5) that corresponds to the measured data of the robot are mapped at the neurons of the input layer, while the last 2 fields are mapped at the corresponding neurons of the output layer. A sigmoid type non-linearity has been employed for all neurons of the neural network. The mean square error sum of the neurons at the output layer has been used as the measure of the performance of the BP training algorithm. The training cycles are continued until the mean square error sum of all neurons at the output layer corresponding to the training instances goes below a pre-defined threshold of  $10^{-3}$ .

**Configuring the RBF neural net:** The RBF Neural Net used in this specific application consists of 3 feed-forward layers with 8 neurons at the input layer, 8 at the hidden layer and 2 at the output layer. The measured first 8 fields of the training instance are mapped at the neurons of the input layer, while the last two fields are mapped at the neurons of the output layer. Each neuron at the hidden layer is tuned to a fixed Gaussian type RBF given by

$$f(r_i) = \exp(-r_i^2/\sigma_i^2) \quad (21.3)$$

where

$$r_i = \|X_i - C_i\|$$

$X_i$  is a 10-dimensional input vector, the components of which correspond to the fields of the training instances (vide Fig.21. 5).

$C_i$  is a 10-dimensional cluster center obtained by averaging the corresponding fields of  $X_i$  under a given cluster. The clusters are selected by judiciously identifying the first and the last 2 fields of the training instances such that the above fields of  $X_i$  do not have a large variance.

$\sigma_i^2$  denotes the variance of the neighborhood  $m$  number of points, each corresponding to a different  $X_i$ ,  $1 \leq i \leq m$  around the cluster center  $C_i$ . Formally,

$$\sigma_i^2 = (1/m) \sum_{i=1}^m \|X_i - C_i\|^2 \quad (21.4)$$

**Configuring the SOFM neural net:** A  $(8 \times 8)$  2-dimensionbal space of neurons have been used as the first layer of the SOFM [17]. The training instances, each of which corresponds to a point in 10-dimensional space are mapped to the neurons in the first layer of the SOFM by satisfying the following criterion

**Criterion:** Select the neuron  $N_{k,l}$  from the set of neurons  $N_{i,j}$ ,  $1 \leq i, j \leq 8$ , such that for a training instance  $X_r$

$$\min_{1 \leq i \leq n} \min_{1 \leq j \leq n} \|X_r - W_{i,j}\| = \text{Min} [\text{Min}_{1 \leq i \leq n} \min_{1 \leq j \leq n} \|X_r - W_{i,j}\|] \quad (21.5)$$

Once the  $(k, l)^{th}$  neuron is selected on the 2D plane, the weights of its neighborhood are adjusted by

$$W_{ij}(t+1) = W_{ij}(t) + \eta \|X_r - W_{ij}\| \quad (21.6)$$

In the proposed application, we considered a square neighborhood of  $(3 \times 3)$ . The selection of neurons and adaptation of their weights is continued for each training instance. In case, more than one training instance is mapped onto the same neuron, a second plane of  $(4 \times 4)$  neurons is considered for their subsequent mapping onto the new plane. Further, if a coincidence of more than one training instance is detected, another plane of  $(3 \times 3)$  is considered for mapping the training instances onto distinct neurons of the new plane. In the recognition phase, thus we need a depth-first traversal in hierarchical organization of the SOFM to determine the nearest instance corresponding to an unknown sensory vector. It is worth-mentioning here that the last 2 fields of the application instances is to be recovered from the best matched training instance of the SOFM structure, when the matching is carried out by determining the Euclidean distance with respect to the first 8 fields of the application instances.

### 21.2.3 Parameters Used for Performance Evaluation of the Neural Path-Planning Algorithms

The following heuristic measures used in performance evaluation of the path-planning algorithms by neural nets are presented in order for the subsequent benchmark analysis.

**Definition 21. 2:** *Traversal Time (T) is defined as the time taken by the robot to move from a given starting position to a fixed goal position in its workspace. For the estimation of the traversal time, the Linux built-in function ‘`gettimeofday()`’ is used twice, once at the beginning and once at the termination of a journey. Let these two times be  $t_s$  and  $t_f$  respectively. Then the traversal time*

$$T = t_f - t_s \quad (21.7)$$

**Definition 21.3:** *Total Distance of Traversal (or Traversal Distance in brief) is estimated by summing up the traversal by the robot between successive generation of control commands for motion. It is denoted here by D. Let  $(x_i, y_i)$  &  $(x_{i-1}, y_{i-1})$  be the current and the previous position of the robot in the map. Thus, the total distance of traversal for successive n positions, is computed by*

$$D = \sum_{i=1}^n (((x_i - x_{i-1})^2 + (y_i - y_{i-1})^2)^{0.5}) \quad (21.8)$$

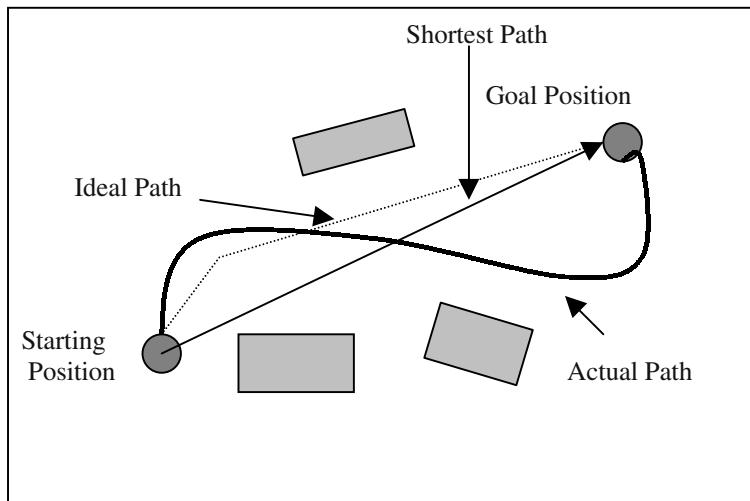
**Mean Path Deviation ( $\sigma_p$ )** is a very important measure to determine the accuracy of the path of traversal by the robot. We formally define the mean path deviation  $\sigma_p$  for a given path P by evaluating the distance of the actual path

traversed w.r.t the ideal path of traversal of the robot between the same terminal points. Unfortunately, computing the ideal path of traversal itself is a complex problem that requires a polynomial time. So without loosing much accuracy, we define the distance of the actual path traversed w.r.t the shortest path of traversal (even though the shortest path is occluded by obstacles; see Fig. 21.11).

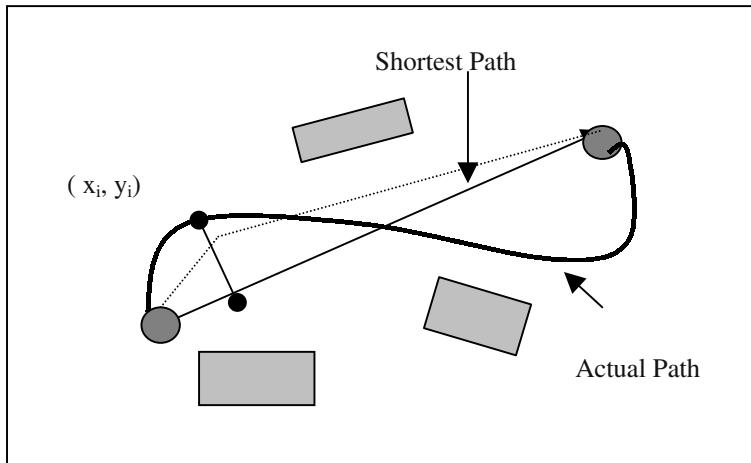
**Definition 21.4:** *The computation of the parameter  $\sigma_P$  for a given path  $P$ , thus can be formally defined as follows-*

$$\sigma_P = \left[ \sum_i (d_{ij})^2 / N \right]^{0.5} \quad (21.9)$$

where  $d_{ij}$  denotes the perpendicular distance from a point  $i$  on the traversed path by the robot over the ideal path (or shortest path) at point  $j$  and  $N$  is the number of points taken on the actual path of the robot. The parameter  $d_{ij}$  can be best-explained using Fig.21.12.



**Fig. 21.11:** The shortest path and the ideal path of a robot for a given starting and a goal position in the map.



**Fig.21.12:** Illustration of the computation of  $d_{ij}$ .

**Definition 21.5:** A *Performance Index (PI)* that is used to compare the performance of different path planning algorithms is defined as the sum of the linear combination of the heuristic measures presented earlier. Thus,

$$PI = \alpha_1 T' + \alpha_2 D' + \alpha_3 \sigma_p' \quad (21.10)$$

such that

$$\sum \alpha_i = 1 \quad (21.11)$$

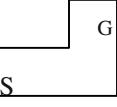
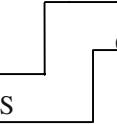
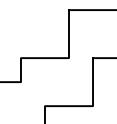
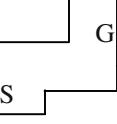
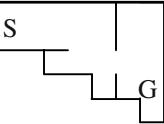
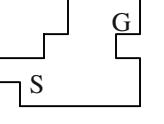
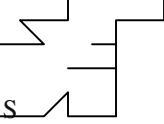
and  $0 < \alpha_i < 1$  for all  $i$ .

$T'$ ,  $D'$ ,  $\sigma_p'$  denote normalized traversal time, normalized traversal distance and normalized mean path deviation respectively. For a given path and a given algorithm, normalized traversal time is defined as the ratio of the traversal time to the sum total traversal time of all the 3 algorithms. The normalized traversal path and the normalized mean path deviations are defined similarly.

#### 21.2.4 Experimental Results- a Benchmark Analysis

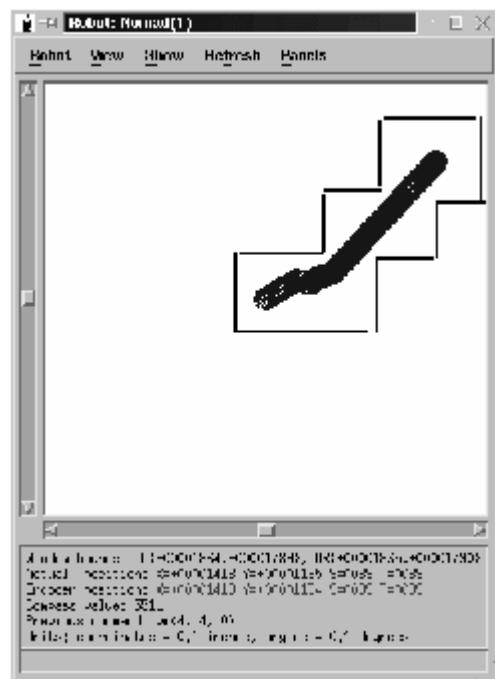
For comparing the performance of various neural path-planning algorithms, we consider 50 benchmark obstacle maps of diverse complexities to represent the robot's workspace. The programs for path planning using the proposed 3 neural topologies were simulated and also tested on a real Nomad Super Scout II mobile robot platform. In the proposed path planning programs, the sensory range data and the information about the goal angle are used to determine the

**Table 21.1.** Performance evaluation of the proposed neural path planning algorithms using 7 selected obstacle maps. (S and G in the map correspond to the starting and the goal positions)

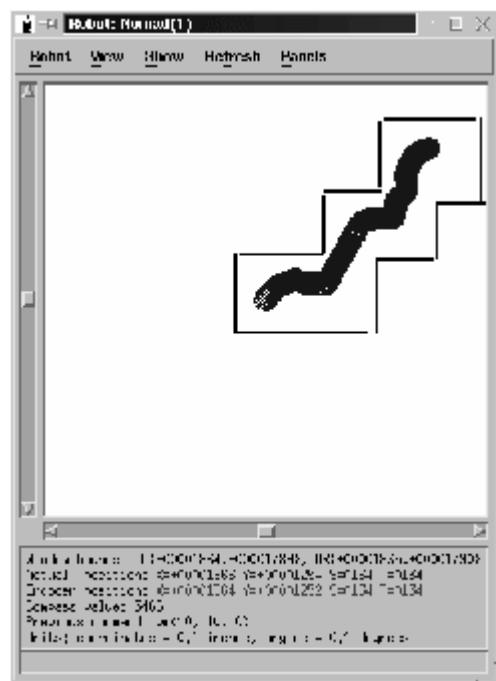
Typical Benchmark Paths	Neural Net Used	Traversal Time (sec)	Traversal Distance (inch)	Mean Path Deviation (inch)	Performance Index
	BP RBF SOFM	197 183 247	119.2 117.7 116.4	27.3 32.0 30.6	0.313 0.307 0.363
	BP RBF SOFM	348 396 1267	192.7 194.5 186.2	64.9 65.9 54.7	0.258 0.276 0.466
	BP RBF SOFM	248 383 644	185.2 210.5 225.9	60.3 74.7 73.3	0.242 0.326 0.466
	BP RBF SOFM	239 244 422	159.5 159.8 157.8	6.6 7.2 6.5	0.295 0.308 0.397
	BP RBF SOFM	210 221 618	146.2 147.5 146.9	11.4 11.5 11.9	0.265 0.270 0.463
	BP RBF SOFM	305 361 459	173.6 195.9 192.9	10.8 11.3 14.88	0.184 0.323 0.393
	BP RBF SOFM	259 455 534	143.6 140.7 137.2	10.8 11.0 11.2	0.272 0.345 0.383

subsequent heading direction of the robot. This is realized by turning the robot around its axis and a fixed displacement is given in the direction of S<sub>0</sub> (heading direction). Thus, the robot moves from one point to the next point in its workspace. The above process is repeated until the robot reaches the predefined goal point. A summary of performance analysis of the paths generated by the robot with respect to 7 typical benchmark world maps is given in Table 21.1. The values of  $\alpha_1, \alpha_2, \alpha_3$  are chosen as 0.5, 0.2 and 0.3 respectively, based on the priority of the heuristic measures in expression 21.10 and the constraint equation 21.11. The lower the value of PI, the better is the performance of an algorithm in a given obstacle map.

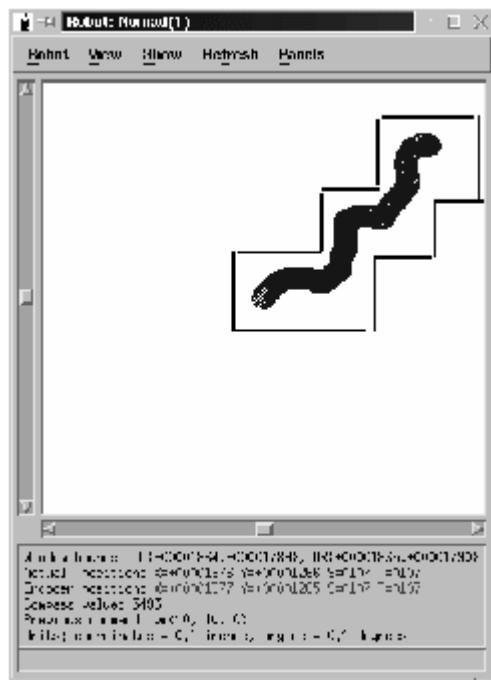
Fig. 21.13 provides a specific instance (row 3) of Table 21.1 as an illustration. It is apparent from Fig. 21.13 that Back-propagation algorithm generates a straight path, whereas RBF and SOFM generate zigzag trajectories for the robot.



(a)



(b)



(c)

**Fig. 21. 13:** Path planning of a mobile robot by (a) Back-propagation, (b) RBF and (c) SOFM neural net for the fixed starting and the goal positions. The long thick black patch in the above map describes the path of traversal of the robot towards the given goal point. The GUI is available with the Nomad simulator. The map in this figure corresponds to row 3 of Table 21.1.

### Criteria for the selection of the Benchmark Maps

To test the learning behavior of the different neural algorithms, a set of sample maps that explores the feasibility of motions of the robot in an exhaustive sense have been constructed. Map 1, for instance, requires only one 90° counter-clockwise rotation of the robot. The map 2 calls for one 90° counter-clockwise rotation followed by another 90° clockwise rotation of the robot. For traversal in map 3, the robot needs to rotate 45° counter-clockwise, followed by another 45° clockwise rotation. In map 4, the robot has to turn first 90° counter-clockwise, followed by 90° clockwise and finally 90° counter-clockwise. The map 5 includes more constraints to pass the robot through a small spacing and needs two 45° clockwise rotations. The map 6 and 7 are similar to map 2 and 3 respectively but with more constraints. To take into account all possible actions of the robot we altogether experimented with 50 such maps. Only 7

representative samples out of these are presented in Table 21.1. The results obtained from all the 50 sample maps, however, are more or less similar.

### 21.2.5 Implication of the Results

The section 21.2 examined the performance of 3 classical neural learning algorithms in path-planning application of a mobile robot. A performance index has been employed to compare the relative merits of the algorithms in runtime. Computation of this index for different maps reveals that the BP algorithm is performance-wise superior to the other neural algorithms. However, a deep look into Table 21.1 envisages that individual measure such as total traversed distance or mean path deviation is not necessarily minimum for the BP algorithm. The result being counter-intuitive is of major concern from a practical point of view. It is needless to say that the BP algorithm requires a significant amount of training time. This large training cost is justified only when the algorithm outperforms others in all respects. Unfortunately, the situation here is not so much promising always. The question that naturally arises: why?

A further look at the table reveals that for complex paths the BP algorithm works relatively better than others, but its performance w.r.t  $D$  and  $\sigma_p$  in simpler maps is comparable with the other 2 algorithms. Since BP is trained with a large number of instances, a noisy measurement of range data forces the input to be interpreted like a different map, consequently causing the network to select a wrong direction of motion. But when the spacing of the obstacles with respect to the possible direction of motion is large, the BP selects the right path. The SOFM and RBF fortunately are less sensitive to noise, as they have a natural inherent clustering behavior. Consequently,  $D$  and  $\sigma_p$  for smaller paths are comparably less for RBF and SOFM with respect to those of the BP algorithm.

## 21.3 Object Localization

Once the robot and other objects, which have sufficiently low gray scale values, are segmented from the image by FCM clustering algorithm [5], [20], [22], [31], [42] (vide chapter 5), the task that remains at hand is to localize [15] the robot solely from the other black objects. This is however difficult because it involves separating the geographically isolated objects in the image. The most widely used technique for object localization is region growing. In the present scenario, the conventional region-growing technique fails because the seed point, which needs to be initialized in the algorithm, cannot always be located within our object of interest. Moreover, high time complexity of the algorithm renders it automatically vulnerable for real time applications like navigation or target tracking.

After considering all these constraints, a novel algorithm for object localization has been devised [11]. One useful feature that is used at this point is the size information of the object under consideration. The essences of this algorithm lie in splitting the FCM clustered image into some fixed size windows/blocks and determine whether each window is red. The next task is to identify the connected windows by testing the 4-connectivity between them. Then the 4-connected windows are clubbed together to form the separate geographic regions.

The selection of the experimental window size is a pertinent parameter to be determined judiciously. A large sized window may include more than one object. For example, a large window may include the target object with others. A small window size is good for accuracy, but a too small window demands a significant computational time. The size of the window chosen in the present application is of  $12 \times 16$  pixels. In fact, the choice of 4-connectivity testing over that of the 8-connectivity testing for identifying geographically connected regions is also guided by the same consideration that the whole computation process should be amenable to a real-time application. A procedure is presented below, describing the various steps of the object localization process.

### **Procedure Localize-Object**

**Input:** FCM clustered image (containing the robot and other black objects as one cluster and every other object as another cluster)

**Output:** An image containing localized target object, well separated from noise and other objects having similar type of coloration as that of the target.

**Begin**

**Step1:** partition the input image into  $m \times n$  number of equal sized blocks;

**Step2: For** each block  $(i, j)$  **do**

**Begin**

**For** each pixel within a block **Do**

**Begin**

**If** intensity of pixel  $\leq 20$

**Then** declare it dark and increase  
dark\_pixel\_count\_block  $(i, j)$  by 1;

**End For;**

**Step3:** **If** dark\_pixel\_count\_block  $(i, j) > 50$

**Then** mark block  $(i, j)$  as dark;

**End For;**

**Step4:**  $k := 1$ ;

**Repeat**

Region[k] = any dark block  $(i, j)$ ;

Region\_count[k] = 0;

**For** each dark block  $(i, j)$  in Region[k]

**If** its neighbors are dark,

**Then do**

---

```

Begin
Region[k]: =Region[k] ∪ (block(i, j));
Region_count[k]: = Region_count[k] +1;
End For;
Mark Region[k] dark;
k: =k+1;
Until no dark blocks remain;
K: =kmax;

```

**Step 5: For k: =1 to k<sub>max</sub>**

Identify the Region with the largest Region\_count[k] and

Call it target;

**End For;**

**End.**

The procedure Localize-Object aims at identifying the locations of the target object in a segmented image. In the present context, the target object has been assumed to occupy the largest area. Thus identifying the largest dark region in the image suffices our purpose.

The procedure comprises of 5 main steps. In step1 we partition the given segmented image into m. n number of equal sized blocks. Step2 of the algorithm identifies dark pixel based on their intensity values. For this implementation, we select a threshold of 20. Thus if pixel intensity is less than 20, it is declared dark. The dark pixel count in each block is also determined in this step. Step3 marks a block dark if its dark pixel count exceeds 50. Step4 of the procedure assembles neighborhood dark blocks in the segmented image into regions such that each two regions are disjoint. In step5 of the algorithm we declare the largest region as the target object. The localization of the robot from the segmented image of Fig. 21.14(a) is presented in Fig. 21.14(b) for convenience.

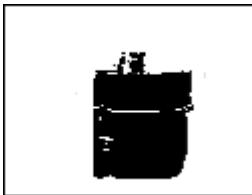
In case the largest region does not correspond to the target object, the regions need to be sorted in descending order based on their block counts. Now a shape-matching algorithm may be invoked for comparing the boundary of the target object with each region in the list in sequence. The region having the closest resemblance with the reference object shape is declared as the target.

The process of image localization presented in this section has been utilized to build a real-time system for vision based target tracking and co-operation schemes in mobile robotics. The novelty of the approach lies in a successful merging of the FCM algorithm for image segmentation with the “windowing-and-connectivity-checking” method for image localization. One interesting and noteworthy issue about the application of the FCM algorithm is the selection of the logarithm of the pixel intensities as the feature. Such selection is important for segmenting a dark target from a background of comparable (but unequal) darkness. If gray scale intensities, instead of their logarithms, are directly used

as the feature then segmentation of comparable dark regions in an image is not feasible.



(a)



(b)

**Fig. 21.14:** The target robot (b) has been isolated from the image (a) by “windowing and connectivity checking”.

## 21.4 Target Tracking and Interception by Mobile Robots Using Kalman Filtering

Target tracking is a problem of common interest for researchers of numerous domains. The concept of target tracking classically emerged from the disciplines of control engineering. The concerned problem in the present context is to predict the online trajectory of a moving target by a given tracker. Usually the speed of response of the tracker is higher or at least comparable to that of the target. Since the intelligent target and the tracker both use the same level of technology, it is expected that their speed of response is more or less comparable. Designing an intelligent tracker under such circumstance is really a complex problem. In this section, we present the design and implementation aspects of an intelligent tracker. The tracker employs a video camera to capture the images of the moving target for segmentation and localization of the target in its image. It then identifies the location of the target by a range finder and consequently plans a path towards the target by using the knowledge of the

obstacle map in its workspace. An overview of the proposed scheme of target tracking is outlined below.

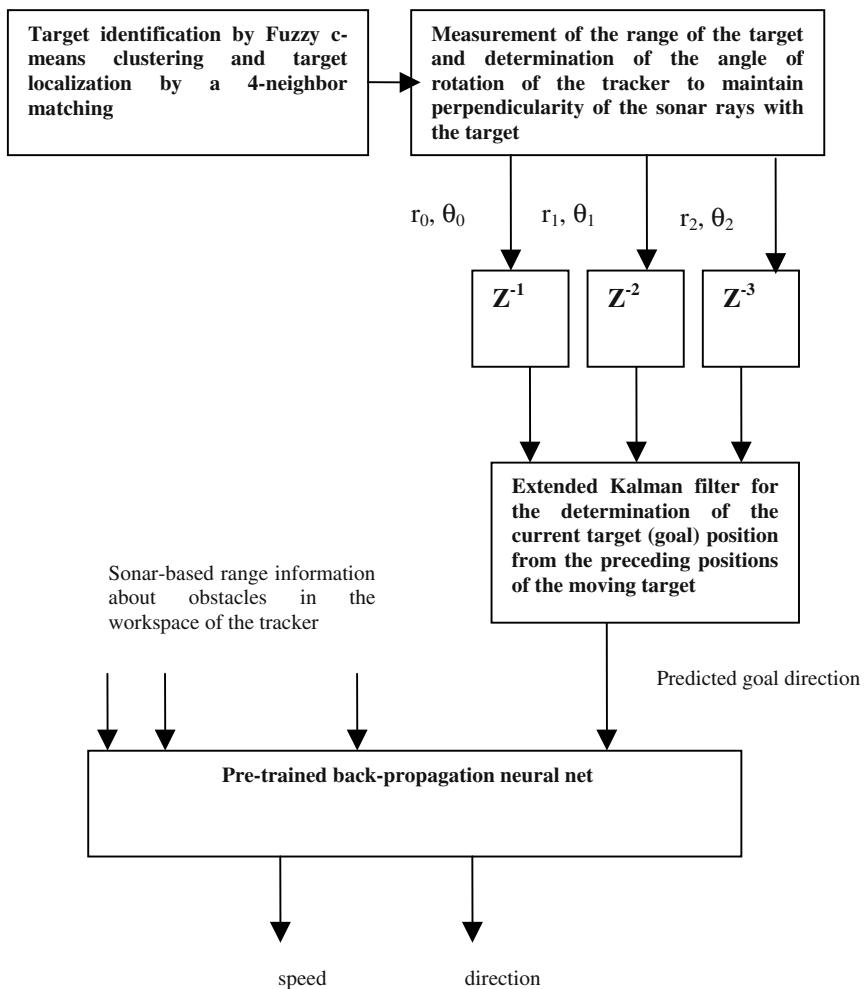
For realization of the target-tracking scheme, 2 mobile robots identical in all respects have been configured as the target and the tracker. Each robot has its own desktop server and is equipped with a movie type video camera, a video frame grabber, a radio communication system and 16 ultrasonic transducers mounted around the periphery of the robot at uniform spacing. The target robot is controlled to move on a fixed trajectory by a control program running at its desktop server. The tracker robot on the other hand receives sensory information by using the video camera and the ultrasonic sensors. The received real time video frames collected by the tracker robot are first transferred to its server. The server preprocesses the image and segments it into objects of interest (here the target robot). A fuzzy clustering technique presented in chapter 5 is then invoked for segmentation and subsequent localization of the target. After the target is identified in the image, its distance and orientation with respect to some reference axis of the tracker needs to be determined. The following scheme has been undertaken to evaluate the polar co-ordinate of the target with reference to two mutually perpendicular axes of the tracker.

First, the tracker robot should activate its sonar transducers. The particular sonar transducer radiating its beam towards the target gives an approximate measure of the range of the target with respect to the tracker. The polar co-ordinate ( $r, \theta$ ) of the target with respect to a reference x-axis, say  $S_{12}$ -axis, of the tracker is then determined.

The sonar readings in all other directions of the tracker describe the range of the obstacles in the tracker's world map. These readings along with the estimated range of the target can be fed to a pre-trained neural net for controlling the speed and direction of motion of the tracker. Since the control decision about the motion generated by the neural net is based on the sampled (sonar/ vision) data of the last cycle, the tracker may occasionally be misled by wrong control commands. To overcome the limitation, a 'tracker motion predictor' may be employed in the system. The predictor should provide the possible current direction of motion of the target from its preceding positions. This has been realized by embedding an extended Kalman filter in the proposed scheme. A complete schematic diagram of the proposed tracking system is presented in Fig. 21.15

The proposed work has a number of merits with respect to traditional target tracking systems. First, most traditional systems are designed with the pre-assumption that speed of the target is less than that of the tracker. Fortunately, the proposed design is free from such constraints. Secondly, conventional trackers usually do not estimate the range of the target and therefore do not pay much attention to velocity modulation of the tracker. The

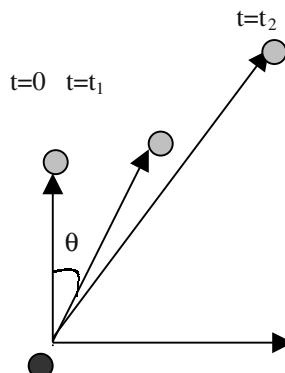
present work, however, takes care of the range measurement of the dynamic target, and consequently adjusts the velocity of the tracker on-line. Thirdly, accuracy in range estimation along near-linear trajectories has shown significant improvement in this presentation by employing extended Kalman filtering. Lastly, the proposed tracker requires insignificantly small time of the order of 400 milliseconds only to respond to a change in target position. Finally, the back-propagation algorithm being trained with quite a large number of sensory-response instances selects the right step of discrete motion of the tracker. The mean square error in position of the tracker with respect to the dynamic target position thus diminishes in most cases.



**Fig. 21.15:** A schematic diagram of the proposed target– tracking scheme.

### 21.4.1 Measurements of the Input to Kalman Filter

The Kalman filter employed in the tracking system can predict the current position of the target from its preceding positions. The accuracy in prediction by a Kalman filter greatly depends on the time gap between successive data samples. The time needed for on-line image registration, segmentation, localization and neural net activation being of the order of 400 milliseconds, the sampling interval cannot be fixed less than 400 milliseconds. The prediction of the current position of the target from its last, say 3 positions indirectly means exciting the filter with the sonar data collected ( $400 \times 3$ ) milliseconds = 1.2 seconds earlier. Since the speed of the robots is considerably high (around 20 inches/ second), the predicted current position of the target may suffer from inaccuracy. To overcome the above problem the tracker is designed to work in 2 phases. In the first phase, the tracker is given a controlled rotation around its z-axis so that it can direct its vision system to grab 3 successive frames of the target. Let the polar co-ordinates of these 3 positions be  $(r_0, \theta_0)$ ,  $(r_1, \theta_1)$  and  $(r_2, \theta_2)$  respectively. All these measurements are performed by considering the  $S_{12}$ -axis of the Super Scout II robot as the reference x-axis and  $S_0$  as the reference y-axis.

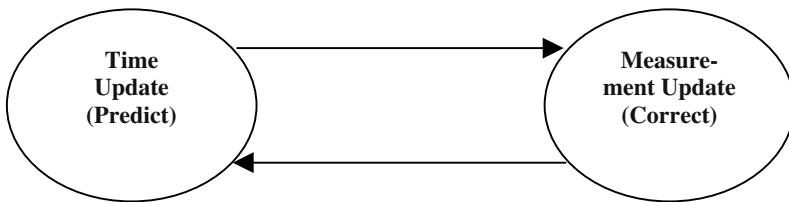


- The tracker robot in its observation phase locates the target
- The target robot in three different positions.

**Fig 21.16:** The tracker observing the motion of the target.

Fig. 21.16 describes 3 successive positions of the target in the measurement update phase of the tracker. The time  $t=0$ ,  $t=t_1$ ,  $t=t_2$  correspond to the image sampling times associated with these measurements. After the measurement update phase is over, the tracker switches to prediction phase. In the prediction phase, the tracker determines the current position of the target from its preceding

positions ( $r_i, \theta_i$ ) for  $i=0$  to 2. Once the prediction phase is over, the tracker starts its next measurement update phase, and the process continues until the tracker intercepts the target. A schematic view of a tracking cycle comprising of the measurement update phase and prediction phase is presented in Fig. 21.17.



**Fig.21.17:** The recursive Kalman filter cycle.

### 21.4.2 Extended Kalman Filter- an Overview

Kalman Filtering [7], which basically stems from statistical estimation theory, has tremendous applications in robotics. A Kalman filter is a digital filter that provides a recursive solution to an estimation problem. It is impossible to explore such a vast and important mathematical tool in a few paragraphs. In the present section, the principle of extended Kalman filtering is just outlined to demonstrate its application in target tracking.

An extended Kalman filter [39] is a digital filter that attempts to minimize the measurement noise for estimating a set of unknown parameters, linearly related with a set of measurement variables. The most important significance of this filter is that it allows recursive formulation and thus improves accuracy of estimation up to users' desired level at the cost of new measurement inputs.

Let

$f_i(x_i, a) = 0$  be a set of measurement equations describing relationships among an estimator vector  $a$  and measurement variable vector  $x_i$ ,

$x_i^* = x_i + l_i$ , where  $l_i$  is a white Gaussian type measurement noise such that  $E[l_i] = 0$ ,  $E[l_i l_i^T] = \text{positive symmetric matrix } A_i$ , and  $E[l_i l_j^T] = 0$ ,

$a_{i-1}^* = a + s_{i-1}$ , where  $s_{i-1}$  is a white Gaussian type estimation noise such that  $E[s_{i-1}] = 0$ ,

$E[s_{i-1} s_{i-1}^T] = \text{positive symmetric matrix } S_{i-1}$  and  $E[S_{i-1} S_{i-1}^T] = 0$ .

Expanding  $f_i(x_i, a)$  by Taylor's series around  $(x_i^*, a_{i-1}^*)$ , we find

$$\begin{aligned}
 f_i(\mathbf{x}_i, \mathbf{a}) &= f_i(\mathbf{x}_i^*, \mathbf{a}_{i-1}^*) + (\partial f_i / \partial \mathbf{x})(\mathbf{x}_i - \mathbf{x}_i^*) + (\partial f_i / \partial \mathbf{a})(\mathbf{a} - \mathbf{a}_{i-1}^*) \\
 &= 0.
 \end{aligned} \tag{21.12}$$

After some elementary algebra, we find

$$\mathbf{y}_i = \mathbf{M}_i \mathbf{a} + \mathbf{w}_i \tag{21.13}$$

$$\text{where } \mathbf{y}_i = -f_i(\mathbf{x}_i^*, \mathbf{a}_{i-1}) + (\partial f_i / \partial \mathbf{a})(-\mathbf{a}_{i-1}^*) \tag{21.14}$$

is a new measurement vector of dimension ( $p_i \times 1$ ).

$$\mathbf{M}_i = (\partial f_i / \partial \mathbf{a}) \tag{21.15}$$

$$\text{and } \mathbf{w}_i = (\partial f_i / \partial \mathbf{x})(\mathbf{x}_i - \mathbf{x}_i^*) \tag{21.16}$$

is a measurement noise vector of dimension ( $p_i \times 1$ ). We also want that  $E[\mathbf{w}_i] = 0$  and define

$$\mathbf{W}_i = E[\mathbf{w}_i \mathbf{w}_i^T] = (\partial f_i / \partial \mathbf{x}) \mathbf{\Lambda}_i (\partial f_i / \partial \mathbf{x})^T. \tag{21.17}$$

$$\text{Let } \mathbf{S}_i = E[(\mathbf{a}_i - \mathbf{a}_i^*)(\mathbf{a}_i - \mathbf{a}_i^*)^T] \tag{21.18}$$

An attempt to minimize  $\mathbf{S}_i$  yields the filter equations [32], given by:

$$\mathbf{a}_i^* = \mathbf{a}_{i-1}^* + \mathbf{K}_i (\mathbf{y}_i - \mathbf{M}_i \mathbf{a}_{i-1}^*) \tag{21.19}$$

$$\mathbf{K}_i = \mathbf{S}_{i-1} \mathbf{M}_i^{-T} (\mathbf{W}_i + \mathbf{M}_i \mathbf{S}_{i-1} \mathbf{M}_i^{-T})^{-1} \quad \text{and} \tag{21.20}$$

$$\mathbf{S}_i = (\mathbf{I} - \mathbf{K}_i \mathbf{M}_i) \mathbf{S}_{i-1}. \tag{21.21}$$

Given  $\mathbf{S}_0$  and  $\mathbf{a}_0$ , the Kalman filter recursively updates  $\mathbf{a}_i$ ,  $\mathbf{K}_i$ ,  $\mathbf{S}_i$  until the error covariance matrix  $\mathbf{S}_i$  becomes insignificantly small, or all the number of data points have been submitted. The  $\mathbf{a}_i$  thus obtained after termination of the algorithm is the estimated value of the parameters.

### 21.4.3 Predicting Target Position Using Extended Kalman Filter

Let the input measurement vector  $\mathbf{x}$  be given by

$$\mathbf{x} = \begin{pmatrix} r \\ \theta \\ t \end{pmatrix} \tag{21.22}$$

where

$r$  is the perpendicular distance of the target from the tracker at time  $t$ ,

$\theta$  is the angular displacement of the target measured with respect to the axis  $S_{12}$  of the tracker, i.e.  $\theta$  is the angular shift of tracker with respect to the target from time  $t=0$  to the current time of observation,

$t$  is the time elapsed measured from the beginning of the observation phase

Let the measurement equations in the present context be

$$\mathbf{f}_i = \begin{pmatrix} at^2 + bt + c - r\cos\theta \\ pt^2 + qt + s - r\sin\theta \end{pmatrix} = 0 \quad (21.23)$$

where

$c$  and  $s$  are the initial displacements of the target with respect to  $S_{12}$  axis and its perpendicular direction,

$b$  and  $q$  are the velocity in the corresponding directions, and

$a$  and  $p$  are the time rate of change of velocity in the corresponding directions.

For determination of the current position of the target we need to evaluate  $a$ ,  $b$ ,  $c$ ,  $p$ ,  $q$ ,  $s$  from the measured  $(r, \theta)$ s at time  $t=0$ ,  $t=t_1$  and  $t=t_2$  respectively. The estimated values of  $a$ ,  $b$ ,  $c$ ,  $p$ ,  $q$ ,  $s$  then may be substituted in the measurement equations to evaluate  $r \cos \theta$  and  $r \sin \theta$  at time  $t \geq t_2$ .

The estimator in the present context thus is given by

$$\mathbf{a} = \begin{bmatrix} a \\ b \\ c \\ p \\ q \\ s \end{bmatrix} \quad (21.24)$$

For the evaluation of the estimator we, however, need to determine the following derivatives:

$$\frac{\partial \mathbf{f}_i}{\partial \mathbf{x}} = \begin{pmatrix} -\cos\theta & r \sin\theta & 2at+b \\ -\sin\theta & -r \cos\theta & 2pt+q \end{pmatrix} \quad (21.25)$$

$$\frac{\partial f_i}{\partial a} = \begin{pmatrix} t^2 & t & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & t^2 & t & 1 \end{pmatrix}. \quad (21.26)$$

Let

$$\Lambda_i = \begin{pmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & \gamma \end{pmatrix} \quad (21.27)$$

where  $\alpha$  = the variance of the noise in measurement of range  $r$ ,  
 $\beta$  = the variance of the noise in measurement of angle  $\theta$ ,  
and  $\gamma$  = the variance of the noise in measurement of time  $t$ .

The algorithm for the evaluation of the current position  $(r, \theta)$  of the tracker at time  $t = t_3$  from its preceding positions is presented below.

**Procedure Evaluate ( $r_0, \theta_0, r_1, \theta_1, r_2, \theta_2$ )**

**Begin**

**1. Initialize**

- a)  $W_0 := (\partial f_0 / \partial x) \Lambda_i (\partial f_0 / \partial x)^T$  where  $\Lambda_0$  and  $(\partial f_0 / \partial x)$  are available in expressions (21.27) and (21.25) respectively;
- b)  $S_0$  as a diagonal matrix with large positive diagonal values;
- c)  $a_0$  to be zero;
- d)  $M_0 := \partial f_0 / \partial a$  vide expression (21.26);
- e) loop iteration index  $i := 1$ ;

**2. Repeat**

- a) input new measurement  $x_i$  and evaluate  $y_i$  by expression (21.13);
- b) update  $K_i, a_i, S_i$  in order using expressions (21.20), (21.19) and (21.21);

**Until**  $\text{abs}(a_i - a_{i-1}) < \text{a pre-defined threshold}$ ;

**3. Determine**

- a)  $at^2 + bt + c$  at time  $t = t_3 > t_2$ ;
- b)  $pt^2 + qt + r$  at time  $t = t_3 > t_2$   
for known  $a, b, c, p, q$  and  $r$ ;

**End.**

#### **21.4.4 Use of the Back-Propagation Neural Net**

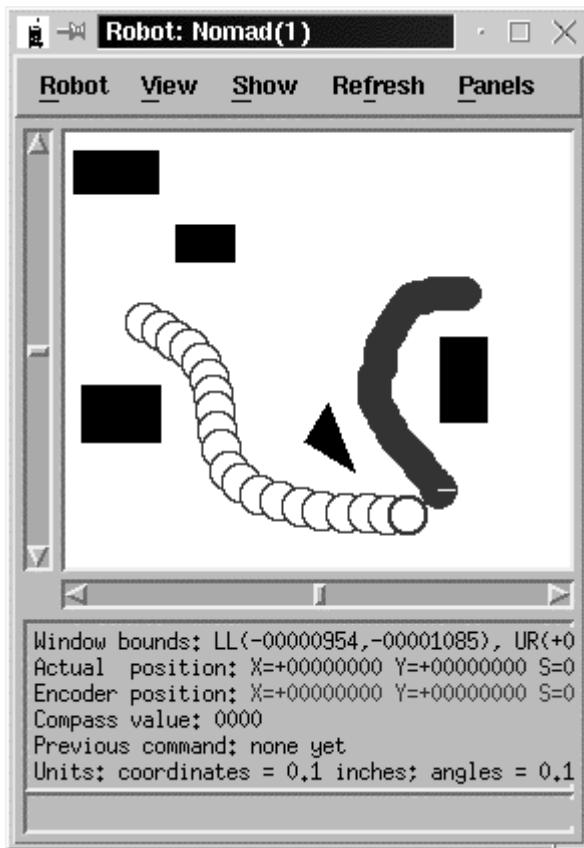
After the current target position and the obstacle locations in the tracker's workspace are determined, we use a pre-trained back-propagation neural net for controlling the step-wise motion of the tracker. A three layered feed-forward neural net has been employed in the present context for generating the control commands for motion of the tracker. The inputs of the neural net are the sonar readings obtained by the tracker robot at time  $t=t_3$  and the predicted target position at time  $t=t_3$ . In our realization, we considered the readings from 7 sonar transducers; consequently, the neural net has 8 inputs, the last one being the predicted position of the target.

Like section 21.2, the output of the neural net could be the amplitude and direction of motion of the tracker robot. But since speed of response is a prime consideration in the tracker design, we need to modulate the speed of the target depending on the estimated range of the tracker and obstacle locations. Thus speed and heading direction have been considered as the output fields of the proposed neural net. It may be added here that unlike section 21.2, where the heading direction was represented by 2 fields (amplitude and clockwise/counter-clockwise orientation), the heading direction in the present context has been represented by a single field to limit the output fields to 2 only.

The proposed neural net is trained using 600 training instances with a root mean square sum error at the output layered nodes below 0.003 units. Approximately 10,000 learning epochs are needed to train the neural net with the said error margin. In the application phase, the neural net just requires one forward pass, thus the speed of response of the neural net is very fast of the order of 0.5 milliseconds on a IBM 300 M-Hz Pentium machine.

#### **21.4.5 Experimental Results**

The experiments were carried out on Nomad simulator and also on 2 practical super scout II mobile robots. Figures 21.17 and 21.18 describe 2 simulated runs of the tracking program. Fig. 21.17 has been run by directly feeding the measured range to the input of the neural net. In Fig. 21.18 extended Kalman filter has been employed to predict the current position of the moving target. It is clear from these 2 figures that the tracker can generate more accurate control commands in presence of the extended Kalman filter.

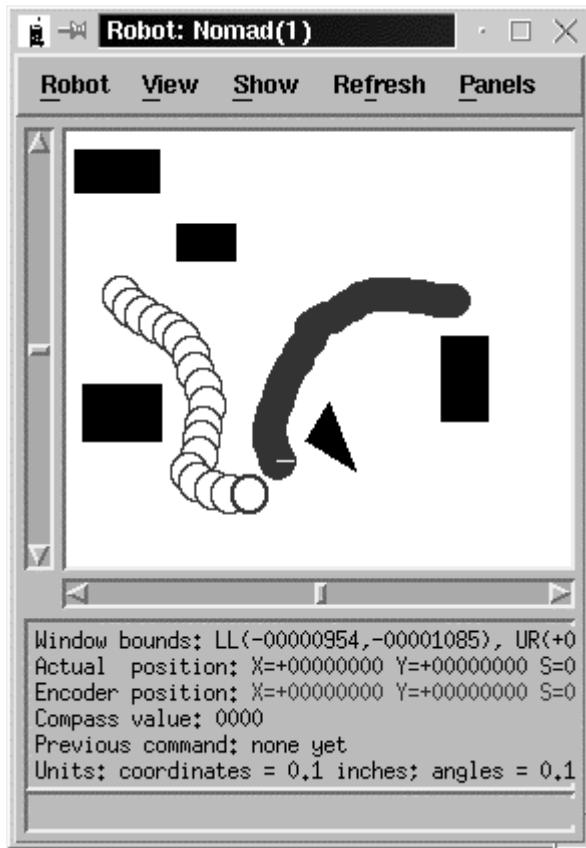


**Fig. 21.17:** Target tracking and interception without Kalman filter. The white window represents the tracker's world map. The rectangular dark objects denote the obstacles in the map. The path with circular traces denotes the trajectory of the target, and the solid path describes the trajectory of the tracker. The scheme was tested on a Nomadic platform

## 21.5 Conclusions and Future Directions

The chapter analyzed the scope of various computational tools of machine intelligence in realizing the cognitive skill [35] of mobile robots. Experimental results demonstrate that the neural network is a good choice for applications in path-planning problem of mobile robots. The chapter provided a common framework for comparing the behavioral performance of the different neural algorithms in path-planning applications. The benchmark analysis reveals that

the back-propagation algorithm is superior to RBF and SOFM-based algorithms for path-planning.



**Fig. 21.18:** Target tracking and interception with Kalman filtering. The tracker here intercepts the target much earlier than what it had done in fig. 21.17.

The chapter also examined the scope of fuzzy c-means clustering algorithm in segmentation of a moving object from the real-time video stream and its localization by the proposed 4-neighbor match algorithm. The experimental results on FCM clustering reveal that the logarithm of the gray scale pixel intensity is a good feature for clustering the dark pixels from relatively less dark ones. The localization algorithm is very efficient, as it needs minimum search to localize the object in the scene.

The chapter ends with a discussion on the utility of the above techniques in target tracking and interception of mobile robots. An extended Kalman filter has been employed here to predict the next position of the target robot from its few preceding positions. A pre-trained back-propagation neural net is used to generate the motion information of the tracker from the predicted target position and the obstacle map around the tracker. The speed of response of the tracker being very fast of the order of 400 milliseconds, and consequently the probability of missing the target is insignificantly small.

It needs mention here that the problems addressed in this chapter is not the only issues to be considered on mobile robots. In fact there are quite a number of issues which has drawn interest of the researchers working on mobile robots. The most challenging among these is the co-ordination of robots. Timed Petri nets [12], [48] have successively been used for handling the co-ordination problem in mobile robotics. The co-ordination problem can be co-operative [2] or destructive (predator-prey) type. The target-tracking problem we introduced in this chapter is a predator-prey type co-ordination. Box pushing or stick-carrying by 2 robots [1], where both the robots participate in path planning, fall within the co-operative type co-ordination. The most challenging part of co-operation is the data sharing by the robots through the radio network and thus forming a perception about their world. There is a vast literature available on co-operation [1] but few of these [19] employed the radio network for data sharing and perception building.

Another interesting problem on mobile robots that has not been addressed here for space limitation is the 3-D reconstruction [30] from multiple 2-D images of the robot's work space by using Kalman filtering. The detail of this is available in one of our recent work [44]. This too will find a new horizon for map building [23], [32], [41], [46], [48] in 3-D environment.

Most sophisticated robots are currently being designed for playing tournaments. Some interesting issues on soccer playing robots are readily available in [16] and [40].

## Exercise

1. Suppose, the measured value of range  $R_{\text{measured}}$  is quantized into 3 distinct categories:

$$R_{\text{quantized}} = 40 \text{ inches, if } 0 < R_{\text{measured}} \leq 40 \text{ inches,}$$

$$= 60 \text{ inches, if } 40 < R_{\text{measured}} \leq 60 \text{ inches,}$$

$$\text{and} \quad = 100 \text{ inches, if } R_{\text{measured}} > 60 \text{ inches.}$$

- a) Draw the transfer characteristics of quantized range versus measured range.
- b) Assume that a mobile robot possesses 8 transducers to measure range of obstacles around it. The goal direction and the current heading direction are also discretized into 4 categories. Determine the theoretical maximum number of patterns used for training the robot.

[**Hints:** b) Total no. of training instances =  $3^8 \times 2^4$ , where the factor  $3^8$  is due to range patterns by 8 transducers, and the factor  $2^4$  is due to discretized direction for the goal and the current-heading.]

2. In the target tracking problem presented in the text, given that

$$\mathbf{x} = [r \ \theta \ t]$$

where the definitions of  $r$ ,  $\theta$  and  $t$  remain same as used in the text.

Suppose, the measurement equations  $f$  is given by

$$\mathbf{f}_t = \begin{pmatrix} at^3 + bt^2 + ct + d - r \cos \theta \\ pt^3 + qt^2 + st + u - r \sin \theta \end{pmatrix}.$$

Let the estimator  $\mathbf{a} = [a \ b \ c \ d \ p \ q \ s \ u]^T$ .

- i) Evaluate  $\partial f_i / \partial \mathbf{x}$  and  $\partial f_i / \partial \mathbf{a}$ .
- ii) Compute  $W_i = (\partial f_i / \partial \mathbf{x}) \Lambda_i (\partial f_i / \partial \mathbf{x})^T$ , with  $\Lambda_i = 10 \mathbf{I}$ , where  $\mathbf{I}$  is the identity matrix.
- iii) Determine  $\mathbf{y}_i$ , using  $\mathbf{y}_i = \mathbf{M}_i \mathbf{a} + \mathbf{w}_i$  and  $\mathbf{w}_i = (\partial f_i / \partial \mathbf{x}) (\mathbf{x}_i - \mathbf{x}_i^*)$ , when  $\mathbf{x}_i^* = \mathbf{0}$ ,

[**Hints:**  $\partial f_i / \partial \mathbf{x}$

$$= \begin{pmatrix} -\cos \theta & r \sin \theta & 3at^2 + 2bt + c \\ -\sin \theta & -r \cos \theta & 3pt^2 + 2qt + s \end{pmatrix},$$

$$\frac{\partial f_i}{\partial \mathbf{a}}$$

$$= \begin{pmatrix} -\cos \theta & r \sin \theta & c \\ -\sin \theta & -r \cos \theta & s \end{pmatrix}.$$

The rest ia straight forward.]

3. With reference to the parameters given in problem 2, show one step of computation of  $\mathbf{K}_i$ ,  $\mathbf{a}_i$  and  $\mathbf{S}_i$  in order.

## References

- [1] Arkin, R. C., *Behavior-based robotics* , MIT Press: MA, pp.283, 1998.
- [2] Asada, M., Uchibe, E., and Hosoda, K., "Cooperative behavior acquisition for mobile robots in dynamically changing real worlds via vision-based reinforcement learning and development," *Artificial Intelligence*, vol. 110, pp. 2375-292, 1999.
- [3] Banerjee, R. K., *Navigational Planning of a Mobile Robot Using Neural Nets – a Comparative Study: Experiments with Nomad Super Scout II Robot*, M.E. Thesis, Robotics and Vision Lab., Jadavpur University, India. 2001.
- [4] Bender, E. A., *Mathematical Methods for Artificial Intelligence*, IEEE Computer Society Press, Los Alamitos, pp. 15-17, 1996.
- [5] Biswas, B., Konar, A. and Mukherjee, A. K., "Image matching with fuzzy logic," *Proc. of Int. Conf. on Control, Automation, Robotics and Computer Vision (ICARCV'94)*, Singapore, 1994.
- [6] Borenstain, J., Everett, H. R., and Feng, L., *Navigating Mobile Robots: Systems and Techniques*, A. K. Peters, 1996.
- [7] Brown, R. G. and Hwang, Patrick Y. C., *Introduction to Random Signals and Applied Kalman Filtering*, John Wiley & Sons, New York, 1997.
- [8] Chowdhury, A. S., *Range Estimation, Target Identification and Localization by a Mobile Robot for Tracking Applications*, M.E. Thesis, Robotics and Vision Lab., Jadavpur University, India. 2001.

- [9] Dudek, G. and Jenkin, M., *Computational Principles of Mobile Robots*, Cambridge University Press, pp. 151-171. 2000.
- [10] Evartee, R., *Sensors for Mobile Robots*, A. K. Peters Press, 1992.
- [11] Fok, Koon-Yu and Kabuka, M. R., "An automatic navigation system for vision guided vehicles using a double heuristic and a finite state machine," *IEEE Trans. on Robotics and Automation*, vol. 7, no.1, Feb. 1991.
- [12] Freedman, P., "Time, Petri nets, and robotics," *IEEE Trans. on Robotics and Automation*, vol.7, no. 4, pp.417-433, 1991.
- [13] Ghosh, B. K., Xi, N. and Tarn, T. J., *Control in Robotics and Automation: Sensor-Based Integration*, Academic press, chapter 13, pp. 381-418, 1999.
- [14] Goldberg, D. E., *Genetic Algorithm in Search Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.
- [15] Gonzalez, R. C. and Woods, R. E., *Digital Image Processing*, Addison-Wesley, 1997.
- [16] Hedberg, S., "Robots playing soccer? RoboCup poses a new set of AI research challenges," *IEEE Expert*, pp. 5-9, Sept/Oct, 1997.
- [17] Heikkonen, J. and Oja, E., "Self-organizing maps for visually guided collision-free navigation," *Proc. of the Int. Joint Conf. on Neural Networks*, Vol. 1, 669-692, 1993.
- [18] Hutchinson, S. A. and Kak, A. C., "Planning sensing strategies in a robot work cell with Multi-sensor capabilities," *IEEE Trans. on Robotics and Automation*, vol. 5, no.6, 1989.
- [19] Jain, L. C. and Fukuda, T. (Ed.), *Soft Computing for Intelligent Robotic Systems*, Studies in Fuzziness and Soft Computing Series, Physica-Verlag, Heidelberg, 1998.
- [20] Jang, J. S. R., Sun, C. T. and Mizutani,, E., *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*, pp. 424-425, Prentice-Hall, Englewood Cliffs, NJ, 1997.
- [21] Keramas, J. G., *Robot Technology Fundamentals*, Delmar Publishers, pp. 153-173. 2000.

- 
- [22] Klir, G. J. and Yuan, B., *Fuzzy Sets and Fuzzy Logic: Theory and Applications*, Prentice-Hall, Englewood Cliffs, N. J., pp. 358-361, 1995.
  - [23] Kluge, K. and Thorpe, C., "Explicit models for road following," *Proc. of IEEE Conf. on Robotics and Automation*, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1989.
  - [24] Kohonen, T., "The Self-Organizing Map," IEEE Proc., September, 1990.
  - [25] Kortenkamp, D., Bonasso, R. P. and Murphy, R., (Eds.), *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, AAAI Press/ The MIT Press, Cambridge, MA, 1998.
  - [26] Konar, A. and Pal, S., "Modeling Cognition with Fuzzy neural nets" In *Fuzzy Systems Theory: Techniques and Applications*, Leondes, C. T. (Ed.), Academic Press, NY, 1999.
  - [27] Konar, A., An introduction to computational intelligence paradigms, In *Engineering Applications of Computational Intelligence*, Jain, L. C. (Ed.), Kluwer Academic Publishers, Dordrecht, 2001.
  - [28] Konar, A., *Artificial Intelligence and Soft Computing: Behavioral and Cognitive Modeling of the Human Brain*, CRC Press: Boca Raton, Florida, 2000.
  - [29] Konar, A. and Jain, L. C., *Intelligent Robots: An Experimental Approach*, World Scientific, 2005 (to appear).
  - [30] Krebs, B., Korn, B., and Burkhardt, M., "A task driven 3D object recognition system using Bayesian networks," *Proc. of the Int. Conf. on Computer Vision*, pp.527-532, 1998.
  - [31] Krishnapuram, R., Segmentation, In *Handbook of Fuzzy Computation*, Ruspini, E. H., Bonissone, P. P. and Pedrycz, W. (Eds.), IOP Press, pp. F. 7. 4:1-F. 7.4:5, 1998.
  - [32] Ku, C. H., "Obstacle avoidance for autonomous land vehicle navigation in indoor environments by quadratic classifier," *IEEE Trans. on Systems, Man and Cybernetics*, vol. 29, no. 3, June 1999.
  - [33] *Language Reference Manual, Software Version: 2.6*, Nomadic Technologies, Inc.: CA. Part Number: DOC00002. 1997.
  - [34] Luo, F. and Unbehauen, R., *Applied Neural Networks for Signal Processing*, Cambridge University Press, pp. 22-26. 1997.

- [35] Matlin, W. Margaret, *Cognition*, Hault Sounders, printed and circulated by Prism books, India, 1996.
- [36] Meng, H. and Picton, P. D., "Neural network for local guidance of mobile robots," *Proc. of the Third Int. Conf. on Automation, Robotics and Computer Vision (ICARCV' 94)*, Nov.1994, Singapore.
- [37] Nebot, E. M., Sensors used for autonomous navigation In *Advances in Intelligent Autonomous Systems*, Tzafestas, S. G. (Ed.), Kluwer Academic Publishers, Dordrecht, pp. 135-156, 1999.
- [38] Nehmzow, U., *Mobile Robotics: A Practical Introduction*, Springer-Verlag, London, pp. 26-29, 2000.
- [39] Nicholas, A., *Artificial Vision for Mobile Robots*, The MIT Press, Cambridge, MA, 1991.
- [40] Nourbakhsh, I., Morse, S., Becker, C., Balabanovic, M., Gat, E., Simons, R., Goodridge, S., Potlapalli, H., Hinkle, D., Jung, K., and Vanvector, D., "The winning robot from the 1993 robot competition," *AI Magazine*, vol. 14 (4), pp. 51-62, 1993.
- [41] Pagac, D., Nebot, E. M. and Durrant. W., H., "An evidential approach to map building for autonomous robots," *IEEE Trans. on Robotics and Automation*, vol.14, no.2, pp. 623-629, Aug. 1998.
- [42] Pal, S. K. and Mitra, S., *Neuro-Fuzzy Soft Computing*, John Wiley & Sons, Ch. 1, 1999.
- [43] Patnaik, S. *Building Cognition for Mobile Robots*, Doctoral thesis, Robotics and Vision Lab., Jadavpur University, India. 2000.
- [44] Patnaik, S., Konar, A. and Mandal, A. K., "Building 3-D visual perception of a mobile robot employing extended Kalman filter," *J. of Intelligent and Robotic Systems*, vol. 34, pp. 99-120, 2002
- [45] Patnaik, S., Konar, A., and Mandal, A. K., "Constrained hierarchical path planning of a robot by employing neural nets," *Proc. of the Fourth Int. Symposium on Artificial Life and Robotics (AROB 4<sup>th</sup> '99)*, Japan, Jan. 1999.
- [46] Patnaik, S., Konar, A., and Mandal, A. K., "Map building and navigation by a robotic manipulator," *Proc. of Int. Conf. on Information*

- Technology, pp.227-232, TATA-McGraw-Hill, Bhubaneswar, Dec. 1998.
- [47] Patnaik, S., Konar, A., and Mandal, A. K., "Navigational planning with dynamic scenes," *Proc. of the Int. Conf. on Computer Devices for Communication (CODEC-98)*, pp. 40-43 , held at Calcutta, Jan., 1998.
- [48] Patnaik, S., Konar, A., and Mandal, A. K., "Visual perception for navigational planning and coordination of mobile robots," *Indian Journal of Engineers*, vol. 26, Annual Number '97, pp.21-37, 1998.
- [49] Raychowdhury, A. , Chattopadhyay, B., Chowdhury, A. S. and Konar, A., Range estimation using multi-sensory data fusion, *Autonomous Robots*, Kluwer Academic Publishers (communicated).
- [50] Rumelhart, D. E., Hilton, G. E., and Williams R. J., "Learning internal representations by error propagation," In *Parallel Distributed Processing*, Rumelhart, D. E. and McClelland, J. L. (Eds.), vol. 1, The MIT Press, Cambridge, MA, pp. 318-362, 1986.
- [51] Thrope, C. E. (Ed.), *Vision and Navigation: The Carnegie Mellon Navlab*, Kluwer Academic Publishers, pp. 84-90, 1990.
- [52] *User's Manual*, Version: 2.6, Nomadic Technologies, Inc.: CA. Part Number: DOC00004. 1997.
- [53] Xiao, J., "The evolutionary planner / navigator in a mobile robot environment," in *Handbook of Evolutionary Computing*, Ed., Back, J., Fogel, D. B. and Michalewicz, Z., IOP and Oxford University Press, New York, 1997.
- [54] Xiao, J., Michalewicz, Z., Zhang, L. and Trojanowski, K., "Adaptive evolutionary planner/ navigator for mobile robots," *IEEE Trans. on Evolutionary Computation* , vol.1, no.1, April 1997.
- [55] Zavlangas, P. G. and Tzafestas, S. G., "Industrial robot navigation and obstacle avoidance employing fuzzy logic," *J. of Intelligent Robotic Systems*, Kluwer Academic Publishers, vol. 27, pp. 85-97, 2

# 22

# Emerging Areas of Computational Intelligence

*The chapter provides an introduction to the new members of the computational intelligence family that are currently gaining importance for their increasing applications in both science and engineering. The list of the new members includes Artificial Life, Particle Swarms, Artificial Immune Systems, Chaos Theory, Rough Set Theory and Granular Computing. The biological concepts involved in the first three topics are briefly explained to enable the readers to construct their mathematical models for specific engineering applications. The chaos theory is introduced to demonstrate the behavior of fuzzy dynamical systems. The chaotic behavior of fuzzy dynamics is illustrated with typical models of fuzzy liars. Rough sets and granular computing are presented in a nutshell to familiarize the readers with these growing disciplines of knowledge.*

## 22.1 Introduction

Pioneered by Langton, Artificial Life [12] embraces human-made systems that possess some of the fundamental properties of natural life [18]. While studying

artificial life, we are specifically interested in artificial systems that serve as models of living systems for the investigation of open questions in biology. The chapter begins with a brief introduction to artificial life with special reference to a problem of artificial fish schools.

Kennedy and Eberhart [10] proposed a new concept of particle swarms as an optimization method that stimulates the social behavior of animals. *Particle Swarm Optimizer* (PSO) is a population based stochastic algorithm that exploits a population of individuals to synchronously probe promising regions in the search space. The points in the search space are called particles. Each particle moves with an adaptable velocity within the search space, and retains a memory of the best position it encountered so far. In the global variant of PSO, the best position ever attained by all individuals of the swarm is communicated to all the particles at each iteration. In the local variant of PSO, each particle is assigned to a topological neighborhood consisting of a pre-specified number of particles. In this case, the best position ever attained by the particles that comprise a neighborhood is communicated among [4], [10].

*Artificial Immune Systems* (AIS) are biologically inspired models for immunization of engineering systems. The pioneering task of AIS is to detect and eliminate non-self materials, called antigens such as virus or cancer cells. The AIS also plays a great role to maintain its own system against dynamically changing environment. The immune systems thus aim at providing a new methodology suitable for dynamic problems dealing with unknown/ hostile environments.

Dynamics of complex non-linear systems may sometimes give rise to chaotic phenomena. Currently, fuzzy logic and chaos theory form two of the most intriguing and promising areas of mathematical research. In this chapter, we want to explore a region of fuzzy logic, which exhibits some of the phenomena of chaos theory. It is indeed important to note that there could be other type of fuzzy non-linear dynamics that also exhibits chaotic behavior.

We have discussed a lot on uncertainty management using fuzzy sets. In this chapter, we shall briefly introduce another interesting method for uncertainty management by rough sets. Introduced by Pawlak [14] in the 1980's, rough set theory constitutes a sound basis for discovering patterns in hidden data and thus have extensive applications in data mining in distributed systems. In this chapter, we, however, discuss some applications of rough sets in granular computing.

Granular computing is another emerging conceptual and computational paradigms of information processing. It has been motivated by the urgent need for intelligent processing of empirical data, which is now available in vast quantities on distributed databases into a humanly manageable abstract knowledge. The theoretical foundations of granular computing involve set

theory (interval analysis), fuzzy sets, rough sets and random sets. The chapter briefly outlines the fundamental principles of granular computing.

In section 22.2 we discuss artificial life with a case study to artificial fish schools and their dynamic behavior for varying body size and body forms. Section 22.3 introduces the concepts of particle swarms. In section 22.4, we present the principles of immune systems with applications to a garbage collection problem by a mobile robot. The phenomena of chaos in fuzzy reasoning will be discussed in section 22.5. A brief introduction to rough sets will be given in section 22.6. Granular computing will be addressed in section 22.7. In section 22.8 we redefine computational intelligence using emerging technologies. Concluding remarks are summarized in section 22.9.

## 22.2 Artificial Life

Natural life on earth is organized into at least four fundamental levels of structure: the molecular level, the cellular level, the organism level and the population-ecosystem level. A living creature at any of these levels is a complex adaptive system exhibiting behavior that emerges from the interaction of a large number of elements from the levels below.

Dealing with this multilevel complexity requires a broad methodological shift in the biological sciences today as a new collection of Artificial Life models of natural biological systems. In this chapter, we provide a model for artificial fish schools, and discuss the effects of school size, body size and body form on their behavior.

### 22.2.1 The Model of Artificial Fish Positions, Speed and Heading

Let

$\mathbf{x}_i(t)$  be the position of the  $i$ -th artificial fish agent at time  $t$ ,

$\mathbf{v}_i(t)$  be the velocity of the  $i$ -th agent at time  $t$ ,

$\Delta t$  be the increment in time.

Then we can write:

$$\mathbf{x}_i(t) = \mathbf{x}_i(t - \Delta t) + \mathbf{v}_i(t) \cdot \Delta t \quad (22.1)$$

Let  $\alpha_i(t)$  be the heading direction of the  $i$ -th agent with respect to its body-line. Then we can express  $\mathbf{v}_i(t)$  by

$$\mathbf{v}_i(t) = \begin{pmatrix} v_i(t) \cos \alpha_i(t) \\ v_i(t) \sin \alpha_i(t) \end{pmatrix} \quad (22.2)$$

In certain models of fish schools, the speed  $v_i(t)$  of an agent does not depend on that of other agents, but is an independent stochastic variable. It is drawn from a Gaussian probability distribution  $P(v_{avg}, v_{sd})$  each time step. The stochasticity is used to represent unspecified variation in speed.

The heading direction  $\alpha_i(t)$  of the  $i$ -th agent at time  $t$  is given by

$$= P(\alpha_i(t - \Delta t) + \omega_i(t) \cdot \Delta t, \alpha_{sd}) \quad (22.3)$$

where  $\alpha_i(t - \Delta t)$  is the agent's heading direction in the last simulation step, and  $\omega_i(t)$  is its rate of rotation, which depends on the interaction with other agents.  $\alpha_i(t)$  is drawn from a Gaussian distribution  $P((\alpha_i(t))_{avg}, \alpha_{sd})$ .

### 22.2.2 Repulsion, Attraction and Aligning

The artificial fish have three types of behavioral responses, namely repulsion (short distances), attraction (intermediate distance) and alignment (greater distance) [1, 19]. This has often been modeled by splitting the region surrounding the agent into behavioral zones with definite boundaries; therefore an agent triggers exactly one type of behavioral response in a given neighborhood. In the present model, we, however, consider the possibility of triggering all the three behaviors with varying effectiveness determined by their respective weights.

Let

$\omega_{\text{def}}$  = default rate of rotation of the agents,

$\omega_r$ = rate of rotation of an agent due to repulsion by its neighbors,

$\omega_a$ = rate of rotation of an agent due to attraction by its neighbors.

$\omega_p$ = rate of rotation of an agent to align itself with others in a given neighborhood,

$\theta_{ij}(t)$  = angle between  $\mathbf{v}_i(t)$  and  $(\mathbf{x}_j(t) - \mathbf{x}_i(t))$ .

In the present context,

$$\left. \begin{array}{l} \omega_i = -\omega_{\text{def}} \text{ if } \theta_{ij} > 0 \text{ (avoid agent to the left)} \\ \quad +\omega_{\text{def}} \text{ otherwise (avoid agent to the right)} \end{array} \right\} \quad (22.4)$$

$$\omega_a = \omega_{def} \cdot \theta_{ij}(t) \quad (22.5)$$

and  $\omega_p = \omega_{def} \varphi_{ij}(t)$

$$= \angle(\mathbf{v}_i(t), \mathbf{v}_j(t)) \quad (22.6)$$

where  $\angle(\mathbf{v}_i(t), \mathbf{v}_j(t))$  denotes the angle between the orientations of the two agents.

The combined behavior of agent i in the neighborhood of agent j is a weighted sum given by

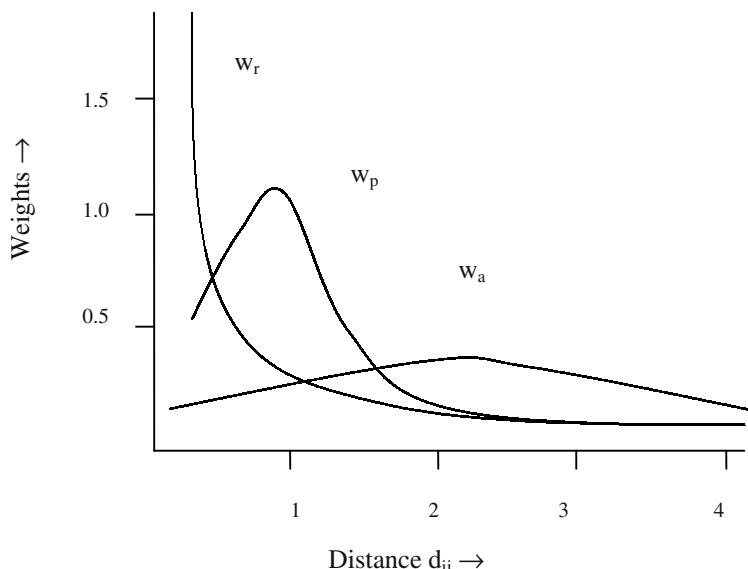
$$\omega_{ij}(t) = w_r(d_{ij}(t)) \omega_r + w_a(d_{ij}(t)) \omega_a + w_p(d_{ij}(t)) \omega_p \quad (22.7)$$

where the weights  $w_r, w_a, w_p$  are functions of  $d_{ij}(t)$ . The details of the functions are available in [11].

When an agent i faces a number of agents j for  $j=1$  to  $N$ ,  $\omega_i(t)$  is determined by averaging the agent i's response in presence of individual agent j. Thus

$$\omega_i(t) = \frac{1}{N} \sum_{j=1}^N \omega_{ij}(t) \quad (22.8)$$

The variation of weights  $w_r, w_a$  and  $w_p$  over distance  $d_{ij}$  between agent i and agent j are plotted below vide [11].

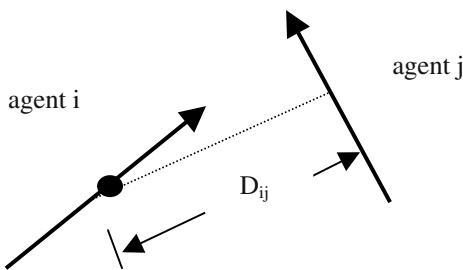


**Fig. 221:** Effect of variation of  $d_{ij}$  on weights  $w_r, w_p$  and  $w_a$ .

The plots given in Fig. 22.1 are obtained through modeling of  $w_r$ ,  $w_p$  and  $w_a$ , the details of which are available in [11]. It is clear from Fig. 22.1 that the weight  $w_r$  decreases with increasing  $d_{ij}$ . The result is intuitive as with increasing  $d_{ij}$ , the repulsive effect of  $j$  on  $i$  usually has a steep fall off. The  $w_p$  has a peak at  $d_{ij}$  close to 1. This intuitively carries some meaning. As distance  $d_{ij}$  increases from zero, the alignment of  $i$  with  $j$  should increase, but for a larger gap  $d_{ij}$ , the alignment of  $i$  with  $j$  simply is meaningless. Lastly, the phenomena of weight  $w_a$  having a peak more or less at the center of the  $d_{ij}$ -spectrum is self-explanatory. It indicates that for very small gap or very large gap, the attraction of agent  $i$  by agent  $j$  does not carry any meaning; but for a moderate gap of (2- 3) units the attraction of agent  $i$  by agent  $j$  logically makes sense.

### 22.2.3 Body Size and Form

So far we considered only point mass agents. But in practice agents have a definite shape and size. For simplicity, we can consider the fish agents to have a straight-line body. The assumption does not change the agent's behavior directly, but it alters the way the agent is perceived by others. The distance between two point mass agents can be computed by measuring their Euclidean distance. But when the agents have a linear body-shape, the distance between agent  $i$  and agent  $j$  can be defined by measuring the gap between agent  $i$ 's center and the nearest point on agent  $j$ . We call it  $D_{ij}(t)$ .



**Fig. 22.2:** Defining  $D_{ij}(t)$  as the distance between the center of agent  $i$  to the nearest point on the body of agent  $j$ .

The body size of agents thus has an effect on the repulsion, attraction and alignment of other agents. The body form (shape) is modeled by including sensory characteristics of the lateral line, thereby changing the agent's perception of its neighbors.

The behavior of school of fish can be simulated using the above model. The necessary observations are i) location of the center of mass of each school, ii) the adaptation of orientation in absence of a group leader, and iii) the effect of body size and shape on the geometry of the overall school pattern.

## 22.3 Particle Swarm Optimization

Particle swarms, as first described by Eberhart and Kennedy [3] are best classified under the heading of evolutionary computing. The basic operational principle of the particle swarm is reminiscent of the behavior of flock of birds or the sociological behavior of a group of people. In this chapter we outline the scope of particle swarms as global optimizers.

A swarm consists of several particles, each particle keeps track of its own attributes. The most important attribute of the particles is their current positions, represented by n-dimensional vectors. The position of the particles corresponds to potential solutions of the function to be minimized. Along with the position, the particle has a current velocity, keeping track of the speed and direction of travel by the particle. Analogous to population members in GA, each particle also has a current fitness value, which is obtained by evaluating the error function of the particle's current position.

Additionally, each particle also remembers its own personal best position, so that potentially good solutions can be used to guide the construction of new solutions. At swarm level, the best overall positions among all particles is also recorded. This position upon termination of the algorithm may serve as the answer.

During each epoch, every particle is accelerated towards its individual best as well as the global best position. This is achieved by evaluating a new velocity term for each particle based on the estimate of a distance from its individual best and global best position. These two components (individual best and global best) of velocity are then randomly weighted to produce the velocity value for the particle. This new velocity affects the position of the particle in the next epoch.

Let us now mathematically formulate the above issue. Consider a D-dimensional search space  $S \subset R^D$  and a swarm consisting of N particles. The i-th particle is in effect a D-dimensional vector:

$$\mathbf{X}_i = (x_{i1}, x_{i2}, \dots, x_{iD})^T \in S \quad (22.9)$$

The velocity of this particle is also a D-dimensional vector,

$$\mathbf{V}_i = (v_{i1}, v_{i2}, \dots, v_{iD})^T \in S \quad (22.10)$$

The best previous position encountered by the i-th particle in S is denoted by

$$\mathbf{Y}_i = (y_{i1}, y_{i2}, \dots, y_{iD})^T \in S \quad (22.11).$$

Let  $\mathbf{Y}^*$  be the global best position amongst all the particles, and  $t$  be the program iterations. During each iteration, the velocity and position of each particle/swarm is updated following equations (22.12) and (22.13) below. Further, assume that we want to minimize a function  $\mathbf{Y}_i = f(\mathbf{X}_i)$  using m particle swarms.

$$\mathbf{V}_i(t+1) = \mathbf{W} \mathbf{V}_i(t) + c_1 r_1 (\mathbf{Y}_i(t) - \mathbf{X}_i(t)) + c_2 r_2 (\mathbf{Y}^* - \mathbf{X}_i(t)) \quad (22.12)$$

$$\text{and } \mathbf{X}_i(t+1) = \mathbf{X}_i(t) + \mathbf{V}_i(t+1) \quad (22.13)$$

where

$$\left. \begin{array}{l} \mathbf{Y}_i = \mathbf{Y}_i, \text{ if } f(\mathbf{X}_i) \geq f(\mathbf{Y}_i) \\ \mathbf{Y}_i = \mathbf{X}_i, \text{ if } f(\mathbf{X}_i) < f(\mathbf{Y}_i) \end{array} \right\} \quad (22.14)$$

$\mathbf{Y}^* \in \{\mathbf{Y}_0, \mathbf{Y}_1, \dots, \mathbf{Y}_n\}$  such that

$$f(\mathbf{Y}^*) = \min(f(\mathbf{Y}_0), f(\mathbf{Y}_1), \dots, f(\mathbf{Y}_n)) \quad (22.15)$$

$r_1 \sim U(0, 1)$  and  $r_2 \sim U(0, 1)$  are elements from two uniform random sequences in the range  $(0, 1)$ .

$\mathbf{W}$  is a diagonal matrix, representing weights of  $\mathbf{V}_i(t)$  as a contribution to  $\mathbf{V}_i(t+1)$ .

It is clear from expression (22.15) that  $\mathbf{Y}^*$  is the global best position amongst all the particles. The value of each component of  $\mathbf{V}_i$  vector is clamped to the range  $[-v_{\max}, v_{\max}]$  to prevent the PSO from leaving the search space. Usually,  $v_{\max}$  is chosen to be  $k \times x_{\max}$  with  $0.1 \leq k \leq 1.0$ . It is to be noted that above selection of  $v_{\max}$  does not restrict the component of  $\mathbf{X}_i$  in  $[-v_{\max}, v_{\max}]$ ; it only limits the maximum distance that a particle moves in one direction.

The matrix  $\mathbf{W}$  in (22.12) is called the inertia weight matrix. The diagonal elements of this matrix are set up to vary linearly from one to near zero during a training run.

Acceleration co-efficients  $c_1$  and  $c_2$  also control the displacements of a particle in a single iteration. Typically, these are set to a value 2 [4], although assigning different values to  $c_1$  and  $c_2$  sometimes leads to improved performance [4].

The pseudo-code for the simplest PSO algorithm is given below:

**Procedure Plain Swarm Optimization****Begin**

Create and initialize D-dimensional PSO: S

**Repeat**    **For**  $j \in [1..N]$         **If**  $f(S, X_j) < f(S, Y_j)$             **Then**  $S, Y_j = S, X_j;$         **If**  $f(S, Y_j) < f(S, Y^*)$             **Then**  $S, Y^* = S, Y_j;$     **End For;**

Perform updates on S using equations (22.12) and (22.13);

**Until** stopping criterion is reached;**End.**

The PSO algorithm presented above is self-explanatory. We considered N particles and for each particle we attempted to identify optimal  $Y_j$  both individually and globally. The updates on position and velocity of each particle is continued until the terminating condition such as steady-state value of  $X_i = X_i^*$  is reached, following which any further motion of particle ceases.

### 22.3.1 Particle Swarm Size

The swarm size is a critical parameter in the PSO algorithm. Very few particles will cause the algorithm to become stuck at local minima. On the other hand, too many particles will slow down the algorithm. Assume that a simulation will run over I iterations, using an n-dimensional function. The number of Error Function Evaluations (EFEs) will be equal to the product of the swarm size, P, and the number of iterations; thus  $E = I \times P$ . This equation holds for the classical PSO algorithms.

It is important to note that EFEs remain constant irrespective of its implementation. The number of EFEs in the split swarm =  $I' \times P \times D$ , where  $I'$  and D denote iterations and number of partitions respectively. Consequently,  $E = I \times P = I' \times P \times D$ . It is also clear from the formulation of EFEs that for a given E, either iterations or swarm size can be large. A large number of iterations will eventually improve the quality of the solutions as long as the whole swarm does not get stuck in a local minimum.

### 22.3.2 The Split Swarm Algorithm

The split swarm algorithm takes the D-dimensional solution vector and breaks it into D one-dimensional components. The optimization of each component is then performed by a separate PSO. The error function is evaluated using a vector formed by concatenating the components from the D-swarms to again construct a D-dimensional vector.

**Procedure Split Swarm****Begin****Define**

$$b(j, z) = (S_1.Y^*, \dots, S_{j-1}.Y^*, Z, S_{j+1}.Y^*, \dots, S_D.Y^*)$$

**Initialize D one-dimensional PSOs:**

$$S_i, i \in [1..D].$$

**Repeat****For**  $i \in [1..D]$ **For**  $j \in [1..N]$ 

$$\text{If } f(b(i, S_i.X_j)) < f(b(i, S_i.Y_j))$$

**Then**  $S_i.Y_j = S_i.X_j;$ 

$$\text{If } f(b(i, S_i.Y_j)) < f(b(i, S_i.Y^*))$$

**Then**  $S_i.Y^* = S_i.Y_j;$ **End For;**Perform updates on  $S_i$  using (22.12) and (22.13)**End For;****Until** stopping criterion is met;**End.**

## 22.4 Artificial Immune Systems

In recent years much attention has been focused on behavior-based AI for its proven robustness and flexibility in a dynamically changing environment. *Artificial Immune Systems* (AMS) is one such behavior-based reactive system that aims at developing a decentralized consensus-making mechanism, following the behavioral characteristics of biological immune system.

The basic components of the biological immune system are macrophages, antibodies and lymphocytes, the last one being classified into two types: B-lymphocytes and T-lymphocytes are the cells stemming from the bone marrow. The human blood circulatory system contains roughly  $10^7$  distinct types of B-lymphocytes, each of which has a distinct molecular structure and produces Y-shaped antibodies from its surface. Antibodies can recognize foreign substances, called antigens that invade living creature. Virus, cancer cells etc. are typical examples of antigens. To cope with continuously changing environment, living system possess enormous repertoire of antibodies in advance.

T-lymphocytes, on the other hand, are the cells maturing in thymus, and they are used to kill infected cells and regulate the generation of antibodies from B-lymphocytes as outside circuits of B-lymphocyte networks.

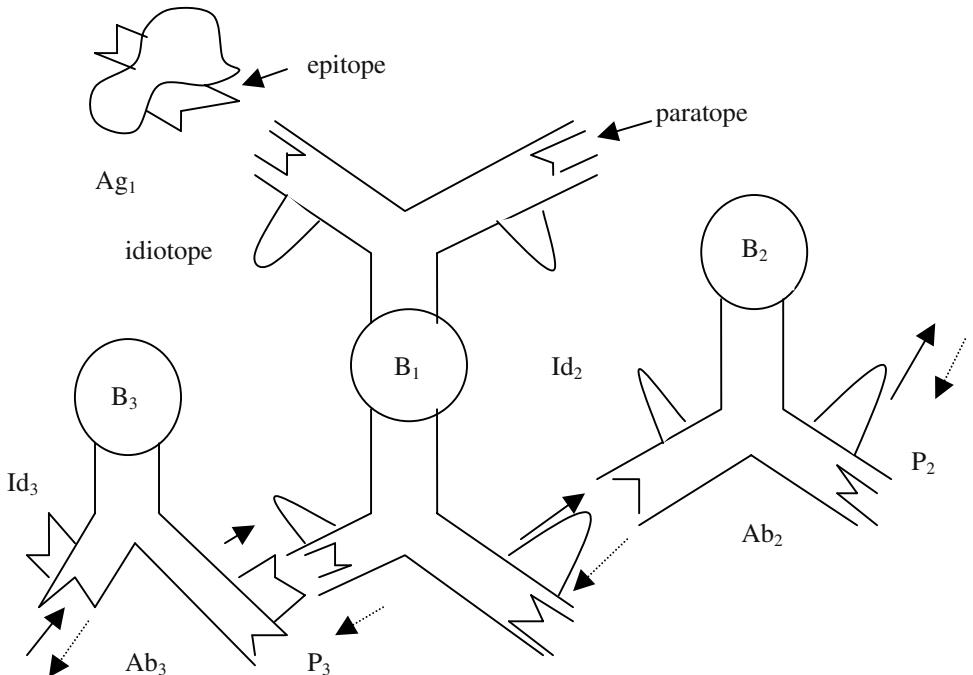
It is interesting to note that an antibody recognizes an antigen by part of its structure called *epitope*. The portion of the antibody that has the recognizing capability of an antigen is called *paratope*. Usually, epitope is the key portion of the antigen, and paratope is the keyhole portion of an antibody. Recent study in immunology reveals that each type of antibody has its specific antigen-determinant, called *idiotope*.

### 22.4.1 Biological Communication among Different Species of Antibodies

Jerne [7-9] proposed a remarkable hypothesis called the *idiotypic network hypothesis*. According to the hypothesis, antibodies/ lymphocytes are not isolated, but they communicate to each other among their variant species. Fig. 22.3 provides a schematic view of biological communication among different species of antibodies.

The idiotope  $Id_1$  of antibody  $Ab_1$  stimulates the B-lymphocyte  $B_2$ , which subsequently attaches antibody  $Ab_2$  to the surface of  $B_2$  through paratope  $P_2$ . However, from the standpoint of  $Ab_2$ , the idiotope  $Id_1$  of  $Ab_1$  acts as an antigen, and consequently  $Ab_2$  suppresses the B-lymphocyte  $B_1$  from  $Ab_1$ .

Similarly, the antibody  $Ab_3$  stimulates  $Ab_1$  and the idiotope  $Id_3$  of  $Ab_3$  acts as an antigen to  $B_3$  in view of  $Ab_1$ . Stimulation and suppression of antibodies thus continue in a chain form in idiotypic networks.



$B_1$ ,  $B_2$  and  $B_3$  are B-lymphocytes;  $Id_1$ ,  $Id_2$  and  $Id_3$  are idiotypes;  $Ab_1$  and  $Ab_2$  are antibodies;  $Ag_1$  is an antigen;  $P_1$ ,  $P_2$  and  $P_3$  are paratopes. Solid and dotted arrows denote stimulation and suppression respectively.

**Fig. 22.3:** Jerne's idiotypic network hypothesis.

### 22.4.2 A Simple Model of the Immune System

Let

- $a_i(t)$  be the concentration of the  $i$ -th antibody,
- $m_{ji}$  be the affinity between antibody  $j$  and antibody  $i$ ,
- $m_{ik}$  be the affinity between antibody  $i$  and the detected antigen  $k$ ,
- $k_i$  be the natural decay rate of antibody  $i$ ,
- $N$  and  $M$  respectively denote the number of antibodies that stimulate and suppress antibody  $i$ .

The growth rate of antibody  $i$  is given below:

$$\frac{da_i}{dt} = \frac{N}{\sum_{j=1}^N m_{ji} a_j(t)} - \frac{N}{\sum_{k=1}^M m_{ik} a_k(t)} - m_i - k_i a_i(t) \quad (22.16)$$

$$\text{and } a_i(t+1) = \frac{1}{1 + \exp(0.5 - a_i(t))}. \quad (22.17)$$

The first and the second term in the right hand side of (22.16) respectively denote the stimulation and suppression by other antibodies respectively. The third term denotes the stimulation from the antigen, and the fourth term represents the natural decay of the  $i$ -th antibody. Equation (22.17) is a squashing function used to ensure the stability of the concentration.

In this chapter, selection of antibodies is simply carried out by Roulette wheel bass according to the magnitude of concentration of the antibodies. It is to be noted that only one antibody is allowed to activate and act its corresponding behavior in its world.

### 22.4.3 Application in Garbage Collection by Mobile Robots

Consider the problem of garbage collection by a mobile robot [6]. The task of the robot is to collect the garbage into a garbage-can located inside the robot's world, without running out of its internal energy. It is important to note that the immunoid consumes certain energy as it moves around the environment.

In the garbage collection problem, the detected current internal/ external situation such as existence of garbage in a prescribed direction, direction of home-base and current energy level of the batteries act as antigens. On the other hand, the prepared simple behaviors of the robot, such as move forward, turn left/ right, explore, catch garbage and searches for home-base are modeled as

antibody. The structure of a typical antibody to be used in the garbage collection problem is presented in Fig. 22.4.

The field definitions of the antibody, given in Fig. 22.4, are self-explanatory. The third field in the figure corresponds to two parameters: i) identifier of the stimulating antibody, and ii) degree of stimulus of the antibodies. The degree of stimuli of an antibody is obtained by solving (22.16) and (22.17) in sequence.

Paratope	Idiotope
Pre-condition	Behavior
Pre-condition	ID# of stimulating antibodies and their degree of stimuli
G: Garbage (F: Front, R: Right, L: Left N: None, H: in Hand)	F: Move forward L: Turn left R: Turn right
W: Wall (F: Front, R: Right, L: Left N: None)	Sh: Search for home-base C: Catch garbage E: Explore
Hb: Home-base (F: Front, R: Right, L: Left N: None, Nr: Near, M: Middle Fr: Far)	
Eg: Internal energy level (L: Low, H: High)	

**Fig. 22.4:** Structure and field definitions of an antibody.

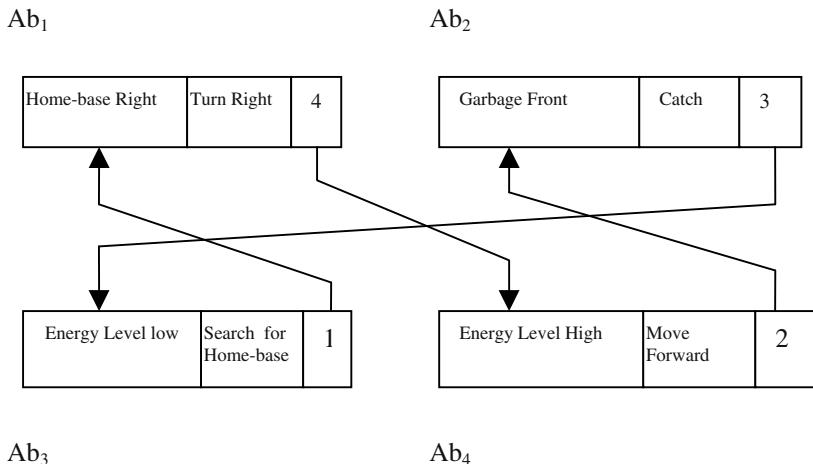
**Example 22.1:** Suppose the robot detects its home-base in its right side, the energy level of the robot is high, and the garbage is in its front. These three information act as 3 antigens to invade the immunoid's interior.

To handle the situation, 4 antibodies are prepared in advance. Antibody 1, for example, means that if immunoid detects home-base in the right direction, this antibody can be activated and would cause *turn right* behavior. Similarly, if the current energy level is high, this antibody would give way to other antibodies represented by its idiotype (here antibody 4) to prevent charging.

Suppose the immunoid has enough energy. In this case, antibodies 1, 2 and 4 are simultaneously activated by the antigens. As a result, the concentration of the antibody will increase. But because of interactions among the antibodies

(Fig. 22.5), the concentration of each antibody varies, and the one with the highest concentration is selected.

In the present context, the immunoid will catch the garbage (as the energy level is high). If the energy level was low, the immunoid would have preferred to re-charge the battery.



**Fig. 22.5:** An example of consensus-making through interaction among antibodies.

## 22.5 Fuzzy Chaos Theory

Chaos theory is a promising area in non-linear dynamical systems. Recently, fuzzy logic and chaos theory have merged to form a new discipline of knowledge, called fuzzy chaos theory [2]. To understand the implications of fuzzy chaos theory, let us consider the fuzzy liar models [5].

Let  $\mu_n(x)$  be the membership of a statement  $x$  to be true at the  $n$ -th iteration. Then we can model a fuzzy liar by

$$\mu_{n+1}(x) = 1 - \mu_n(x) \quad (22.18)$$

What does equation (22.18) implicate? It means that the fuzzy liar always states false statement. So, if he says that the given sentence  $x$  is false, we consider it to be true in the contrary.

Starting with a initial/ seed value of  $\mu_0(x) = 0.3$ , we can iterate  $\mu_{n+1} = 1 - \mu_n$ , and obtain

$$\mu_1 = 0.7, \mu_2 = 0.3, \mu_3 = 0.7 \text{ and so on.}$$

In other words,  $\mu_{n+1}$  oscillates with  $n$ , and such periodic oscillation (limit cycles) sustains for ever. If we draw a phase trajectory of  $\mu_{n+1}$  versus  $\mu_n$ , we can observe a closed curve. Such closed curve is an indication of limit cyclic behavior of the dynamics.

Now, consider a second example. Suppose the fuzzy liar states that the statement is very true. We can re-write it as follows:

The given statement is false. (1)

(1) is very true. (2)

In the language of fuzzy sets, we have:

$$\mu_{n+1} = (1 - \mu_n)^2 \quad (22.19)$$

To check whether (22.19) has any fixed point, we set  $\mu_{n+1} = \mu_n = \mu^*$  in the said equation and thus obtain:

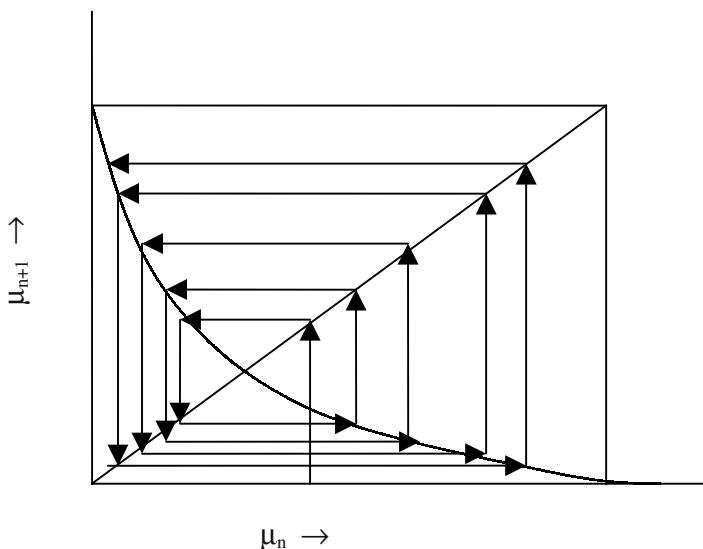
$$\mu_n^* = 1 - 2\mu_n^* + (\mu_n^*)^2 \quad (22.20)$$

$$\text{or, } \mu_n^* = \frac{3 \pm \sqrt{5}}{2}. \quad (22.21)$$

Since  $0 \leq \mu_n \leq 1$ , only the point  $(3 - \sqrt{5})/2$  lies in the given interval. So, there is a single fixed point  $\mu_n = (3 - \sqrt{5})/2$ .

The phase trajectory of  $\mu_{n+1}$  versus  $\mu_n$  is presented in Fig. 22.6 below. We obtain this plot by joining the coordinates of the points:  $(\mu_0, 0)$ ,  $(\mu_0, 1 - \mu_0^2)$ ,  $(1 - \mu_0^2, 1 - \mu_0^2)$ ,  $(1 - \mu_0^2, 1 - (1 - \mu_0^2)^2)$ , ..., by straight lines. For a real seed value of  $\mu_0 = 0.3$ , the emphatic liar forces a series of revised values which eventually converge on the familiar oscillation between 0 and 1 characteristic of classical liar. It is indeed important to note that at  $\mu_n = (3 - \sqrt{5})/2$ , the dynamics of the emphatic liar has a fixed (stable) point. So, at this point  $\mu_{n+1} = \mu_n$ , and consequently this point on the phase trajectory acts as a fixed repellent point.

The behavior of a dynamical system can be of any of the 4 types: i) stable, ii) unstable, iii) oscillatory or limit cycles and iv) chaotic. The examples we have taken so far are for simple liar and emphatic liar. We observed that for a simple liar, the dynamics is oscillatory, whereas for an emphatic liar it is unstable with one repellent fixed point only. The next example illustrates the case of a chaotic liar.

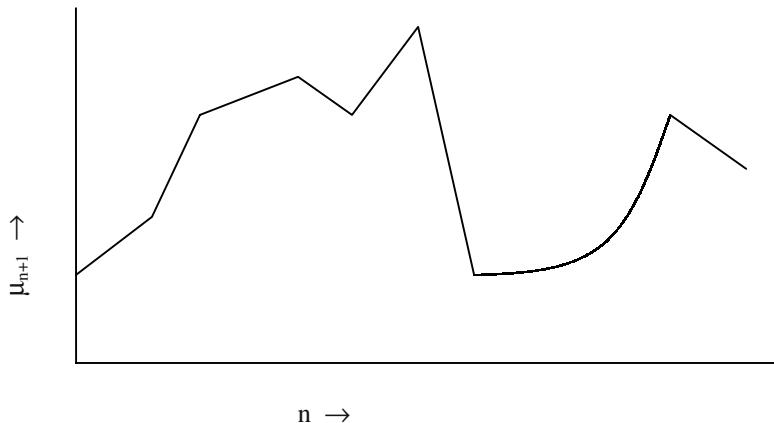


**Fig. 22.6:** The phase trajectory of emphatic liar.

The chaotic liar states: this sentence is true if and only if it is false. Using the Lukasiewicz conditional and modeling self-reference in terms of iterated algorithms semantic values of the chaotic liar is given by

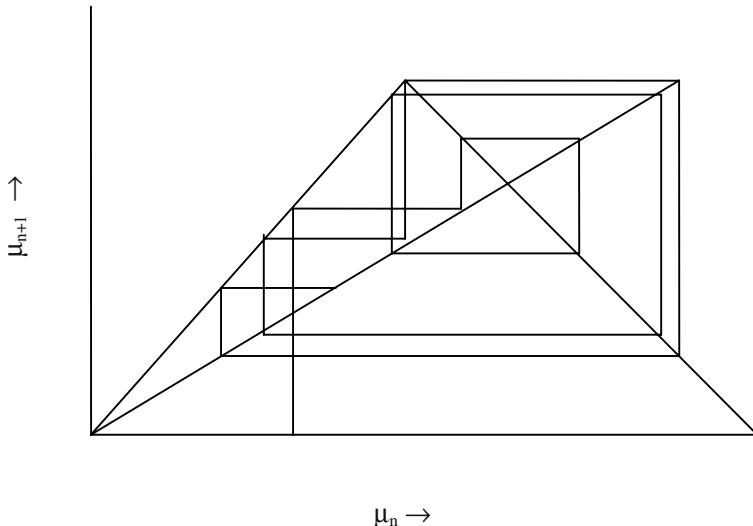
$$\mu_{n+1} = 1 - \text{abs}((1 - \mu_n) - \mu_n) \quad (22.22)$$

A seed value of  $\mu_0=0.314$  gives us a chaotic behavior of  $\mu_{n+1}$  over iterations n (Fig. 22.7).



**Fig. 22.7:** The chaotic behavior of the chaotic liar.

The phase trajectory of the chaotic liar is given in Fig. 22.8 below.



**Fig. 22.8:** The phase trajectory of the chaotic liar.

## 22.6 Rough Sets

Let  $X$  be a universal set and  $R$  be an equivalence relation on it. Remember that a relation  $R$  is a set of ordered pairs, defined by  $R = \{(x_1, x_2) : x_1 \in X \text{ and } x_2 \in X\}$ , and this relation is *equivalent* if the following three conditions are jointly satisfied:

- i)  $(x_j, x_j) \in R$  for all elements of the domain of  $R$  (reflexive)
- ii) If  $(x_1, x_2) \in R$  then  $(x_2, x_1) \in R$  (symmetric)
- iii) If  $(x_i, x_j)$  and  $(x_j, x_k) \in R$  then  $(x_i, x_k) \in R$  (transitive).

Before defining rough sets we should explain equivalent classes. An *equivalent class*  $A$  is a subset of  $X$  whose any two elements satisfy the above three characteristics of an equivalent relation.

**Example 22.2:** Let  $Z$  be a set of integers. Suppose  $R$  is an equivalent relation on  $Z$ , such that  $R = \{(x, y) : x \equiv y \pmod{5} \text{ for } x, y \in Z\}$ . Since  $x \equiv y \pmod{5}$  we write:  $x =$

$5y + r$ , where  $r$  has 5 possible values; 0, 1, 2, 3 and 4. Consequently, we obtain 5 equivalent classes:  $E_0, E_1, E_2, E_3$  and  $E_4$  where

$$\begin{aligned}E_0 &= \{x: x = 5y + r, r=0\}, \\E_1 &= \{x: x = 5y + r, r=1\}, \\E_2 &= \{x: x = 5y + r, r=2\}, \\E_3 &= \{x: x = 5y + r, r=3\}, \\E_4 &= \{x: x = 5y + r, r=4\}.\end{aligned}$$

Let  $X/R$  denote the family of equivalent classes induced by  $X$  on  $R$ . One such class in  $X/R$  that contains  $x$  belonging to  $X$  is designated by  $[x]_R$ . For any output class  $A \subseteq X$ , we define the lower approximation  $R_1(A)$  and the upper approximation  $R_2(A)$ , which approach  $A$  as closely as possible from inside and outside of  $A$  respectively. Here,

$$R_1(A) = \cup\{[x]_R: [x]_R \subseteq A, x \in X\} \quad (22.23)$$

is the union of all equivalent classes in  $X/R$  that are contained in  $A$ , and

$$R_2(A) = \cup\{[x]_R: [x]_R \cap A \neq \emptyset, x \in X\} \quad (22.24)$$

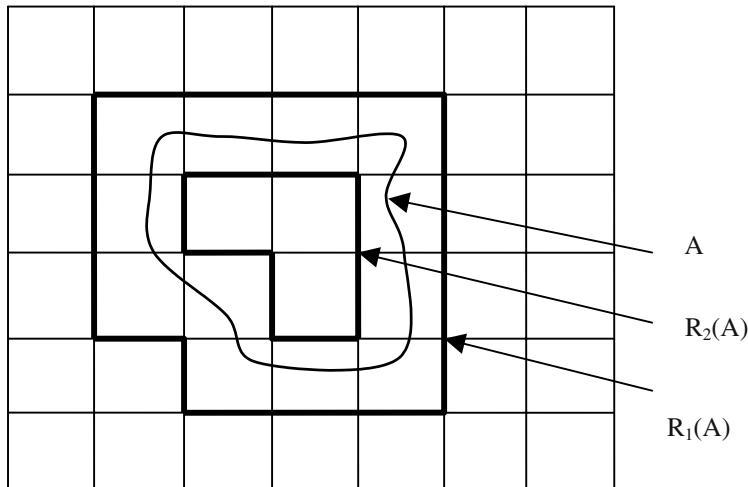
Is the union of all equivalent classes in  $X/R$  that overlaps with  $A$ .

A *rough set*  $R(A) = \langle R_1(A), R_2(A) \rangle$  is a representation of the given set  $A$  by  $R_1(A)$  and  $R_2(A)$  [14, 15]. The set  $BN(A) = R_1(A) - R_2(A)$  is a rough description of the boundary of  $A$  by the equivalent class of  $X/R$ .

The rough membership function  $r_A(x): A \rightarrow [0, 1]$  of a pattern  $x$  belonging to  $X$  for the output class  $A$  is defined by

$$r_A[x] = \frac{\|[x]_R \cap A\|}{\|[x]_R\|} \quad (22.25)$$

**Example 22.3:** This example illustrates the geometric boundaries of rough sets. Consider the points set  $A$  represented by the closed curve in Fig. 22.9. We can approximately represent the set  $A$  by two boundaries  $R_1(A)$  and  $R_2(A)$ . The boundary  $R_2(A)$  attempts to approximate  $A$  from inside, while boundary  $R_1(A)$  attempts to approximate  $A$  from outside.



**Fig. 22.9:** The rough boundaries  $R_1(A)$  and  $R_2(A)$  of a given point set  $A \subseteq X$ .

### 22.6.1 Rough-Fuzzy Sets

Let  $X$  be a universal set,  $R$  be an equivalent relation defined on  $X$ , and the output class  $A \subseteq X$  be a fuzzy set. A rough-fuzzy set is a tuple  $\langle R_1(A), R_2(A) \rangle$  where the lower approximation  $R_1(A)$  and the upper approximation  $R_2(A)$  are fuzzy sets of  $X/R$ . The membership functions in the present context are defined as follows:

$$\mu_{R1(A)}([x]_R) = \inf \{\mu_A(x) | x \in [x]_R \subseteq R_1(A)\} \quad (22.26)$$

$$\mu_{R2(A)}([x]_R) = \sup \{\mu_A(x) | x \in [x]_R \subseteq R_2(A)\} \quad (22.27)$$

In (22.26) and (22.27),  $\mu_{R1(A)}([x]_R)$  and  $\mu_{R2(A)}([x]_R)$  represent the membership values of  $[x]_R$  in  $R_1(A)$  and  $R_2(A)$  respectively.

The rough-fuzzy membership function of a pattern  $x \in X$  for the fuzzy output class  $C_c \subseteq X$  is defined by

$$M_{C_c}(x) = \frac{\|F \cap C_c\|}{\|F\|} \quad (22.28)$$

where  $F=[x]_R$  and  $C_c =$  the cardinality of the fuzzy set  $C_c$ . One way of determining cardinality of  $C_c$  is to use

$$\|C_c\| = \sum_{x \in X} \mu_{C_c}(x) \quad (22.29)$$

$$\text{and } \mu_{A \cap B}(x) = \min\{\mu_A(x), \mu_B(x)\}. \quad (22.30)$$

$$\forall x \in X$$

When the output class is crisp, definition (22.28) simplifies to (22.25).

## 22.7 Granular Computing

According to Zadeh, information granulation involves partitioning a class of objects (points) into granules, where each granule is a clump of objects (points) drawn together by the indistinguishability, similarity or functionality [17].

The above definition reveals the following points:

1. Information granules are obtained through partitioning a class of objects.
2. A granule consists of a number of objects/ points.
3. The objects/ points in a granule are selected by their similarity, inseparability or functionality.

One simple example of information granulation is the segmentation of a digital image. Consider an image consisting of trees, sky, lake and the roads surrounding the lake. A segmentation algorithm partitions the image into regions containing the sky, the lake and the roads. It is important to note that each region includes a number of pixels or points having some common image characteristics.

A number of formal frameworks are currently being used for the construction of information granules [16]. The frameworks include

1. Set theory and interval analysis
2. Fuzzy sets
3. Rough sets
4. Shadowed sets
5. Probabilistic sets
6. Higher level granular constructs.

An interesting point to note is that these theories were developed almost independently, but they are used individually or jointly in the construction of information granules.

Now, we can formally define an information granule system S as a 5-tuple:

$$S = \langle E, O, G, H, \{v_p\}_{p \in H} \rangle$$

where

$E$ = a finite set of objects called information granules,

$O$ = a set of operators on granules,

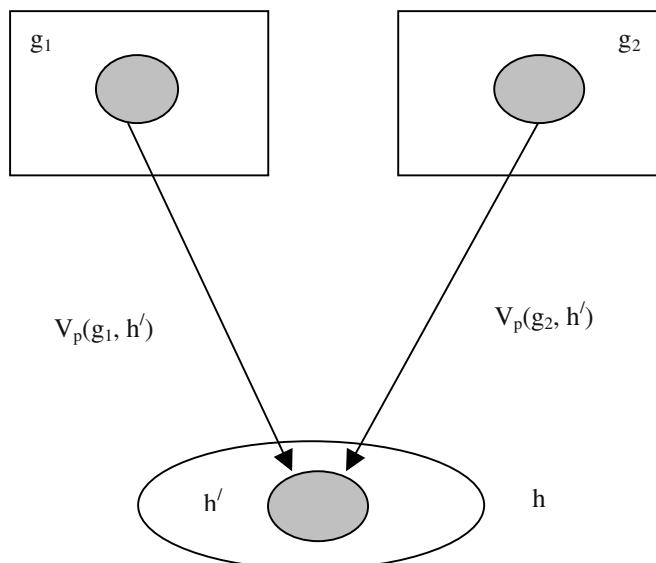
$G$ = a subset of the set of granules generated from  $E$  by means of operation  $O$ ,

$H$ = a degree set with partial ordering,

$V_p$  subset of  $G \times G$ = the relation to be a part to a degree  $p$ .

The information granulation system available with agent  $A_1$  is thus given by

$$S(A_1) = \langle E(A_1), O(A_1), G(A_1), H(A_1), \{v_p(A_1)\}_{p \in H(A_1)} \rangle.$$



**Fig. 22.10:** Computing rough inclusion and indiscernibility.

### 22.7.1 Rough Inclusion and Indiscernibility

Let

$X$  and  $Y$  be two sets of objects,  
 $v_p$  be a relation between  $X$  and  $Y$ ,  
 $p \in [0, 1]$  be a degree that measures the strength of the relation  $v_p$ .

We say that  $v_p(X, Y)$  exists if and only if

$$\left. \begin{array}{l} \text{either } |X \cap Y| / |X| \geq \text{the prescribed value of } p \\ \text{or } X \text{ is a null set.} \end{array} \right\} \quad (22.31)$$

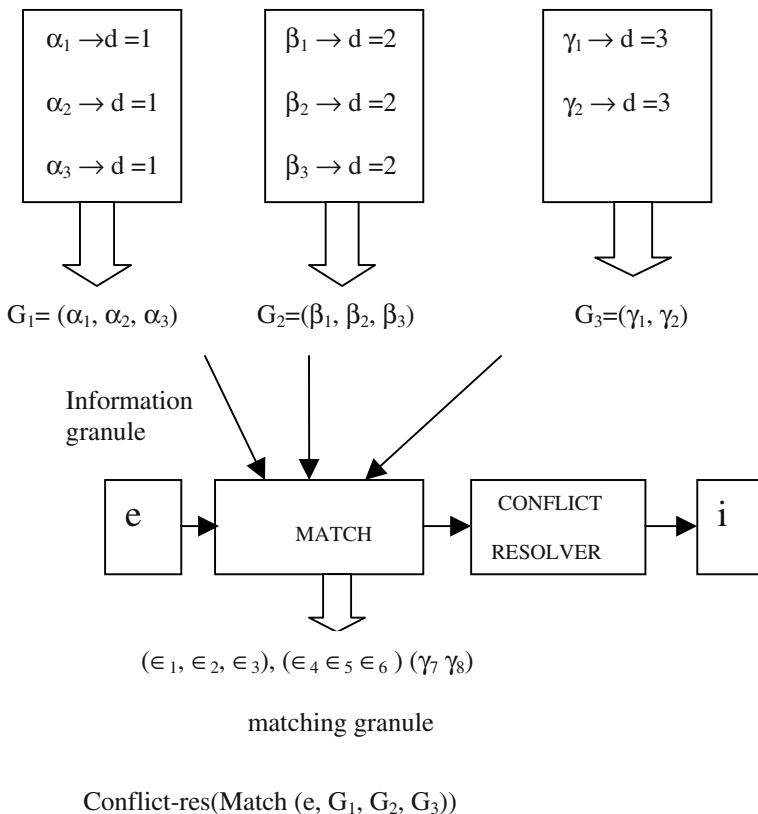
Closeness of sets  $X$  and  $Y$  (or rough inclusion) can be measured by the above definition. The closer is the value of  $p$  to unity, the closer is the relation between  $X$  and  $Y$ .

We now define indiscernibility of two subsets  $g_1$  and  $g_2$ . Suppose, we compute:  $v_p(g_1, h')$  and  $v_p(g_2, h')$ . If they are close enough, we say that the granules  $g_1$  and  $g_2$  are indiscernible with respect to the degree set  $h$ . Fig. 22.10 demonstrates the rough inclusion and indiscernibility of  $g_1$  and  $g_2$ .

### 22.7.2 Classifier Design

We can extend the principles of information granule matching in classifier design. Let us assume that we have three information granules  $G_1$ ,  $G_2$  and  $G_{3e}$ . Further assume that  $e$  is new information granule. We design a matcher using the principles of rough indiscernibility. The matcher generates corresponding matching granules, and the conflict resolver identifies the best matching and outputs the class of the input granule  $e$ . The details of the scheme is given in Fig. 22.11 below.

Among other useful applications of granular computing, building agents with granular computational models and establishing communications among the agents in a distributed environment needs special mention. A simple scheme of agent design by granular models is available in Skowron's lecture presentation [17] on Soft Computing organized by CIMPA-UNESCO-INDIA school, held in Indian Statistical Institute, Calcutta



**Fig. 22.11:** A scheme of classifier design.

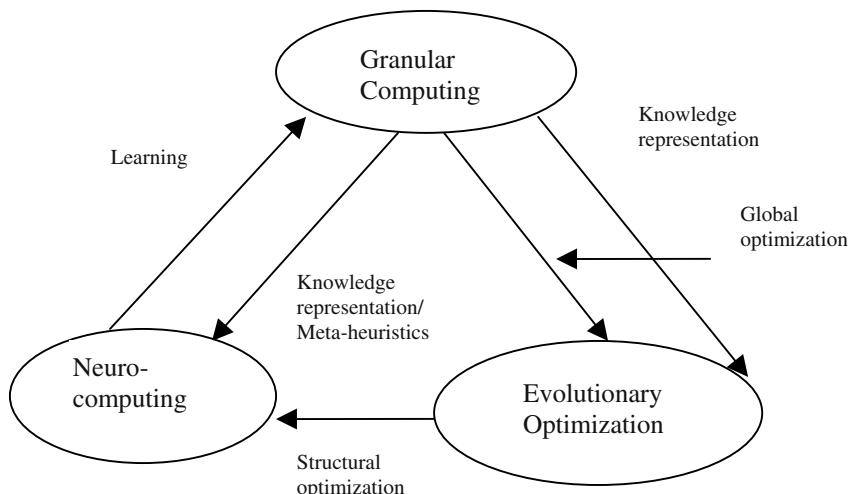
## 22.8 Redefining Computational Intelligence

In chapter 1, we defined Computational Intelligence as a consortium of fuzzy sets, neuro-computing, genetic algorithm, belief networks and artificial life. After reading this chapter, readers can form an opinion that most of the materials we discuss on fuzzy sets such as approximate reasoning and fuzzy clustering broadly fall in the category of granular computing. The classical GA, particle swarm optimization and genetic programming can be covered under evolutionary computing. Lastly, all neural learning algorithms and immune computing can be together called neuro-computing.

Computational intelligence can now be redefined as a consortium of granular computing, neuro-computing and evolutionary computation. Fig. 22.12 presents a synergistic relationship among these modules. Granular computing helps neuro-computing by suggesting guidelines in the construction of neurons from knowledge representation viewpoint. It also helps evolutionary computation by providing suitable strategies for genetic coding from the viewpoint of knowledge representation. Evolutionary computing provides necessary support to neuro-computing and granular computing in optimization of system resources. Neuro-computing provides behavioral support to adapt system resources of granular computing by its learning algorithms.

## 22.9 Summary

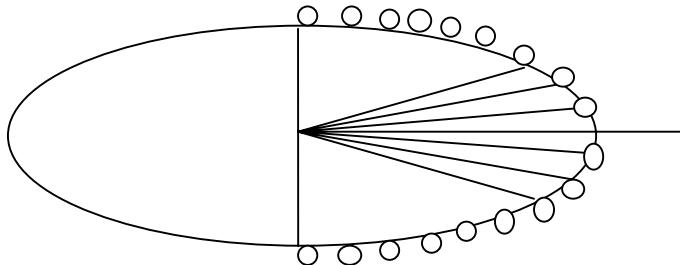
The chapter introduced many topics of current interest in a nutshell. Artificial life is discussed with an illustrative case study on artificial fish schools. Principles of particle swarms are introduced as an alternative optimization technique in computational intelligence. Immune computing model is outlined with special reference to garbage cleaning by a mobile robot. Fuzzy chaos theory is outlined with an illustrative problem of modeling fuzzy liar. Elements of rough sets are discussed from their first principles. Granular computing is outlined from an engineering standpoint. The scope of the emerging techniques in revising the definition of computational intelligence is discussed at the end of the chapter.



**Fig. 22.12:** Redefining computational intelligence by emerging technologies.

## Exercise

- Given 18 fish agents forming an elliptical region, each attaining  $\pm k (10^\circ)$  with respect to the major axis, where  $k$  is an integer in [1..9]. See Fig. 22.12 below. Determine the average center distance of all agents.



**Fig. 22.12:** A school of 18 fish agents, represented by small circles, forming a semi-elliptical array.

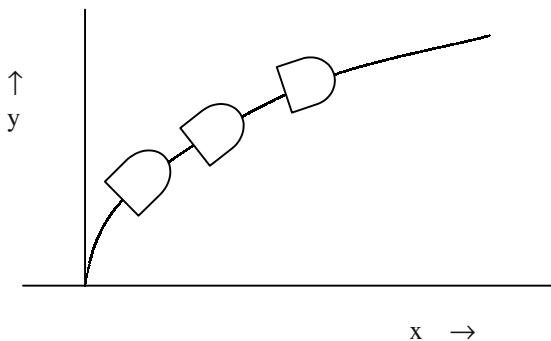
[**Hints:** Use the parametric equation of an ellipse  $x=a \cos \theta$ ,  $y=a \sin \theta$ , where  $\theta=\pm k (10^\circ)$ . Then determine the center of gravity of the fish school.]

$$\text{Average center distance} = \frac{1}{N} \sum_{i=1}^N \|X^* - X_i\|,$$

where  $X^*$  denotes the center of gravity and  $X_i$  is the position of the  $i$ -th agent.  $N$  denotes the population of the fish agents. Note that

$$X^* = \frac{1}{N} \sum_{i=1}^N X_i$$

- The centre of gravity of an elliptical fish school traverses a parabolic path  $y^2 = 4ax$  (See Fig. 22.13 below). Determine the group velocity of the fish school at  $y=2a$ , when  $dx/dt = 2$  m/s.



**Fig. 22.13:** The center of gravity of a semi-elliptical fish school traverses a parabolic path.

[**Hints:** Use the following definitions:

$$v_x = \frac{dx}{dt}, \text{ and } v_y = \frac{dy}{dt}.$$

Given that  $y^2 = 4ax$ .

$$\text{Therefore, } 2y \frac{dy}{dt} = 4a \frac{dx}{dt}.$$

$$\text{which implies } \frac{dy}{dt} = (2a/y) \frac{dx}{dt}.$$

$$\text{Group velocity } v = \sqrt{(v_x^2 + v_y^2)}$$

$$= \frac{dx}{dt} \sqrt{1 + (2a/y)^2}$$

Taking  $dx/dt = 2 \text{ m/s}$  and  $y = 2a$ , we obtain  $v = 2\sqrt{2} \text{ m/s.}$ ]

3. Construct the model of a fuzzy liar for the following statement:

That the given sentence is very true is fairly false,

and show that the dynamics has a fixed point at  $\mu_n = 1/\sqrt{2}$ .

**[Hints:** To model very true we use  $\mu_n^2$ , and to represent fairly false, we use  $\sqrt{1-\mu_n^2}$ . Thus,  $\mu_{n+1} = \sqrt{1-\mu_n^2}$ . Let  $\mu^*$  be the fixed point, if any. Then

$$\mu^* = \sqrt{1 - \mu^{*2}}$$

$$\text{or, } \mu^{*2} = 1 - \mu^{*2}$$

$$\text{or, } \mu^* = 1/\sqrt{2}.$$

4. How can you represent the following fuzzy statement about the fuzzy liar by 2 simple sentences?

The fuzzy liar is very true.

Hence construct the dynamics of the sentence.

**[Hints:** We can represent the given sentence by the following pair of sentences.

The sentence is false. (1)

(1) is very true. (2)

Let  $\mu_n(x)$  be the membership of a given sentence to be true. Then the model of (1) is given by  $1 - \mu_n(x)$ .

The model of (2) is given by

$$\mu_{n+1}(x) = (1 - \mu_n(x))^2]$$

5. Represent the following into two simple statements and hence determine the dynamics. Show that the dynamics has a fixed point attractor.

The modest liar is fairly false.

**[Hints:** The pair of sentences are:

The statement is false. (1)

(1) is fairly true. (2)

Thus we have  $\mu_{n+1} = \sqrt{(1 - \mu_n)}$ .

To show that the dynamics has a fixed point, let  $\mu_{n+1} = \mu_n = \mu^*$  at steady state.

Then  $\mu^* = \sqrt{(1 - \mu^*)}$

$$\text{or, } \mu^{*2} = 1 - \mu^*$$

$$\text{or, } \mu^{*2} + \mu^* - 1 = 0$$

$$\text{or, } \mu^* = \frac{-1 \pm \sqrt{5}}{2}.$$

The only fixed point that lies in  $0 \leq \mu^* < 1$  is  $\mu^* = \frac{-1 + \sqrt{5}}{2}$ .

To show that the above fixed point is an attractor, we plot phase trajectories between  $\mu_{n+1}$  and  $\mu_n$ , and take any arbitrary seed point. We can check that the dynamics settles down to a point with co-ordinates  $((-1 + \sqrt{5})/2, (-1 + \sqrt{5})/2)$ .]

6. Four mobile robots are employed to carry an elliptical shaped object towards a given goal position amidst obstacles. Explain how particle swarm optimization can be applied here to solve this problem.

**[Hints:** Consider 4 equal fragments on the elliptical object and determine the position of robots in each fragment so that the object moves in the desired goal direction.]

7. Consider the following model between two antibody population a and b.

$$\frac{da}{dt} = \alpha a b - \beta a$$

$$\frac{db}{dt} = \gamma a b - \delta b.$$

where  $\alpha, \beta, \gamma$  and  $\delta$  are rate constants.

- a) Show that at dynamic equilibrium  $a/b = \alpha\delta/\beta\gamma$ .

- b) Hence justify that “a“ grows as  $\alpha$  or  $\delta$  increases , whereas “b” grows as  $\beta$  or  $\gamma$  increases at near steady-state.

[**Hints:** a) Setting  $da/dt = 0$  and  $db/dt = 0$  and simplifying the resulting expressions we arrive at  $a/b = \alpha\delta/\beta\gamma$ .

b) From the results obtained in (a), we write  $a \propto \alpha\delta$ , and  $b \propto \beta\gamma$ . Hence the results follow.]

8. Show that increasing  $k$  has a role in the growth of antibody  $a$  and decay of antigen  $b$  in the following dynamics:

$$\frac{da}{dt} = \alpha (a \cdot b)^k - \beta a$$

$$\frac{db}{dt} = \gamma b - \delta (ab)^k.$$

[**Hints:** Let  $F_1 = \alpha (a \cdot b)^k - \beta a$  and  $F_2 = \gamma b - \delta (ab)^k$ .

Here,  $\frac{\partial F_1}{\partial k} = \alpha k (a b)^{k-1} > 0$  for positive  $a, b$  and  $k$ .

Thus  $F_1$  is monotonically increasing.

Similarly, it can be shown that  $\frac{\partial F_2}{\partial k} < 0$ , and thus it is monotonically decreasing. Hence, the conclusion.]

9. Let  $Z$  be a set of integers, and  $R= \{ (x, y) | x \equiv y \text{ mod } 5 \}$  for  $x, y \in Z$  be an equivalent relation on  $Z$ . Determine the equivalent classes.

[**Hints:** There are exactly 5 equivalent classes in  $Z/R$ . Let them be  $E_0, E_1, E_2, E_3$  and  $E_4$ . Since each integer  $x$  is uniquely expressible in the form  $x = 5q + r$  where  $0 \leq r < 5$ , then  $x$  is a member of the equivalence class  $E_r$ , where  $r$  is the remainder. Thus

$E_0 = \{ \dots, -10, -5, 0, 5, 10, \dots \}$   
 $E_1 = \{ \dots, -9, -4, 1, 6, 11, \dots \}$   
 $E_2 = \{ \dots, -8, -3, 2, 7, 12, \dots \}$   
 $E_3 = \{ \dots, -7, -2, 3, 8, 13, \dots \}$   
 and  $E_4 = \{ \dots, -6, -1, 4, 9, 14, \dots \}$ .

10. Determine the rough sets from the equivalent classes obtained in problem 9.

[**Hints:** The rough boundaries in the present context are  $R_1(A) = R_2(A) = Z$ . The reader should verify this using the definition of rough sets.]

## References

- [1] Breeder, J. M., "Equations descriptive of fish schools and other animal aggregations," *Ecology*, vol. 35, no. 3, pp. 361-369, 1954.
- [2] Buckley, J. J., "Fuzzy dynamical systems," *Proc. of IFSA'91*, Brussels, Belgium, pp. 16-20, 1991.
- [3] Eberhart, R. C. and Kennedy, J., "A new optimizer using particle swarm theory," *Proc. of the Sixth Int. Symp. on Micro Machine and Human Science*, Nayoga, Japan, 1995.
- [4] Eberhart, R. C., Simpson, P. and Dobbins, R., *Computational Intelligence PC Tools*, pp. 212-226, Academic Press, 1996.
- [5] Grim, P., "Self-Reference and Chaos in Fuzzy Logic," *IEEE Trans. on Fuzzy Systems*, vol. 1, no. 4, Nov. 1993.
- [6] Ishiguro, A., Watanabe, Y., Kondo, T. and Uchikawa, Y., Artificial Immune Network and Its Application to Robotics, In *Soft Computing for Intelligent Robotic Systems*, Jain, L. C. and Fukuda, T. (Eds.), Physica-Verlag, 1998.
- [7] Jerne, N. K., "The immune systems," *Scientific American*, vol. 229, no. 1, pp. 52-60, 1973.
- [8] Jerne, N. K., "The generative grammar of the immune system," *EMBO Journal*, vol. 4, no. 4, 1983.
- [9] Jerne, N. K., "Idiotypic networks and other pre-conceived ideas," *Immunological Review*, vol. 79, 1984.
- [10] Kennedy, J. and Eberhart, R. C., *Swarm Intelligence*, Morgan-Kaufman, 2001.

- [11] Kunz, H. and Hemelrijk, C. K., "Artificial fish schools: Collective effects of school size, body size and body form," *Artificial Life*, vol. 9, pp. 237-253, 2003.
- [12] Langton, C. G. (Ed.), *Artificial Life: An Overview*, MIT Press, Cambridge, Massachusetts, 1995.
- [13] Roy, P. K., Kozma, R. and Datta Majumder, D., "From Neurocomputation to Immunocomputation: A Model and Algorithm for Fluctuation-Induced Instability and Phase Transition in Biological System," *IEEE Trans. on Evolutionary Computation*, vol. 6, no. 2, 2002.
- [14] Pawlak, Z., "Rough Sets," *Int. J. of Computer and Information Science*, vol. 11, pp. 341-356, 1982.
- [15] Pawlak, Z., *Rough Sets: Theoretical Aspects of Reasoning about Data*, Kluwer Academic Press, Dordrecht, 1991.
- [16] Pedrycz, W., *Granular Computing: An Introduction*, Kluwer Academic Press, Dordrecht, 2002.
- [17] Skowron, A., Rough Sets: Priliminaries, In *Lecture Notes of CIMPA-UNESCO-INDIA School on Soft Computing Approach to Pattern Recognition and Image Processing*, Indian Statistical Institute, Calcutta, December 2-13, 2002.
- [18] Taylor, C. and Jefferson, D., Artificial Life as a tool for biological inquiry, In *Artificial Life: An Overview*, Langton, C. G. (Ed.), MIT Press, Massachusetts, 1995.
- [19] Warburton, K. and Lazarus, J., "Tendency-distance models of social cohesion in animal groups," *Journal of Theoretical Biology*, vol. 150, pp. 473-488, 1991.

# 23

# Research Problems for Graduate Thesis and Pre-Ph D Preparatory Courses

*This chapter addresses selected research problems in computational intelligence. The problems are introduced informally so that anyone without any background in the specific domain easily understands them. The problems require either a mathematical formulation or a computer simulation for their solutions. An outline to the solution of the problems is also suggested.*

## **23.1 Problem 1: Computing Max-Min Inverse Fuzzy Relation**

Given a relational matrix  $R$ , the problem is to determine a matrix  $Q$ , such that

$$Q \circ R = I', \quad (23.1)$$

$$\text{where } I' \approx I. \quad (23.2)$$

The measure of approximate equality of  $I'$  to  $I$  is expressed by

$$m = \sum_i \sum_j e_{ij}^2 = \sum_i \sum_j (I_{ij} - I'_{ij})^2 \quad (23.3)$$

The problem thus can be re-formulated as follows [15-16].

Evaluate  $Q$  such that

$$Q \circ R = I'$$

where  $m = \sum_i \sum_j (I_{ij} - I'_{ij})^2$  is minimum.

## 23.2 An Outline to the Solution of Problem 1

We construct a heuristic function  $h(q_{ij})$  that maximizes the min terms ( $q_{ij} \wedge r_{jk}$ ), having contribution to the one in the resulting  $I$  matrix, and minimizes the min terms ( $q_{ij} \wedge r_{jk}$ ), having contribution to the zeros in the  $I$  matrix. Example 23.1 illustrates the objective discussed above.

**Example 23.1:** Let

$$R = \begin{pmatrix} 0.2 & 0.6 & 0.9 \\ 0.3 & 0.8 & 0.4 \\ 0.5 & 0.7 & 0.2 \end{pmatrix}.$$

Suppose we need to determine  $q_{11}$  first. Now, to construct a heuristic function  $h(q_{11})$ , we first expand  $Q \circ R = I$  by looking at the terms involving  $q_{11}$ . We note the equations that involve  $q_{11}$  are:

$$(q_{11} \wedge 0.2) \vee (q_{12} \wedge 0.3) \vee (q_{13} \wedge 0.5) = 1.0 \quad (23.4)$$

$$(q_{11} \wedge 0.6) \vee (q_{12} \wedge 0.8) \vee (q_{13} \wedge 0.7) = 0.0 \quad (23.5)$$

$$(q_{11} \wedge 0.9) \vee (q_{12} \wedge 0.4) \vee (q_{13} \wedge 0.2) = 0.0 \quad (23.6)$$

It is clear from (23.4 – 23.6) that we need to maximize  $q_{11}$  in (23.4) so as to generate a 1 in the right hand side. On the other hand, we need to minimize  $q_{11}$  in (23.5) and (23.6) as the right hand sides of them are zero. We thus construct a heuristic function to jointly satisfy the constraints mentioned above. One such function is given below.

$$h(q_{11}) = (q_{11} \wedge 0.2) - V \{ (q_{11} \wedge 0.6), (q_{11} \wedge 0.9) \}, \quad (23.7)$$

which is to be maximized.

We can now make an exhaustive search of  $q_{11}$  in  $[0, 1]$  to determine the value of  $q_{11}$  for which  $h(q_{11})$  is maximized.

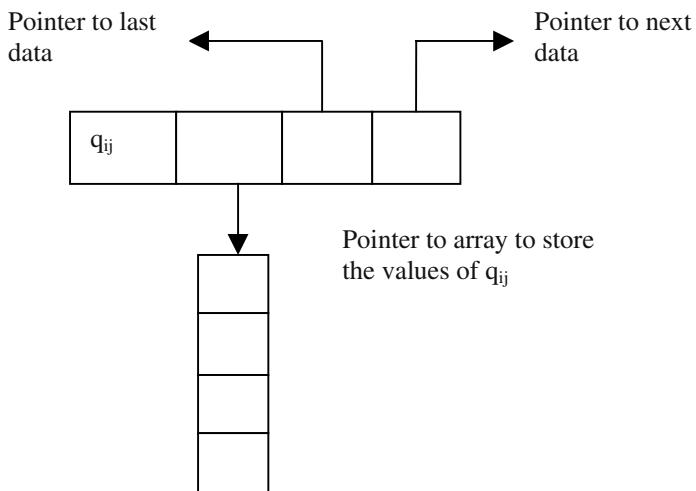
### 23.3 Tutorial Assignments for Graduate Students

Students undertaking a graduate thesis in Computer Science/ Mathematics/ Electrical Engineering should verify that

$[h(q_{11})]$  for  $q_{11} \in \{r_{1j}, r_{2j}, r_{3j}\} \geq h(q_{11})$  for  $q_{11} \in [0, 1]$ . It is to be noted that the above property significantly reduces the search complexity.

- a) Write the general heuristic function  $h(q_{ij})$  for evaluating  $q_{ij}$  following example 23.1 used for construction of  $h(q_{11})$ .
- b) In case  $q_{ij}$  has multiple solution, how will you store them? - select a suitable data structure to save the values of  $q_{ij}$ .

**[Hints:** A bi-directional linked list as shown below may be used to store the values of  $q_{ij}$ .



**Fig. 23.1:** The suggested data structure for  $q_{ij}$ .

The last and the next data in the above structure refer to position-wise predecessor and successor matrix elements. For example the last and next data of  $q_{22}$  are  $q_{21}$  and  $q_{23}$  respectively.]

c) Show that  $Q=R^T$  is always a solution by the approach we suggested.

d) How will you select the best  $Q$  when there are multiple solution.

[**Hints:** Identify the  $Q$  that yields the minimum  $m$  by equation (23.3).]

e) Is the  $Q$  obtained in (iv) optimal?

[**Answer:** No, finding an optimal  $Q$  is an open-ended problem for a Masters/ Ph D thesis.]

## 23.4 Problem 2: Reasoning with Fuzzy Petri Nets

This section includes both tutorial and research problems on fuzzy Petri nets (FPN) [9], [14], [17]. Tutorial problems are meant for graduate students. Research problems are addressed for researchers interested in FPN or related disciplines.

### 23.4.1 Tutorial Problems on Fuzzy Petri Nets

The following 3 tutorial problems can provide an insight to many interesting issues in FPN.

a) In a purely cyclic fuzzy Petri net, all places and transitions lie on a cycle.  
Show that in a purely cyclic network with  $k$ -transitions

$$(Q \circ P)^k = I, \quad (23.8)$$

where  $I$  is the identity matrix.

b) Show that a steady-state solution of the equation

$$N(t+1) = P \circ (Q \circ N^c(t))^c \quad (23.9)$$

is given by

$$N^* = P \circ (Q \circ (N^*)^c)^c \quad (23.10)$$

where  $N^*$  is the steady-state value of  $N(t)$  and  $P=Q=I$  is one trivial solution of the last equation.

[**Hints:** With  $P=Q=I$ ,  $N^* = P \circ (Q \circ (N^*)^c)^c$  (23.11)

$$= (I \circ (I \circ (N^*)^c)^c) \quad (23.12)$$

$$= I \circ ((N^*)^c)^c \quad (23.13)$$

$$= N^*. \quad (23.14)$$

- c) Given  $N(t+1)$ ,  $P$  and  $Q$  matrices, how can you evaluate  $N(t)$  in backward time?

[**Hints:** Determine  $P^{-1}$  and  $Q^{-1}$  with respect to max-min composition operator introduced in Problem 1, and then evaluate  $N(t)$  from  $N(t+1)$  as follows:

$$N(t+1) = P \circ (Q \circ N^c(t))^c \text{ (given)}$$

Pre-multiplying both sides by  $P^{-1}$  we obtain:

$$P^{-1} \circ N(t+1) = (Q \circ N^c(t))^c \quad (23.15)$$

Taking complements on both sides of (23.15), and exchanging the L.H.S and the R.H.S we obtain:

$$Q \circ N^c(t) = (P^{-1} \circ N(t+1))^c \quad (23.16)$$

Pre-multiplying both sides of (23.16) by  $Q^{-1}$ , and taking complement on the results on both sides we finally obtain:

$$N(t) = [Q^{-1} \circ (P^{-1} \circ N(t+1))]^c \quad (23.17)$$

### 23.4.2 Research Problems on Fuzzy Petri Nets

The study of controllability is given prime consideration in classical control theory. Roughly speaking, controllability aims at controlling the system states from a given initial state  $X(0)$  to a desired state  $X(T)$  by suitably selecting the control signal  $u(t)$  over the desired timeframe  $0 < t \leq T$ . In this section, we propose the following problem to control the state vector  $N(t)$  of a FPN using two additional matrices  $P'$  and  $Q'$ .

- d) Suppose we want to reach the state  $N(t^*) = N^*$  (say) from a starting state  $N(0)$  in the basic belief updating equation:

$$N(t+1) = P \circ (Q \circ N^c(t))^c.$$

What control policy should we adopt to handle the problem?

[**Hints:** We can incorporate two fuzzy matrices:  $P'(t)$  and  $Q'(t)$  in the basic belief updating equation as follows.

$$N(t+1) = (P \vee P') \circ ((Q \wedge Q') \circ N^c(t))^c \quad (23.18)$$

The matrices  $P'$  and  $Q'$  should be adapted in a manner so as to make necessary corrections in  $N(t+1)$  during each iteration within a finite time frame of  $0 < t \leq t^*$ . One approach to handle this problem is to strengthen the connectivity from transition to places and to weaken the connectivity from places to transitions. The former is needed to enhance the elements of  $N(t+1)$ , while the latter is needed to reduce the elements of  $N(t+1)$ . But suppose we want to increase one component of  $N(t+1)$  and decrease another component of  $N(t+1)$  simultaneously. The following algorithm may be invoked to solve the problem.

1. Compute Error Vector  $E(t+1) = N^* - N(t+1)$ .
2. If the  $i$ -th component of  $E(t+1)$  is positive, increase the  $i$ -th row of  $P'$  by

$$[P'(t+1)]_{i\text{-th row}} = [P'(t)]_{i\text{-th row}} + [E(t+1)]_{i\text{-th row}},$$

Else if  $i$ -th component of  $E(t+1)$  is negative, then reduce the  $i$ -th row of  $Q'$  by

$$[Q'(t+1)]_{i\text{-th row}} = [Q'(t)]_{i\text{-th row}} - [E(t+1)]_{i\text{-th row}}.$$

3. Repeat step 2 Until  $|N(t+1) - N^*|$  is below a prescribed margin

Note that the initial settings of  $P'(0)$  and  $Q'(0)$  should be zero.

- e) Next we consider a problem of system identification using FPN. Suppose we have two fuzzy connectivity matrices  $P$  and  $Q$ , i.e. instead of rigid connectivity from places to transitions and vice-versa the connectivities are fuzzy. Under this circumstance, how can we determine  $P$  and  $Q$  matrices such that we can obtain a  $N^*$  from a given  $N(0)$ ?

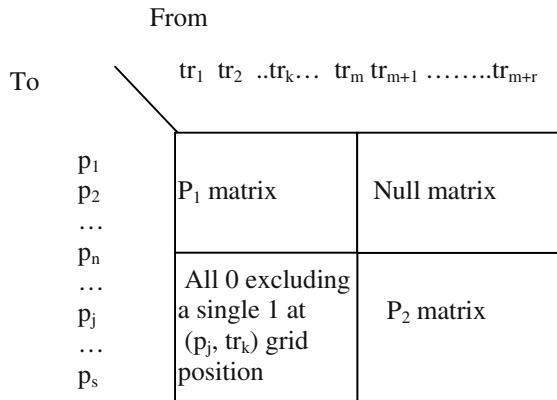
[**Hints:** Use GA to adapt  $P$  and  $Q$  matrices so that  $N(t+1)$  is close enough to  $N^*$ . The fitness function in this case is given by

$$f = 1 / \|N(t+1) - N^*\|$$

where  $\| . \|$  is a Euclidean norm. Matrix type crossover operation may be used to construct new population of  $P$  and  $Q$  matrices. Finally select near-optimal  $P$  and  $Q$  after the GA converges.]

- f) Let  $P_1$ ,  $Q_1$  and  $P_2$  and  $Q_2$  be the  $P$  and  $Q$  matrices of two FPN. Suppose, we need to establish a connection from transition  $t_r$  of net 1 to place  $p_j$  of net 2. Show with an example how the overall  $P$  and  $Q$  matrices of the complete net can be constructed.

[**Hints:** The structure of the overall  $P$  matrix is given below



Similarly construct Q matrix. Check that the Q matrix includes two null matrices. Also write the state equation of the overall system by considering the state vectors of individual FPN.]

### 23.5 Problem 3: Spoken Word Reconstruction from Lip Stylus Sequences

Consider the problem of reconstructing spoken words from facial image sequences [22]. The problem can be sub-divided into 3 main parts. First, we have to segment the lip-region from the rest of the facial image in each image frame. Secondly, we have to identify the spatial samples of mouth opening in the frames. Thirdly we need to map the spatially sampled frames to words by some algorithm. For convenience of the readers we break the problem into more detailed steps.

- a) Use *fuzzy C-means clustering (FCM)* algorithm to segment [10] the facial image into lip and non-lip regions.

[**Hints:** Represent the pixels in the image into R, G, B intensity values, as available in a .bmp file. Use (x, y) positions of the pixels along with their R, G, B values as the input of the FCM clustering algorithm. Set C= 2, and m= 1.01 in your program.]

- b) Represent the segmented lips region in monochrome format and plot average intensity/ row of the image against the height of the image. Determine the mouth opening in terms of the gap between the first and the third maxima in the plot.
- c) Is the result in part (b) unique?

- d) Spatially sample the mouth openings into 30 discrete slices. Repeat it on each grabbed frame used to indicate the major changes in mouth opening in pronouncing a word.
- e) Can we use back-propagation learning algorithm to reconstruct the spoken words from the spatial samples of all the significant frames?

## 23.6 Problem 4: Fuzzy State Equations and Stability Analysis

In classical control theory state variables are defined as variables, whose time derivatives are functions of the variables and inputs. In discrete domain, state variables at time  $(t+1)$  are function of the variables at time  $t$  plus input at time  $t$ . Fuzzy discrete state equations are equations of fuzzy state variables. To understand the implication of fuzzy state variables, let us take a concrete problem.

Let  $x_1(t)$  and  $x_2(t)$  be two fuzzy state variables. Given that

$x_1(t+1)$  is true if  $x_1(t)$  is true but  $x_2(t)$  is false, and if  $x_2(t)$  is true but  $x_1(t)$  is false.

Let  $\mu_1(t)$  and  $\mu_2(t)$  be the memberships of  $x_1(t)$  and  $x_2(t)$  at time  $t$ . Then we can represent  $\mu_1(t+1)$  and  $\mu_2(t+1)$  as a function of  $\mu_1(t)$  and  $\mu_2(t)$  as given below:

$$\mu_1(t+1) = a_{11}\{\mu_1(t) \tau (1 - \mu_2(t))\} + a_{12}\{(1 - \mu_1(t)) \tau \mu_2(t)\}, \quad (23.19)$$

where  $\tau$  denotes a t-norm operator. The parameters  $a_{11}$  and  $a_{12}$  control the state-transitions of the dynamics. Taking  $\tau$ -norm as the product, we re-write (23.19) as

$$\mu_1(t+1) = a_{11} \mu_1(t) (1 - \mu_2(t)) + a_{12} (1 - \mu_1(t)) \mu_2(t). \quad (23.20)$$

Further, assume that  $x_2(t+1)$  is true if  $(x_1(t) \rightarrow x_2(t))$  is true or  $x_2(t)$  is false. This can mathematically be represented by

$$\begin{aligned} \mu_2(t+1) &= a_{21} (\mu_1(t) \rightarrow \mu_2(t)) + a_{22} (1 - \mu_2(t)) \\ &= a_{21} \{1 - \{\mu_1(t) \tau (1 - \mu_2(t))\}\} + a_{22} (1 - \mu_2(t)) \end{aligned} \quad (23.21)$$

Taking  $\tau$ -norm as the product, we re-write (23.21) as follows.

$$\mu_2(t+1) = a_{21} \{1 - \{\mu_1(t) (1 - \mu_2(t))\}\} + a_{22} (1 - \mu_2(t)). \quad (23.22)$$

The question then arises is how to determine the stability of the fuzzy dynamical system given by (23.20) and (23.22). Section 23.7 presents a simple approach to study the stability of fuzzy dynamical systems.

### 23.7 An Outline to the Solution of Problem 4

We first compute the Jacobian  $J$ , given by

$$\begin{aligned}
 J(\mu_1, \mu_2) &= \text{Det} \begin{pmatrix} \partial\mu_1(t+1)/\partial\mu_1 & \partial\mu_1(t+1)/\partial\mu_2 \\ \partial\mu_2(t+1)/\partial\mu_1 & \partial\mu_2(t+1)/\partial\mu_2 \end{pmatrix} \\
 &= \begin{vmatrix} a_{11}(1 - \mu_2(t)) - a_{12}\mu_2(t) & -a_{11}\mu_1(t) + a_{12}(1 - \mu_1(t)) \\ -a_{21}(1 - \mu_2(t)) & a_{21}\mu_1(t) - a_{22} \end{vmatrix}. \quad (23.23)
 \end{aligned}$$

The local stability of the dynamics now can easily be analyzed by the following two steps. First substitute the co-ordinates of the point in  $J$ , around which we need to study the dynamics. Secondly, check whether the Eigen values of  $J$  are negative. Negative Eigen values ensures local stability around the selected point.

**Example 23.2:** Determine local stability of the given dynamics around  $(\mu_1, \mu_2) = (1, 1)$ .

$$J(1, 1) = \begin{vmatrix} -a_{12} & -a_{11} \\ 0 & a_{21} - a_{22} \end{vmatrix} \quad (23.24)$$

Now, to evaluate the Eigen values of  $J(1, 1)$ , we set  $\text{Det } [J(1,1) - \lambda I] = 0$ , and thus obtain:

$$\begin{vmatrix} (-a_{12} - \lambda) & -a_{11} \\ 0 & (a_{21} - a_{22}) - \lambda \end{vmatrix} = 0, \quad (23.25)$$

which yields

$$\lambda = -a_{12} \text{ or } (a_{21} - a_{22}). \quad (23.26)$$

Thus,  $\lambda$  is negative if  $a_{12} > 0$  and  $a_{22} > a_{21}$ .

## 23.8 Graduate Students' Assignments on Stability of Fuzzy State Equations

The following assignments will give an insight to the behavioral dynamics of the fuzzy state equations.

- i) Compute Eigen values at  $(\mu_1, \mu_2) = (0,0), (0,1), (1,0), (0.5, 0.5)$ , and determine the condition of stability/ limit cycles/ instability at those points.
- ii) Construct new dynamics of fuzzy state equations and for each model check stability of the dynamics at given local points.
- iii) Can we construct a Lyapunov energy function to study the stability of the dynamics given in section 23.6?
- iv) Verify the results of the stability analysis for the dynamics given in problem 4 from its numerical solution for selected values of the system parameters:  $a_{11}, a_{12}, a_{21}, a_{22}$ .
- v) Define  $d\mu_1/dt = \mu_1(t+1) - \mu_1(t)$ , and  $d\mu_2/dt = \mu_2(t+1) - \mu_2(t)$  and reformulate the given dynamical system under problem 4. How the numerical solution of the differential equation based system differs from its discrete counterpart?

## 23.9 Problem 5: Fuzzy Data Mining

Data mining [1], [4], [18] refers to extraction of knowledge from a given database. Usually, relational databases are used in most commercial applications. In this section, we would like to outline a novel scheme of knowledge extraction from relational databases.

Typical rules/ pieces of knowledge that carry some meaning from application point of view are presented below for convenience.

**Example 23.3:** Consider a LOAN relation that *includes borrower's name, amount of loan requested, sex and marital status* as 4 attributes. Suppose we extract the following knowledge from the database.

*Women usually take small loans, but widowed women request for large amount of loan.*

This knowledge definitely helps the bankers to identify their possible customers.

**Example 23.4:** Consider a SALES relation that includes the sale of different magazines in a locality. Suppose the following knowledge is extracted from the database.

*People who buy magazines on car also buy magazines on sports.*

Definitely such rules play a great role in identifying the potential customers on sports magazine.

In a fuzzy database, some attributes are fuzzified into HIGH, MEDIUM and SMALL fuzzy sets. Suppose, the relation contains 4 fuzzy variables:  $x, y, z$  and  $w$ . Then altogether, we have  $3 \times 4 = 12$  fuzzy sets. Now, suppose we would like to verify: whether  $a$  is  $A$  and  $b$  is  $B$ , where  $a, b \in \{x, y, z\}$ ,  $a \neq b$  and  $A, B \in \{\text{HIGH, MEDIUM, LOW}\}$  can jointly occur. There are different methods to attack the problem. We, however, propose a new technique based on fuzzy C-means clustering algorithm.

## 23.10 An Approach to the Solution of Problem 5

The problem under consideration may be re-formulated as follows. Determine the cluster centers for the fuzzy class:  $a$  is  $A$  and  $b$  is  $B$  and the rest. In order to handle this problem, we first initialize the memberships of the tuples in the given class and the non-class. Then we determine the cluster centers of both the given class and the rest by fuzzy c-means clustering (FCM) algorithm.

After identifying the cluster centers, we identify the count of total population (i.e., number of tuples) that fall in the given class. If the population of tuples in the given class is large, we regard it a valid cluster, signifying that the joint occurrence of  $a$  is  $A$  and  $b$  is  $B$  is meaningful.

One question that naturally arises is how to represent the data to differentiate the relations: i) a is HIGH and b is LOW and ii) a is MEDIUM and b is HIGH? Since the memberships are initialized randomly, we do not have any control on their selection. So, we need to pre-process the data before applying the FCM algorithm. In fact, to represent a variable LOW, MODERATE and HIGH we take the cube root, square root and square of the value of the variable respectively. An example of pre-processing a relational database suitable for applying FCM is given below.

**Example 23.5:** Consider an Employee relation consisting of the following attributes: employee-name, age, experience and salary. A fragment of the relation is given below for illustration.

**Table 23.1:** A fragment of the Employee Relation

Employee Name	Age	Experience	Salary
A. Konar	40	20	60K
A. Nath	50	30	70K
A. Basak	60	40	80K

Suppose, we want to check: whether MODERATE experience and HIGH Salary is valid in the given relation. The pre-processing of data here includes replacing experience by its square root and replacing salary by its square. The pre-processed relation is presented below.

**Table 23.2:** Moderate experience and High salary

Employee Name	Age	Moderate Experience	High Salary
A. Konar	40	$\sqrt{20}$	$(60K)^2$
A. Nath	50	$\sqrt{30}$	$(70K)^2$
A. Basak	60	$\sqrt{40}$	$(80K)^2$

It is thus clear that fuzzy variations in experience or salary can be taken into consideration by changing the values of the attributes using the notion of fuzzy quantifiers. It may be noted that such representation is not needed in classical fuzzy logic as we can represent the fuzzy quantifiers by membership functions.

Remember that to represent *experience x is MODERATE* we usually represent its membership  $\mu_{\text{MODERATE-EXPERIENCE}}(x) = \sqrt{\mu_{\text{NORMAL-EXPERIENCE}}(x)}$ , where the notations have their usual meaning.

## 23.11 Graduate Assignments on Fuzzy Data Mining

The following problems are suggested for both graduate students and pre-doctoral researchers.

- a) How many times do we have to run FCM to determine the association of two fuzzy variables, where the relation includes 4 fuzzy variables and each variable is represented into 3 fuzzy sets?

[**Hints:** Let x, y z, w be 4 fuzzy variables. Then we have 6 combinations of 2 variables: xy, xz, xw, yz, yw, zw. Now for each pair of variables we have  $3 \times 3 = 9$  possible fuzzy combinations like x-LOW and y-HIGH. So, we need altogether  $6 \times 9 = 54$  runs of FCM to exhaustively explore the possibility of the appropriate fuzzy association of 2 variables.]

- b) How can we consider association of 3 fuzzy variables, such as (a is A) & (b is B) & (c is C), where a, b, c are three fuzzy variables in a relation, and A, B and C are fuzzy sets like HIGH, MODERATE and LOW?

[**Hints:** Run FCM for 3 variable clusters. You need not run on all exhaustive sets of three variables, each having 3 possible fuzzy quantifiers. To reduce search complexity, first identify the clusters of 2 variables that take a large of the tuple population. Then attempt to construct new clusters by adding a third variable to each such good clusters of two variables.]

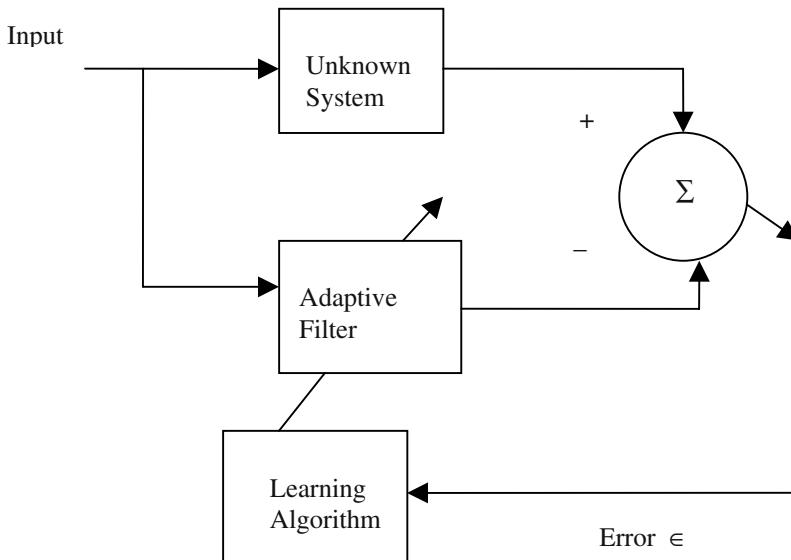
- c) Extend this principle for 4 or more variables. Show that determination of cluster center for n variables has always a lower complexity than the same for  $(n - 1)$  variables.
- d) Does variation of m (see Chapter 5) has any effect on the decision about the association of fuzzy variables?

## 23.12 Problem 6: Selected Items in Signal Processing and Communication Engineering

In this section we briefly outline some classical problems in systems theory, signal processing and communication engineering, and suggest a neural network solutions to these problems. MS students of electrical engineering will find these problems useful for their dissertation works. There is ample scope of research in these problems until this date.

### 23.12.1 System Modeling

In system modeling [20], [21], we are given an unknown physical system, and we need to model the system by an adaptive filter. For modeling, we submit a common input to both the unknown system and the adaptive filter, and based on the difference of their responses we take a corrective action of the adaptive system. If the parameters of the adaptive filter are adjusted in a way so that for all possible inputs the response of both the systems are identical, then we say that the adaptive system behaves like the physical system. The adaptive filter can be realized with a neural net. In Fig. 23.2 below, we present a scheme for system modeling by an adaptive filter.



**Fig. 23.2:** System modeling by an adaptive filter. The adaptive filter module in the present context is realized with a neural net.

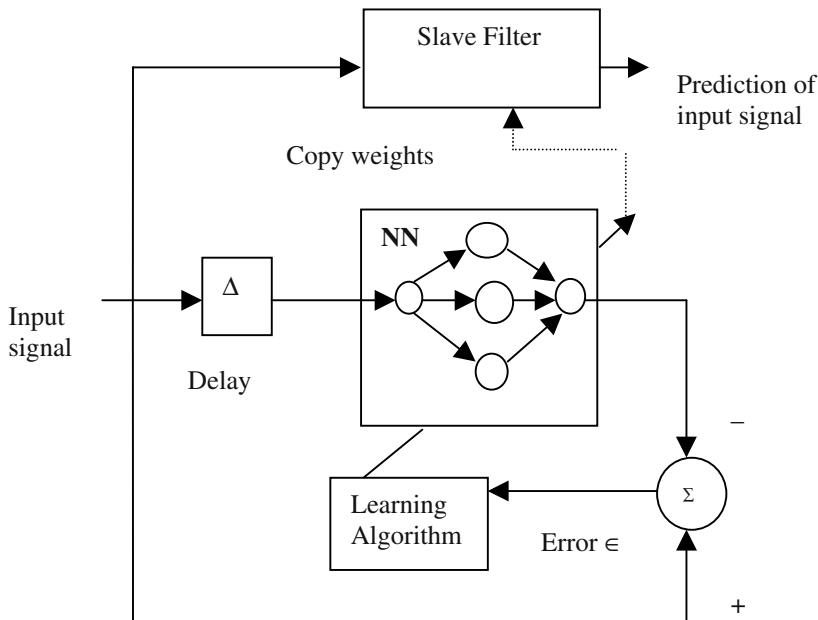
The following 2 problems are suggested.

- i) How can you use Widrow-Hoff's ADALINE neuron as an adaptive filter? Show how LMS can be employed as a learning algorithm?
- ii) How back-propagation algorithm can be employed in the proposed scheme? Verify the performance of the scheme by your own program.

**[Hints:** Replace the adaptive filter by a neural net of feed-forward topology, and the learning algorithm by the back-propagation algorithm.]

### 23.12.2 Non-linear Prediction Problem

In non-linear prediction, one can estimate the future values of a signal from its present and past samples. Fig. 23.3 presents one simple scheme of non-linear prediction. In this figure, the input signal, delayed by  $\Delta$  time units is fed to an adaptive filter, which in the present context is a neural net (NN). The undelayed input serves as the desired response for this filter. The difference of the NN response and undelayed input, called error is computed to adapt the weights of the NN by a learning algorithm. After the weights of the NN converge, they are copied into a slave filter (NN) for prediction of the input signal [21].



**Fig. 23.3:** One typical scheme of signal prediction.

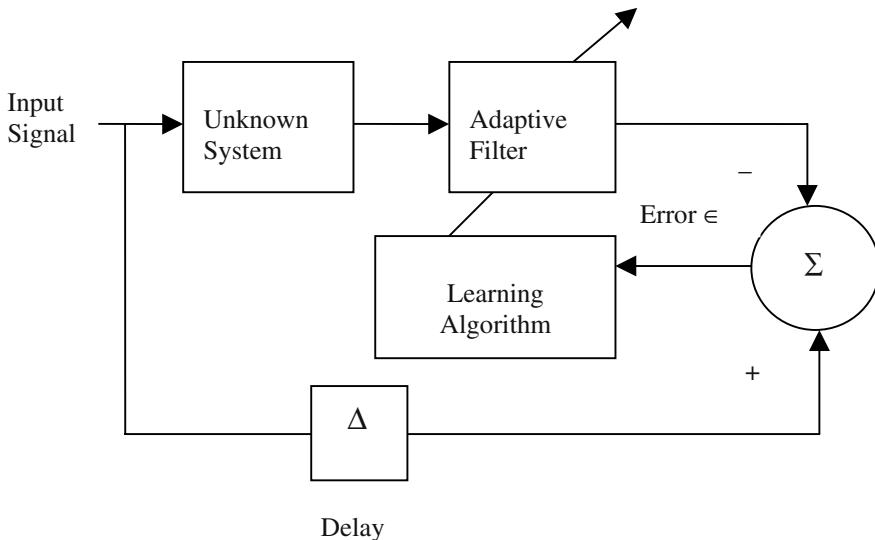
The following problems are suggested for MS theses.

- i) Verify the scope of all typical supervised learning algorithms such as Widrow-Hoff's ADALINE, BP- algorithm, RBF and others for the above application.

- ii) Which algorithm in your opinion will be the best choice for this application?  
Give experimental support to your answer.

### 23.12.3 Inverse Modeling

In many engineering applications, we occasionally require to determine the inverse [5] of an unknown system. Fig. 23.4 provides a scheme for determining the inverse transfer function of an unknown system. The unknown system and the adaptive filter are placed in cascade. The unknown system's input delayed by  $\Delta$  time units is the desired response of the adaptive filter. For simplicity, assume that  $\Delta$  is set to zero. Now, to ensure a small or zero error, the cascaded gain of the unknown system and the adaptive filter should be unity. In other words, the adaptive filter behaves like an inverse of the unknown system. If the response of the unknown system contains a delay or is non-minimum phase, allowing a nonzero delay  $\Delta$  will be highly advantageous. Including  $\Delta$  delays the inverse impulse response but yields a much lower mean-square error [19], [21].

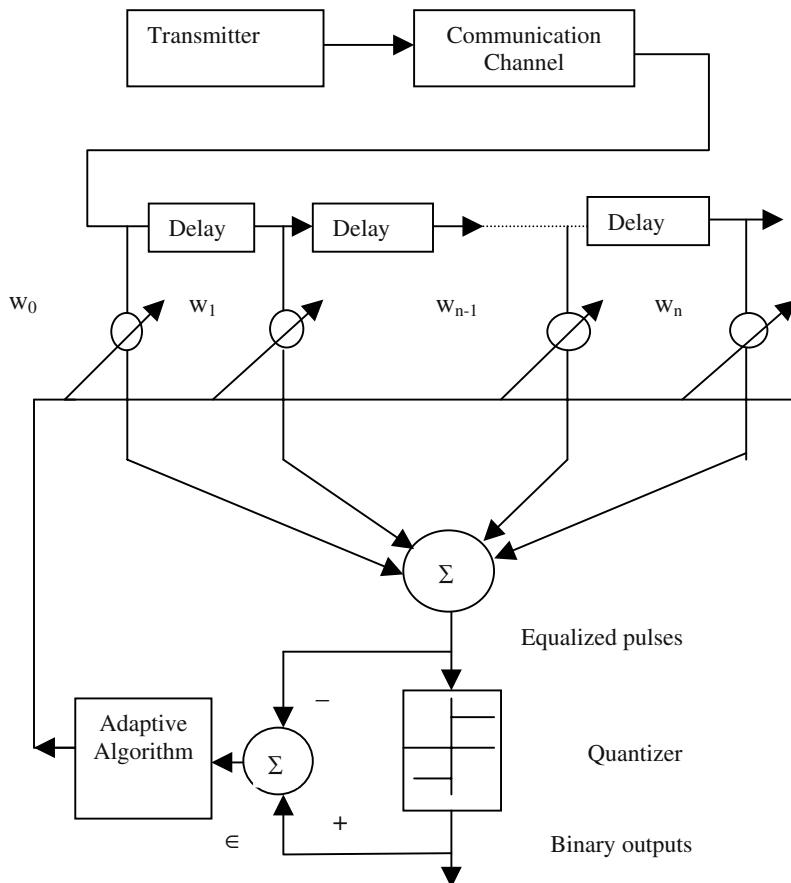


**Fig. 23.4:** The inverse model evaluation of an unknown system.

The following problems are suggested for undertaking graduate theses.

- i) Check how back-propagation learning algorithm/ Widrow-Hoff's LMS rule can be employed to handle the inverse computing problem.
- ii) Suppose the dynamics of an unknown system is given in transfer function form. Show by simulation that after the learning algorithm converges, the adaptive filter truly represents the inverse model of the given transfer function.

- iii) Given a D.C. armature controlled motor. Suppose you do not know its transfer function. How can you set up a laboratory experiment to determine the inverse of the motor transfer function?



**Fig. 23.5:** Adaptive channel equalization.

#### 23.12.4 Adaptive Channel Equalization

Sending digital data at high speed through communication channels often results in a phenomena called *intersymbol interference*, caused by signal pulse smearing in the dispersive medium. Equalization in data modems combats this phenomenon by filtering incoming signals. A modem's adaptive filter adapts itself to act as a channel inverse, and thus compensates for the irregularities in channel magnitude and phase response. Fig. 23.5 provides a scheme for adaptive channel equalization using an ADALINE neuron.

The following problems may be undertaken as part of MS dissertation works.

- i) Realize a program for implementation of the proposed scheme. Generate smeared pulses at the input of the ADALINE and count the number of learning cycles required to bring  $\epsilon$  close enough to zero.
- ii) Can you design any alternative mean square algorithm, other than LMS, to adapt the weights until convergence takes place?
- iii) Can you use a step function as a quantizer in the above figure? If yes, how will you use a perceptron learning algorithm to handle the problem?

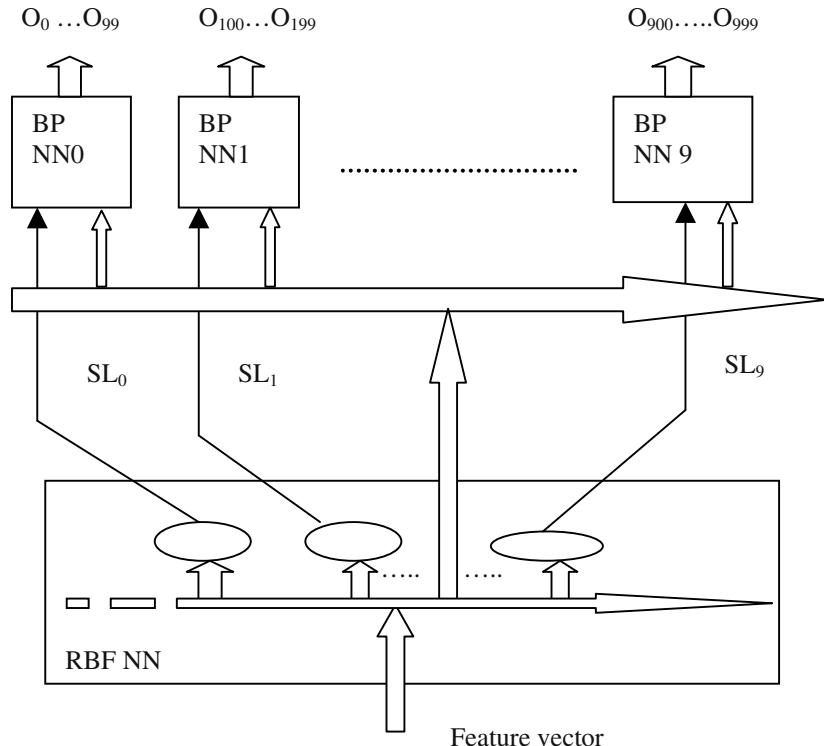
### **23.13 Problem 7: Handling Large Training Instances Using Back-propagation- RBF Synergism**

This section raises a practical problem in training neural nets with a large number of training instances. Modular back-propagation (BP) algorithm can solve the problem to some extent. We, however, take an alternative attempt to handle this problem by utilizing the composite benefits of modular back-propagation and RBF neural nets [2-3]. The example below justifies the importance of the composite use of BP- RBF synergism.

**Example 23.6:** Suppose we have as many as 1000 training instances. If we use a single BP network, training time will be too high to attain a satisfactory training accuracy. We can significantly reduce the training time by dividing the total number of instances into groups of 100 instances, and mapping each group on a separate network. Suppose, we use BP neural nets to train each group of 100 instances. Naturally, we require 10 such BP neural nets. There are two important problems that we need to address at this point. First, how should we select the members within a group? One way to select group members is to employ the principle of mapping similar instances on a common network.

The second problem is to design a scheme to automatically select one of 10 BP neural nets in the recognition phase. Readers may wonder: why such selection is needed? The selection, in fact, is needed because we do not want to remember which training instance is mapped onto which one of the 10 networks. The selection of the particular trained network that contains the desired training instance is triggered automatically by an RBF network.

An architecture of the overall scheme, containing an RBF neural net and 10 BP neural nets is presented below.



**Fig. 23.6:** Multiple BP-RBF synergism for pattern classification with a large number of training instances.

The ellipses in RBF NN (Fig. 23.6) denote RBF neurons. There are 10 RBF neurons each tuned to the cluster center of the patterns stored in one of the 10 BP NNs. For example, the leftmost ellipse representing the RBF neuron 0 is tuned to the cluster center of the patterns stored in  $BP\ NN_0$ . Consequently when a feature vector corresponding to one of the 100 patterns stored in  $BP\ NN_0$  appears at the input of the RBF NN, it fires and triggers the  $BP\ NN_0$ . The  $BP\ NN_0$  then sets one of its 100 output lines high depending on the supplied feature vector.

Graduate students are advised to study the above problem with special attention to computing the following system parameters.

- i) Determine a strategy to group patterns.

[**Hints:** Identify 2/3 most important features of the patterns and use Euclidean norm to test their similarity with respect to the selected features.]

- ii) Determine the mean vector and the variance vector for each group of 100 patterns.

[**Hints:** For each attribute of the training instances, we obtain a mean and a variance. Thus taking all attributes together, we obtain a mean vector and a variance vector.]

- iii) How many RBF neurons will fire at a time?

[**Hints:** Only one RBF neuron, whose cluster center matches with a input instance, fires.]

- iv) There is no second layer of the RBF neural net. Is it not needed?

[**Hints:** Since the first layer performs the clustering, we do not require the second layer.]

- v) Construct a program for the scheme outlined in Fig. 23.6, and determine its total training time and RMS error-margin for the worst known training sample.

[**Hints:** RMS error margin is defined as the Euclidean norm of the target vector and the computed vector with reference to the neurons at the last layer of the BP NN. The worst RMS error margin means the largest Euclidean norm, when the norm is computed for all training instances.]

## **23.14 Problem 8: Macro Cell Placement and Routing in VLSI Design Using Genetic Algorithms**

Macro-cell placement and routing are two important problems in VLSI layout design [13]. In macro-cell placement problem, one has to pack a large number of rectangular cells within a minimum base area. Given the specification list of pin connections of the rectangular cells, the routing problem is concerned with determining the shortest routes of wiring needed to connect the pins. There exist interesting algorithms for both placement and routing. Most of the existing literature on placement and routing execute placement before routing. Much computational time, however, can be saved if a pre-plan for routing is undertaken with placement simultaneously. In this section, we present a scheme

for placement and routing that prepares a pre-plan for routing along with placement. The final routing is done at a later stage.

The placement and routing problem discussed above is re-stated below in a formal way.

- 1) Consider a set of rectangular modules of some given dimensions. Now, compute a placement configuration of these modules on a plane satisfying the following Constraints:
  - i) There should be no overlapping of the modules.
  - ii) The rectangular area circumscribing the placement should be of minimum area.
- 2) Now, let us assume that each of the modules have multiple terminals. (They are basically macro-cells used in VLSI design work). There is a list specifying which terminals are to be interconnected. (Say terminal 1 of module 1, terminal 2 of module 5, and terminal 5 of module 10 are to be interconnected). Corresponding to each specification, try to satisfy the interconnection with minimum wire length and also provide some alternative connection schemes if the best solution is not practically realizable.

Can you suggest a heuristic, which can be employed to realize problems 1 and 2 simultaneously, at least in an approximate manner?

### **23.15 A Possible Solution to Macro Cell Placement and Routing**

Let us consider a Genetic Algorithm based solution approach in which the chromosomes are coded in a way such that each represents a placement configuration. Here each of the genes represent the co-ordinate of the center of gravity of a module (cell) with given dimension and orientation. Let  $A$  denote the area circumscribing the placement given by a chromosome. The total interconnect length of a given placement is estimated below.

Let  $N$  denote the no. of nets. Let  $n(k)$  denote total no. of terminals of the  $k$ -th net. Let  $t(k_i)=[x(k_i),y(k_i)]$  denote the position of terminal ' $i$ ' in net  $k$ . The center of gravity of the  $k$  th net is defined as

$$T(k)=\left[ \sum_{i=1}^{n(k)} t(k_i) \right] / n(k).$$

(Each net specifies the terminals to be connected together.)

Then the estimated total interconnect length

$$L = \sum_{k=1}^{N_n(k)} \left( \sum_{i=1}^n \|t(k_i) - T(k)\| \right).$$

where  $\|x\|$  denotes the usual Euclidean vector norm. The fitness function in the present context is given by  $1/[x_1 * A + x_2 * L]$  where  $x_1 + x_2 = 1$  and  $x_1$  &  $x_2$  denote Lagrange's multipliers.

Minimizing the above function we can get a satisfactory placement with approximate routing by proper tuning of  $x_1$  and  $x_2$ .

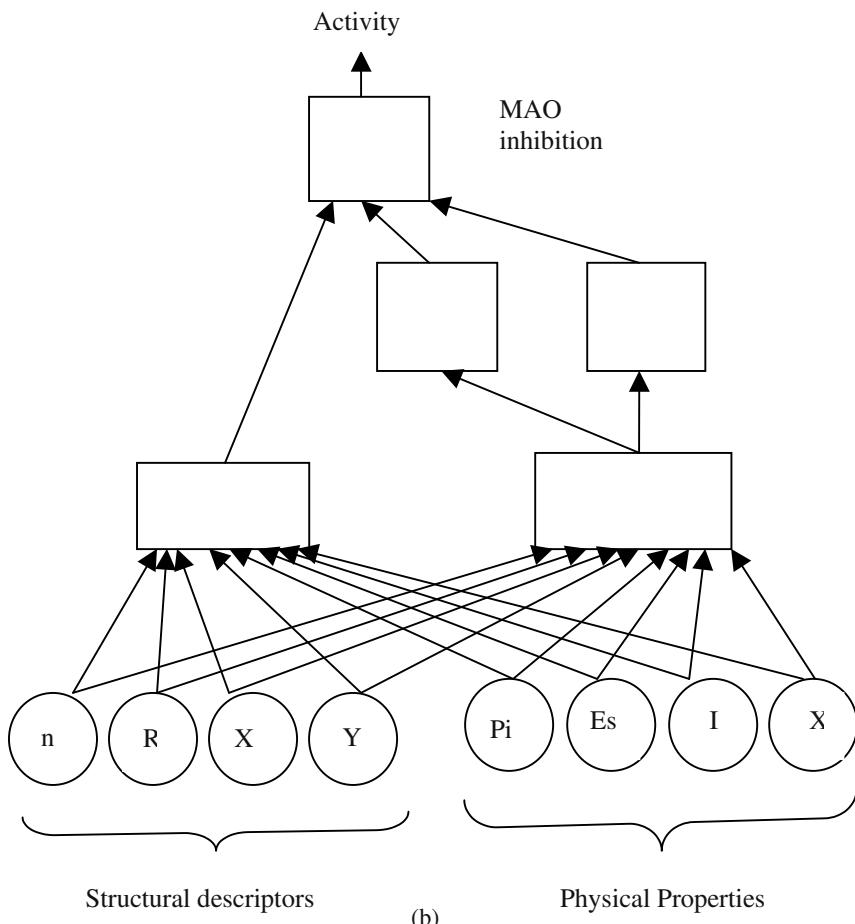
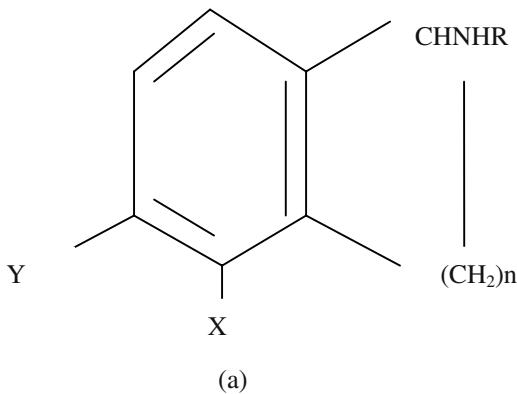
In case of detailed routing (exact solution of problem 2), we need to solve the Steiner problem in graph. One can try many heuristics in these regard. A standard algorithm is the DNH (Distance Network Heuristic). It can be combined with a GA based approach. But DNH generates a single solution. Modifying the Distance graph at the input of DNH can provide with many alternative solutions. But to generate many distance graphs one needs to have some multiple path finding algorithm on graphs. So try and construct one.

### 23.16 Problem 9: Drug Classification Problem Using Artificial Neural Networks

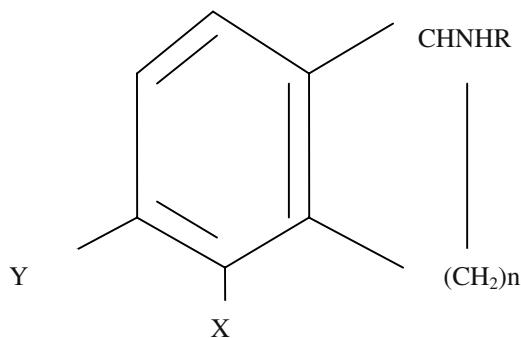
Martin and co-workers (see in [6]) determined structure-activity relationship in a series of aminoindans and aminotetralines to monoamine oxidase (MAO) inhibition. Each compound is described by 4 physical properties (numerical features [11]) and the biological activity of MAO inhibition both in vitro and vivo, as given below:

- P<sub>i</sub>= no. of particular kind of groups,
- E<sub>s</sub>= no. of particular kinds of derivatives,
- I= largest chain of non-aromatic carbonals,
- X= combinations of the above features.
- Output= activity low/ high.

Besides the physical characteristics, the structural characteristics of MAO inhibitors are given in Fig. 23. 7(a). The neural net given in Fig. 23.7 (b) may be used to train compounds of similar class. The advantage of using neural net is that it integrates features of different types in most effective way. Samples of high and low MAO inhibitors are also given in Fig. 23.8 and 23.9 respectively. The problem is to classify the MAOs into LOW/ HIGH inhibitors.



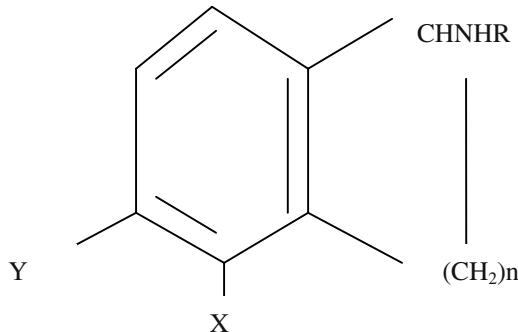
**Fig. 23.7:** (a) The general structure of MAO and (b) its classification by neural net.



Y = OCH<sub>3</sub> or OH; X= H.

R: No. of Carbon <=1, No. of Hydrogen <=3, No. of Oxygen =0.

**Fig. 23.8:** Structural pattern for high MAO inhibition activity.



R: No. of Carbon >1, No. of Hydrogen > 3, No. of Oxygen = 0.

**Fig. 23.9:** The structural pattern for low MAO inhibition activity.

Graduates students interested in Bioinformatics may undertake the following problems.

- i) Select from any standard book of Biochemistry [11] a set of organic structures having low/ high MAO inhibition.

- ii) Construct a table of training instances to classify the list into HIGH/LOW inhibition activity.
- iii) Train the proposed network with the instances you constructed.
- iv) Submit a new application instance, and check whether the trained network can correctly classify it into LOW/HIGH MAO inhibition activity.

## References

- [1] Adriaans, P. and Zantinge, D., *Data Mining*, Addison-Wesley, Singapore, 1999.
- [2] Biswas, B. and Konar, A., "Speaker identification from voice using neural networks," *J. of Scientific & Industrial Research*, vol. 61, pp. 599-606, 2002.
- [3] Biswas, B., *Fuzzy Decision Support System for Criminal Investigation*, Ph.D. Thesis, Jadavpur University, 2001.
- [4] Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P. and Uthurusamy, R. (Eds.), *Advances in Knowledge Discovery and Data Mining*, AAAI Press/MIT Press, Cambridge, 1996.
- [5] Fiesler, E. and Beale, R. (Eds.), *Handbook of Neural Computation, Part G: Neural Network in Practice*, IOP Publishing and Oxford University Press, 1997.
- [6] Fu, L. M., *Neural Networks in Computer Intelligence*, McGraw-Hill, Singapore, 1994.
- [7] Jain, L. C., Halici, U., Hayashi, L., Lee, S. B. and Tsutsui, S., (Eds.), Intelligent Biometric Techniques in Fingerprint and Face Recognition, CRC Press, Boca Raton, Florida, 1999.
- [8] Klir, G. J. and Yuan, B., *Fuzzy Sets and Fuzzy Logic: Theory and Applications*, Prentice-Hall, NJ, pp. 418-441, 1995.
- [9] Konar, A. and Pal, S., "Modeling Cognition with Fuzzy Neural Nets," In *Fuzzy Theory Systems: Techniques and Applications*, Leondes, C. T. (Ed.), Academic Press, pp. 1341-1391, 1999.

- [10] Krishnapuram, R. K., Segmentation, In *Handbook of Fuzzy Computation*, Ruspini, E. H., Bonissone, P. P. and Pedrycz, W. (Eds.), IOP Publishing, 1998.
- [11] Lehninger, A. L., *Biochemistry: The Molecular Basis of Cell Structure and Function*, Worth Publishers, Inc. NY, pp. 67-108, 1970.
- [12] Lesk, A. M., *Introduction to Bioinformatics*, Oxford University Press, NY, pp. 242-246, 2002,
- [13] Majumder, P. and Tudnick, E. M., *Genetic Algorithms for VLSI Design, Layout and Test Automation*, Prentice-Hall PTR, NJ, 1999.
- [14] Pal, S., *Cognitive Modeling Using Fuzzy Logic Network*, Ph. D. Thesis, Jadavpur University, 2002.
- [15] Saha, P. and Konar, A., “A heuristic algorithm for computing the max-min inverse fuzzy relation,” *Int. J. of Approximate Reasoning*, Elsevier, vol. 30, pp. 131-147, 2002.
- [16] Saha, P., *Abductive Reasoning with Inverse Discrete Fuzzy Relations*, Ph.D. Thesis, Jadavpur University, 2001.
- [17] Saha, P. and Konar, A., “Reciprocity and Duality in a Fuzzy Petri Net,” *Int. J. of Modeling and Simulation*, vol. 24, no. 3, 2004.
- [18] Silberschatz, A., Korth, H. F. and Sudarshan, S., *Database System Concepts*, McGraw-Hill, Singapore, pp. 702-710, 1997.
- [19] Solo, V. and Kong, X., *Adaptive Signal Processing Algorithms: Stability and Performance*, Prentice-Hall, NJ, 1975.
- [20] Widrow, B. and Winter, R., “Neural nets for adaptive filtering and adaptive pattern recognition,” *IEEE Computer, Special Issue on Artificial Neural Systems*, pp. 25-39, March 1988.
- [21] Widrow, B. and Stearns, S. D., *Adaptive Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, pp. 193- 404, 1985.
- [22] Yuhas, B. P., Goldstein, M. H., Sejnowski, T. J. and Jenkins, R. E., “Neural network models of sensory integration for improved vowel recognition,” *Proc. IEEE*, vol. 78, no. 10, Oct. pp. 1658-1668, 1990. Also appeared in *Neural Networks: Theoretical Foundations and Analysis*, Lau, C. (Ed.), IEEE Press, pp.311-320, 1992.

# A

## Appendix A: Sample Run of Programs Included in the CD

*Appendix A includes sample run of a few programs developed to illustrate the case studies. The programs were built in C/C++ and Pascal languages. Source codes and .exe files of these programs are included in the CD, placed inside the back cover of the book. Before executing the .exe files, save them in appropriate drive (A or C) as mentioned below. Please be sure to include the associated data files or the .bmp files in the same folder of the appropriate drive before to execute a program. The associated file-names are given in the program description below. The source codes can also be compiled by Microsoft Visual C++ 6.0 Compiler and Turbo Pascal 6.0 compiler and executed on a Pentium-based machine.*

### **A.1 Detection of Mouth Opening of a Person from his/her Facial Image**

The following 4 programs and the image file pic.bmp should be loaded in a floppy before proceeding further.

**a) med.c**

This program performs *median filtering* over an input bitmap file: pic.bmp and generates an output file: med\_fil.bmp. The input file contains an image of (106 × 80). The program 'med.c' results in a reduced image of (103 × 77) pixels and removes noise from the source file to an appreciable extent.

**Sample Run:**

Picture conversion successful.

The input file is pic.bmp

The output file is med\_fil.bmp

Press any key to continue

**Note:** The output file med\_fil.bmp will be created in the same folder of the C drive, which contains the input file and the program. Double click the file icon to view the output image. You need a standard image viewing software such as Corel, Adobe, Microsoft Photo-Editor, Paint etc. to see the input and the output image files.

On double clicking the icon pic.bmp, the following image appears.



On double clicking the icon med\_fil.bmp, the following image appears. This image is obtained as the result of execution of the median filter program on the original image given above.

**b. pix\_xy-g.c**

This program reads the image file ‘median.bmp’ and writes the pixel features (gray scale or any suitable transform of gray scale) in the file ‘pix\_dat’. The initial random seeds to be used as initial membership values for FCM classification are further generated and written in the file ‘init\_file’.

**Note:** The output files viz. Pix\_dat and init\_file are text files, created in the same folder of the C drive as the input files and the programs. They can be opened using standard text editors such as Notepad, Wordpad, MS Word etc.

**Sample Run:**

```
input image is med_fil.bmp.  
two files named pix_dat and init_file generated successfully  
Width of the image is 103  
Height of the image is 77  
Press any key to continue
```

**c. c\_means.c**

This program performs image segmentation by *fuzzy c-means clustering algorithm*. It requires two input files: pix\_dat and init\_file and generates an output file: cluster.bmp. The file pix\_dat contains pixel information such as  $l^*$ ,  $a^*$  and  $b^*$  values, and init\_file contains initial membership of the pixels to fall in the lip and the non-lip class. The ‘cluster.bmp’ is the output file containing the segmented mouth region.

**Note:** As before, go to the relevant folder of the C drive, double-click the cluster.bmp icon, and the image will open in your default image-viewing software.

**Sample Run:**

Input image file is med\_fil.bmp  
Input data files are pix\_dat and init\_file  
Output file is cluster.bmp

count=1

cluster center  
84.534337 14.203845  
84.398015 14.208842

count=2

cluster center  
87.322484 12.325246  
77.994048 18.400881

count=3

cluster center  
88.269858 11.376430  
72.493847 23.014281

count=4

cluster center  
87.902160 11.636955  
70.388852 24.610468

count=5

cluster center  
87.830730 11.701265  
69.985324 24.861082

count=6

cluster center  
87.800712 11.738587  
69.787845 24.940607

count=7

cluster center  
87.792349 11.744961  
69.750012 24.967565

count=8

cluster center  
87.790039 11.746053  
69.742367 24.975803

count=9

cluster center  
87.789401 11.746364  
69.740224 24.978059

count=10

cluster center  
87.789222 11.746452  
69.739617 24.978692

Press any key to continue

On double clicking the icon cluster.bmp, we obtain the mouth region by fuzzy-C-means clustering algorithm.



#### d. intent.c

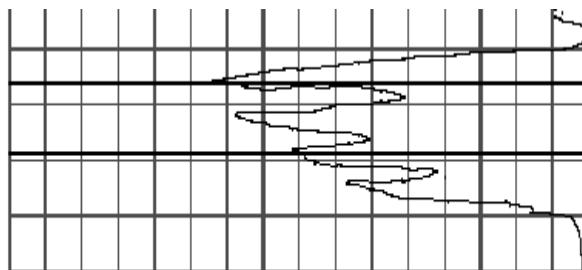
This program calculates the peak mouth opening. The input image file is cluster.bmp and the output file generated by the program is intensity.bmp.

The program intent.c determines the average intensity in each row of the segmented mouth region and plots it against the height of the image.

**Sample Run:**

```
Input file is cluster.bmp  
Output file is intensity.bmp  
THE PEAK VERTICAL OPENING OF THE INNER MOUTH CAVITY->  
17  
The intensity data of the picture has been extracted.  
Press any key to continue
```

The row-wise average intensity profile of the FCM-clustered image is obtained below by double clicking the icon intensity.bmp.



## A.2 A Program for Image Matching Using Fuzzy Moment Descriptors

The Imagemat program, written in C, is used to match a reference image with a given set of images and the best matched image has to be identified. It is designed based on the concept of fuzzy moment descriptors in chapter 23. You can submit image files boy1.dat, boymus.dat (boy with moustache) and boyglass.dat (boy with spectacles) included in the floppy diskette. Note that one file must be the reference image. The other images with which the reference is to be matched should include the copy of the reference image.

**NOTE:** Keep the data files boy1.dat, boymus.dat and boyglass.dat along with the program Imagemat in a floppy before execution.

The sample run of the program is presented below.

no. of pictures to compare = 3

no. of subblocks in each picture

(make sure you are giving perfect square) = 16

enter name of file in which d.image of 1 picture available = boy1.dat

enter number of rows or columns in image = 64

taking the gray values (in rowwise) form from file

calculating gradient matrix

dividing the picture into subblocks

number of subblocks = 16

calculating the statistical parameters

calculating membership values

enter name of file in which d.image of 2 picture available = boy1.dat

enter number of rows or columns in image = 64

taking the gray values (in rowwise) form from file

calculating gradient matrix

dividing the picture into subblocks

number of subblocks = 16

calculating the statistical parameters

calculating membership values

enter name of file in which d.image of 3 picture available = boymus.dat

enter number of rows or columns in image = 64

taking the gray values (in rowwise) form from file

calculating gradient matrix

dividing the picture into subblocks

number of subblocks = 16

calculating the statistical parameters  
calculating membership values

The best matched picture is in the position = 2

The best matched picture is in file boy1.dat

The distance between best matched picture boy1.dat and reference picture boy1.dat is 0.000000

### A.3 GA in Path Planning of a Mobile Robot

The GaPath program employs Genetic Algorithm to plan a trajectory between a given starting point and a goal point.

**NOTE:** Save the program and the BGI file on a floppy drive (A:) before execution of the program.

The sample run of the program for two special cases is given below.

#### When the goal is within convex obstacle:

The workspace coordinates are (50,50) and (450,450)

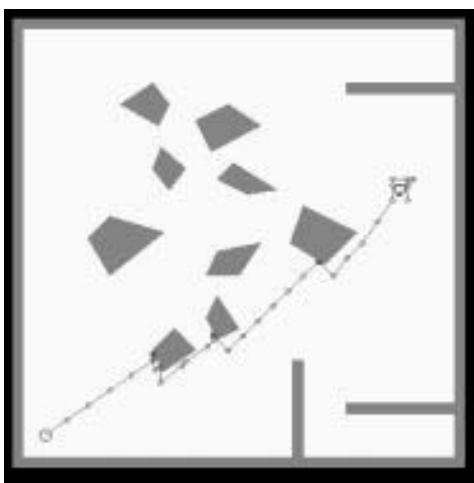
Enter starting x (between 50 & 450): 70 (press enter)

Enter starting y (between 50 & 450): 70 (press enter)

Enter sensing range (between 10 & 50): 30 (press enter)

Enter robot step size (5 to 30) : 20 (press enter)

Distance covered = 604.238 units



Time taken to search path: 0.549451

**When the goal is within concave obstacle:**

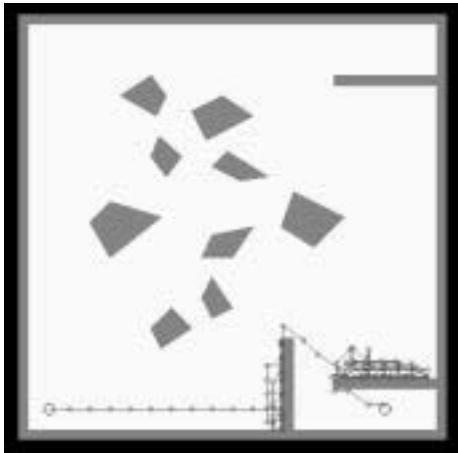
The workspace coordinates are (50,50) and (450,450)

Enter starting x(between 50 & 450): 70 (press enter)

Enter starting y(between 50 & 450): 70 (press enter)

Enter sensing range (between 10 & 50): 30 (press enter)

Enter robot step size (5 to 30) : 20 (press enter)



Time taken to search path: 1.868

Distance covered = 1710.915units

#### A.4 The Fpnreas1 Program

The Fpnreas1 program is a natural language interface program that inputs the database and knowledge base for the criminal investigation problem. The knowledge base has two modules: the default rules and the production rules. The user has to submit both types of rules in a specific menu driven format. This is a Pascal program, which also orders the rules, if they are pipelined (i.e., consequent of one rule is an antecedent of another rule).

**NOTE:** SAVE THREE FILES fpnreas1, fpnreas2 and fpnreas3 TOGETHER IN A FLOPPY (A -Drive) before execution of the programs. The order of execution should be fpnreas1, fpnreas2 and fpnreas3 respectively.

The sample run of the program Fpnreas1 is presented below.

B:\>fpnreas1

KEEP CAPS LOCK ON  
ARE YOU READY (Y/N)? Y

ENTER generic filename without extension KONAR  
MAIN MENU:

- 1- Create / Append static database
- 2- Create / Append production rules
- 3- Create / Append default rules
- 4- Arrange production rules properly
- 5- Delete clauses from database
- 6- Delete clauses from pr rule base
- 7- Delete clauses from def. rule base
- 0- Terminate

SELECT:

1  
C to create, A to append: C  
enter predicate and arguments  
one by one for each clause

if some argument is inapplicable or redundant in the context  
enter "END" as predicate to terminate

press any key to continue

Predicate: L  
Argument1: S  
Argument2: R

Predicate: L  
Argument1: S  
Argument2: M

Predicate: LSR  
Argument1: S  
Argument2: \_

Predicate: S  
Argument1: S  
Argument2: \_

Predicate: END

**MAIN MENU:**

- 1- Create / Append static database
- 2- Create / Append production rules
- 3- Create / Append default rules
- 4- Arrange production rules properly
- 5- Delete clauses from database
- 6- Delete clauses from pr rule base
- 7- Delete clauses from def. rule base
- 0- Terminate

**SELECT:**

2

general structure of a production rule:

antecedents without complementation

antecedents1: predicate, argument1, argument2

antecedents2: predicate, argument1, argument2

antecedents3: predicate, argument1, argument2

antecedents4: predicate, argument1, argument2

antecedents5: predicate, argument1, argument2

antecedents with complementation

antecedents1: predicate, argument1, argument2

antecedents2: predicate, argument1, argument2

antecedents3: predicate, argument1, argument2

antecedents4: predicate, argument1, argument2

antecedents5: predicate, argument1, argument2

conclusion

predicate, argument1, argument2

Threshold value

press any key to continue

further instructions

each rule must contain

atmost 5 antecedents without complementation, and

atmost 5 antecedents with complementation

if the actual no. of antecedents is less than 5.

Enter "END" as predicate

When all the antecedents of the rule are entered

use upper case letters to represent arguments

if some argument is inapplicable or redundant in the context

enter it as an underscore

---

press any key to continue  
antecedents without complementation:  
ANTECEDENT 1: Predicate: L  
, ARGUMENT1 : X  
, ARGUMENT2 : Y

ANTECEDENT 2: Predicate: L  
, ARGUMENT1 : X  
, ARGUMENT2 : Z

ANTECEDENT 3: Predicate: END

antecedents with complementation:  
ANTECEDENT 1: Predicate: END

conclusion  
Predicate: LSR  
Argument1 : X  
, Argument2: \_

Threshold value: 0.7

More rules to create (y / n)? Y

antecedents without complementation:  
ANTECEDENT 1: Predicate: LSR  
, ARGUMENT1 : X  
, ARGUMENT2 : \_

ANTECEDENT 2: Predicate: END

antecedents with complementation:  
ANTECEDENT 1: Predicate: END

conclusion  
Predicate: S  
Argument1 : X  
, Argument2: \_

Threshold value: 0.7

More rules to create (y / n)? N

**MAIN MENU:**

- 1- Create / Append static database
- 2- Create / Append production rules
- 3- Create / Append default rules
- 4- Arrange production rules properly
- 5- Delete clauses from database
- 6- Delete clauses from pr rule base
- 7- Delete clauses from def. rule base
- 0- Terminate

**SELECT:**

3

C to create, A to append: C

General structure of a production rule:

antecedents without complementation

antecedents1: predicate, argument1, argument2

antecedents2: predicate, argument1, argument2

antecedents3: predicate, argument1, argument2

antecedents4: predicate, argument1, argument2

antecedents5: predicate, argument1, argument2

antecedents with complementation

antecedents1: predicate, argument1, argument2

antecedents2: predicate, argument1, argument2

antecedents3: predicate, argument1, argument2

antecedents4: predicate, argument1, argument2

antecedents5: predicate, argument1, argument2

conclusion

predicate: SUSPECT, argument1, argument2

press any key to continue

further instructions

each rule must contain

atmost 5 antecedents without complementation, and

atmost 5 antecedents with complementation

if the actual no. of antecedents is less than 5.

Enter "END" as predicate

When all the antecedents of the rule are entered

use upper case letters to represent arguments

if some argument is inapplicable or redundant in the context

enter it as an underscore

---

press any key to continue  
antecedents without complementation:  
ANTECEDENT 1: Predicate: L  
, ARGUMENT1 : X  
, ARGUMENT2 : Z  
  
ANTECEDENT 2: Predicate: L  
, ARGUMENT1 : X  
, ARGUMENT2 : Y

ANTECEDENT 3: Predicate: END

antecedents with complementation:  
ANTECEDENT 1: Predicate: END

conclusion  
Predicate: SUSPECT Argument1: S  
Argument2: \_

More rules to create (y / n)?N

MAIN MENU:

- 8- Create / Append static database
- 9- Create / Append production rules
- 10- Create / Append default rules
- 11- Arrange production rules properly
- 12- Delete clauses from database
- 13- Delete clauses from pr rule base
- 14- Delete clauses from def. rule base
- 1- Terminate

SELECT:

4

Rules properly arranged

MAIN MENU:

- 15- Create / Append static database
  - 16- Create / Append production rules
  - 17- Create / Append default rules
  - 18- Arrange production rules properly
  - 19- Delete clauses from database
  - 20- Delete clauses from pr rule base
  - 21- Delete clauses from def. rule base
  - 2- Terminate
- SELECT:
- 0

## A.5 The Fpnreas2 Program

It is also a natural language interface program, developed in Pascal, which asks for the name of persons involved in the criminology problem. The variables in the predicates of the production rules are then instantiated with the supplied values and the truth–falsehood (T / F) of the resulting predicates is asked. Thus from a number of instantiated predicates, the program can filter a few, which subsequently results in the rules with properly bound variables in the predicates. This is done so to realize unification in an indirect manner. Note that Pascal does not possess the instructions for unification, so one has to use its power of string matching.

The sample run of the program is presented below.

B:\> fpnreas2

KEEP CAPS LOCK ON. ARE YOU READY  
Y

ENTER generic filename without extension KONAR

QUERY SESSION:

enter names of persons involved

enter "END" to terminate

Person1: S

Person2: R

Person3: M

Person4: END

enter T if true, F if false

L(S,S): F

L(S,R): T

L(S,M): T

L(R,S): F

L(R,R): F

L(R,M): F

L(M,S): F

L(M,R): F

L(M,M): F

suspects selected by DRs are:

S

R

M

Enter the name of the conclusion Nodes

Enter "END" to terminate

S

END

## A.6 The Fpnreas3 Program

This also is a Pascal program that computes the belief for all predicates located at the places of the FPN. The belief revision cycles are continued, until the steady state is reached, following which there will be no more changes in beliefs at a place. The program can also construct an FPN from the rules and data supplied in Fpnreas1 program. The program finally determines the culprit place from many possible suspects and gives an explanation for the selection of the culprit.

The sample run of the program is presented below.

B:\> fpnreas3

net to display (y / n)? Y

Nodes displayed in the following fashion

Level	POSITION_FROM_LEFT	NODE
1	1	H: This is
nethead ( _, _)		
2	1	I: L (S,R)
3	1	T: LSR (S,_)
4	1	N: LSR(S,_)
5	1	T: S (S,_)
6	1	N: S (S,_)
7	1	F: S (S,_)
3	2	T: LSR (S,_)
2	2	I: L(S, M)
3	3	T: LSR (S,_)
3	4	T: LSR(S,_)

palette:2

Here it draws the FPN, the top part of that network we considered in case study in chapter 23. The figure is not good. Readers may try to improve it.

Initialization Session:

which model to use (2 OR 3): 2

I: L (S, R): Initial FTT: 0.4

I: L (S, M): Initial FTT: 0.1

N: LSR (S, \_): Initial FTT: 0.4

N: S (S, \_): Initial FTT: 0.3

F: S (S, \_): Initial FTT: 0.3

No Of updations: 10

Stability obtained after 2 updations

CONCLUSION:

conclusion no.1: S (S,\_): FFT= 4.0000000000E -01

Conclusion1:

paths

path no: 1

path gain: 0.400

I: L (S, R): FTT: 0.400

T: LSR (S, \_): FTT: 0.400

N: LSR (S, \_): FTT: 0.400

T: S (S, \_): FTT: 0.400

N: S (S, \_): FTT: 0.400

F: S (S, \_): Initial FTT: 0.400

node type: F

Predicate :S

Argument1: S

Argument2: \_

Node found

Choose palette no: 2

Here it draws the belief distribution of the place for S (s). This part is not at all good, and I request the readers to spend some time in improving the graphics part of the program.

## A.7 Fuzzy Chaos Program

f\_chaos.c is a C program of fuzzy chaos.

To run this program copy the bgi folder in floppy (drive A:). Then run the exe file named f\_chaos.exe.

One sample run is given below.

1.  $p(n+1) = (1-p(n))(1-q(n)) * (1-q(n)) \&$   
 $q(n+1) = (1-q(n))$
2.  $p(n+1) = p(n) \&$   
 $q(n+1) = \max(\min(p(n), q(n)), \min((1-p(n)), (1-q(n))))$
3.  $p(n+1) = a11 * p(n) + a22 * (1-q(n)) * (1-p(n)) * (1-q(n)) \&$   
 $q(n+1) = (1-p(n)) * (1-q(n))$
4.  $p(n+1) = p(n) \&$   
 $q(n+1) = p(n) * (1-p(n)) * (1-q(n))$
5.  $p(n+1) = p(n) \&$   
 $q(n+1) = q(n)$

Press the index of the required function:

1

Enter the starting values of p & q :

0 .2

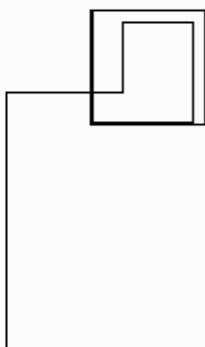
0 .6

Now press the Enter key. The program will plot the curve of function p in the following method. The points given below will be joined sequentially by straight line:

(p(0), 0.0),  
 (p(0),p(1)),  
 (p(1),p(1)),  
 (p(1),p(2)),  
 (p(2),p(2)),  
 (p(2),p(3)),  
 (p(3),p(3)),  
 (p(3),p(4)) and so on.

After the plot for p is over press enter key for q plot. The plot will be drawn in the above manner.

**p[0]=0.200000 & q[0]=0.600000**



**----For p[t]----**

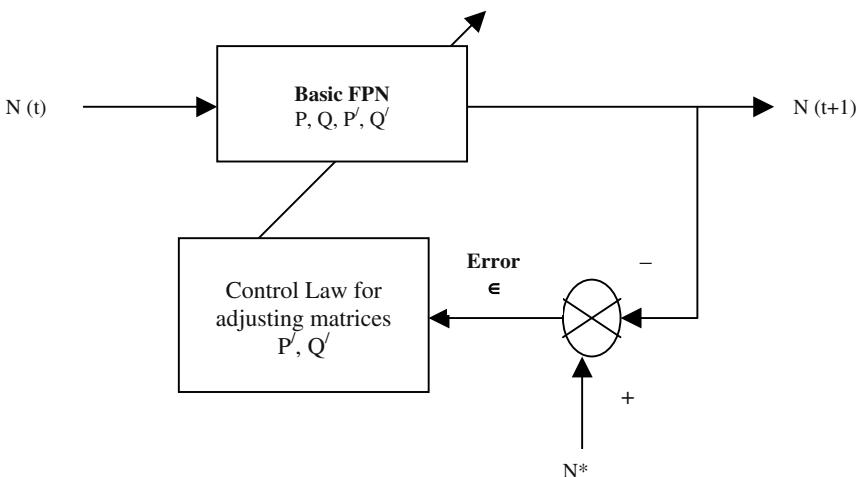
**p[0]=0.200000 & q[0]=0.600000**



**----For q[t]----**

## A.8 The New\_Petri3 Program to Study Controllability on FPN

This program will accept an initial state vector  $N(0)$  and a target vector  $N(t^*) = N^*$  as inputs and will incorporate two fuzzy matrices  $P'(t)$  and  $Q'(t)$  in the basic belief updating equation of an FPN to control the initially supplied state vector as described in section 23.4.2 of the book. The basic  $P$  and  $Q$  matrices have to be supplied by the user and the program will automatically initialize the  $P'(0)$  matrix to all zero and  $Q'(0)$  matrix to all one form. It will then go on adjusting the values of elements of  $P'$  and  $Q'$  matrices as per the following scheme:



**Fig. A.1:** Scheme for implementing the Control Law in FPN through fuzzy  $P'$  and  $Q'$  matrices

The executable file should be copied to the hard disk. Then on typing C:\> new\_petr3, the program will ask for inputs:  $N(0)$ - initial belief vector,  $N^*(t^*)$ -target vector and basic  $P$  and  $Q$  matrices. It will then go on adjusting  $P'$  and  $Q'$  matrices as shown in the following runs. Usually, the error defined as the Euclidean norm  $\|N^* - N(t+1)\|$  reduces significantly to zero after 10 to 12 iterations. Results for the first 5 iterations are given below.

\*\*\*\*\* Run 1 \*\*\*\*\*

Test Data

Your P matrix

1.00	0.00	0.00
0.00	1.00	0.00
0.00	0.00	1.00

---

Your Q matrix

0.00 1.00 1.00  
1.00 0.00 1.00  
1.00 1.00 0.00

Your Belief Vector

0.20  
0.30  
0.40

Your P' matrix

0.00 0.00 0.00  
0.00 0.00 0.00  
0.00 0.00 0.00

Your Q' matrix

1.00 1.00 1.00  
1.00 1.00 1.00  
1.00 1.00 1.00

Target Vector

0.400000  
0.500000  
0.600000

Belief Vector after iteration 1

0.000000  
0.000000  
0.000000

Error after iteration 1

0.40  
0.50  
0.60

Belief Vector after iteration 2

0.182400  
0.228000  
0.273600

Error after iteration 2

0.22  
0.27  
0.33

Belief Vector after iteration 3

0.281626  
0.352032  
0.422438

Error after iteration 3

0.12  
0.15  
0.18

Belief Vector after iteration 4

0.335604  
0.419505  
0.503407

Error after iteration 4

0.06  
0.08  
0.10

Belief Vector after iteration 5

0.364969  
0.456211  
0.547453

Error after iteration 5

0.04  
0.04  
0.05

# B

## Appendix B: Evolutionary Algorithms of Current Interest

*Evolutionary computing has already been introduced in chapter 12 with a special emphasis on GA and GP. Appendix B<sup>1</sup> provides an overview of two promising evolutionary algorithms, which are currently gaining popularity for their increasing applications in solving engineering problems of diverse domains. One of these algorithms, known as Ant Colony Optimization mimics the behavior of group of real ants in multi-agent cooperative search problems. The latter one called Differential Evolution is a deviant variety of genetic algorithm, which attempts to replace the crossover operator in GA by a special type of differential operator for reproducing offspring in the next generation.*

### **B.1 Ant Colony Systems: An Overview**

Insects like ants, bees, wasps and termites are quite social. They live in colonies and follow their own routine of tasks independent of each other. However, when

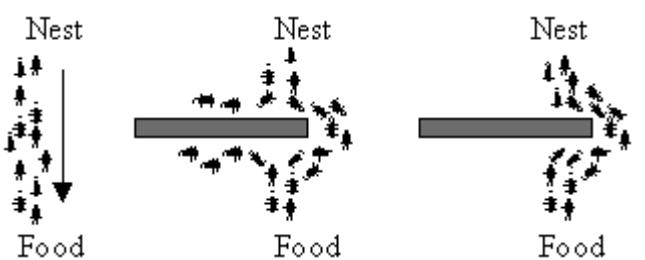
---

<sup>1</sup> Appendix-B is written by Swagatam Das, currently a graduate student of Electronics and Tele-Communication Engineering, Jadavpur University.

acting as a community, these insects even with very limited individual capability can jointly (cooperatively) perform many complex tasks necessary for their survival [1]. Problems like finding and storing foods, selecting and picking up materials for future usage require a detailed planning, and are solved by insect colonies without any kind of supervisor or controller.

It is a natural observation that a group of ‘almost blind’ ants can figure out the shortest route between a cube of sugar and their nest without any visual information. They are capable of adapting to the changes in the environment as well [7]. It is interesting to note that ants while crawling deposit trails of a chemical substance known as pheromone to help other members of their team to follow its trace. The resulting collective behavior can be described as a loop of positive feedback, where the probability of an ant’s choosing a path increases as the count of ants that already passed by that path increases [7], [10].

The basic idea of a real ant system is illustrated in Fig. B.1. In the left picture, the ants move in a straight line to the food. The middle picture illustrates the situation soon after an obstacle is inserted between the nest and the food. To avoid the obstacle, initially each ant chooses to turn left or right at random. Let us assume that ants move at the same speed depositing pheromone in the trail uniformly. However, the ants that, by chance, choose to turn left will reach the food sooner, whereas the ants that go around the obstacle turning right will follow a longer path, and so will take longer time to circumvent the obstacle. As a result, pheromone accumulates faster in the shorter path around the obstacle. Since ants prefer to follow trails with larger amounts of pheromone, eventually all the ants converge to the shorter path around the obstacle, as shown in the right picture.



**Fig. B.1:** Illustrating the behavior of real ant movements.

An artificial Ant Colony System (ACS) is an agent-based system, which simulates the natural behavior of ants and develops mechanisms of cooperation and learning. ACS was proposed by Dorigo et al. [8] as a new heuristic to solve combinatorial optimization problems. This new heuristic, called Ant Colony

Optimization (ACO), has been found to be both robust and versatile in handling a wide range of combinatorial optimization problems.

### B.1.1 The ACO Algorithm

The main idea of ACO is to model a problem as the search for a minimum cost path in a graph. Artificial ants act as if they walk on this graph, looking for cheaper paths. Each ant has a rather simple behavior capable of finding relatively costlier paths. Cheaper paths are found as the emergent result of the global cooperation among ants in the colony. The behavior of artificial ants is inspired from real ants: they lay pheromone trails (obviously in a mathematical form) on the graph edges and choose their path with respect to probabilities that depend on pheromone trails. These pheromone trails progressively decrease by evaporation. In addition, artificial ants have some extra features not seen in their counterpart in real ants. In particular, they live in a discrete world (a graph) and their moves consist of transitions from nodes to nodes.

Pheromone placed on the edges acts like a *distributed long term memory* [8]. The memory, instead of being stored locally within individual ants, remains distributed on the edges of the graph. This indirectly provides a means of communication among the ants called *stigmergy* [4]. In most cases, pheromone trails are updated only after having constructed a complete path and not during the walk, and the amount of pheromone deposited is usually a function of the quality of the path. Finally, the probability for an artificial ant to choose an edge not only depends on pheromones deposited on that edge in the past, but also on some problem dependent local heuristic function.

### B.1.2 Solving the Classical TSP Problem by ACO

Below we illustrate the use of ACO in finding the optimal tour in the classical Traveling Salesman Problem (TSP). Given a set of  $n$  cities and a set of distances between them, the problem is to determine a minimum traversal of the cities and return to the home-station at the end. It is indeed important to note that the traversal should in no way include a city more than once. Let  $r(C_x, C_y)$  be a measure of cost for traversal from city  $C_x$  to  $C_y$ . Naturally, the total cost of traversing  $n$  cities indexed by  $i_1, i_2, i_3, \dots, i_n$  in order is given by the following expression:

$$\text{COST}(i_1, i_2, \dots, i_n) = \sum_{j=1}^{n-1} r(C_{i_j}, C_{i_{j+1}}) + r(C_{i_n}, C_{i_1}) \quad (\text{B.1})$$

The ACO algorithm is employed to find an optimal order of traversal of the cities. Let  $\tau$  be a mathematical entity modeling the pheromone and  $\eta_{ij} = 1/r(i, j)$  is a local heuristic. Also let  $\text{allowed}_k(t)$  be the set of cities that are yet to be

visited by ant  $k$  located in city  $i$ . Then according to the classical ant system [6] the probability that ant  $k$  in city  $i$  visits city  $j$  is given by

$$P_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha [\eta_{ij}(t)]^\beta}{\sum_{h \in \text{allowed}_k(t)} [\tau_{ih}(t)]^\alpha [\eta_{ih}(t)]^\beta}, \text{ if } j \in \text{allowed}_k(t) \quad (\text{B.2})$$

$= 0, \text{ otherwise.}$

In (B.2) shorter edges with greater amount of pheromone are favored by multiplying the pheromone on edge  $(i, j)$  by the corresponding heuristic value  $\eta(i, j)$ . Parameters  $\alpha (> 0)$  and  $\beta (> 0)$  determine the relative importance of pheromone versus cost.

Now in ant system pheromone trails are updated as follows. Let  $D_k$  be the length of the tour performed by ant  $k$ ,  $\Delta\tau_k(i, j) = 1/D_k$  if  $(i, j) \in$  tour done by ant  $k$  and  $= 0$  otherwise and finally let  $\rho \in [0, 1]$  be a pheromone decay parameter which takes care of the occasional evaporation of the pheromone from the visited edges. Then once all ants have built their tours, pheromone is updated on all the ages as,

$$\tau(i, j) = (1 - \rho) \cdot \tau(i, j) + \sum_{k=1}^m \Delta\tau_k(i, j). \quad (\text{B.3})$$

From equation (B.3), we can guess that pheromone updating attempts to accumulate greater amount of pheromone to shorter tours (which corresponds to high value of the second term in B.3 so as to compensate for any loss of pheromone due to the first term). This conceptually resembles a reinforcement-learning scheme, where better solutions receive a higher reinforcement.

The ACS [8] differs from the classical ant system in the sense that here the pheromone trails are updated in two ways. Firstly, when ants construct a tour they locally change the amount of pheromone on the visited edges by a local updating rule. Now if we let  $\gamma$  to be a decay parameter and  $\Delta\tau(i, j) = \tau_0$  such that  $\tau_0$  is the initial pheromone level, then the local rule may be stated as,

$$\tau(i, j) = (1 - \gamma) \cdot \tau(i, j) + \gamma \cdot \Delta\tau(i, j) \quad (\text{B.4})$$

Secondly, after all the ants have built their individual tours, a global updating rule is applied to modify the pheromone level on the edges that belong to the best ant tour found so far. If  $\kappa$  be the usual pheromone evaporation constant,  $D_{gb}$  be the length of the globally best tour from the beginning of the trial and

$\Delta\tau'(i, j) = 1/D_{gb}$  only when the edge ( $i, j$ ) belongs to global-best-tour and zero otherwise, then we may express the global rule as

$$\tau(i, j) = (1 - \kappa) \cdot \tau(i, j) + \kappa \cdot \Delta\tau'(i, j) \quad (B.5)$$

The main steps of ACS algorithm are presented below.

### Procedure ACS

**Begin**

Initialize pheromone trails;

**Repeat** /\* at this stage each loop is called an iteration \*/

  Each ant is positioned on a starting node;

**Repeat** /\* at this level each loop is called a step \*/

  Each ant applies a *state transition rule like rule B. 2* to incrementally build a solution and a *local pheromone updating rule like rule B.4*;

**Until** all ants have built a complete solution;

  A *global pheromone updating rule like rule B.5* is applied.

**Until** terminating\_condition is reached;

**End.**

In the above algorithm, the first loop (iteration) starts with  $m$  ants being placed in  $n$  cities chosen according to some initialization rule (e.g. randomly). In the embedded loop (step) each ant builds a tour (i.e., an acceptable solution to the TSP) by repeatedly applying a stochastic state transition rule. While building its tour, the ant can modify the pheromone level on the visited edges by applying the local updating rule given by (B.4). Once all the ants have terminated their tour, the pheromone trails are modified again by the global updating rule given in (B.5).

### B.1.3 Application Domain of ACO

Ant Colony Optimization has successfully been applied to many NP hard combinatorial optimization problems including TSP. Researchers in [6], [8] and [9] have shown that the ACO can outperform GA in solving variants of TSP. Similar conclusions can also be drawn from the experiments on graph coloring problems [3], quadratic assignment problems [10], [14], and vehicle routing problems [2], [12].

Application of ACO in other fields of engineering is quiet on the rise. Schooerwoerd *et al.* [17] utilized ACO in the load-balancing problem for telecommunication networks. Di Caro and Dorigo [5] proposed the ‘AntNet’ algorithm for computer networks. In their work, intelligent packets – *ants* – are introduced in the networks, which interact to keep the contents of the routing tables up-to-date. There has been considerable research for utilizing the capacity of ACO in behavioral robotics as well.

## B.2 Differential Evolution

In 1995 Storn and Price took a serious attempt to replace the classical crossover and mutation operators in GA by alternative operators [18], and consequently found a suitable differential operator to handle the problem. They proposed a new algorithm based on this operator, and called it Differential Evolution (DE). The algorithm became popular soon within the domain of machine intelligence and cybernetics. The following two issues perhaps have a great role in the popularity of the algorithm. Firstly, DE is easy to implement, and thus is convenient to programmers of diverse domains. Secondly, unlike GA and PSO, where tuning of parameters is an additional problem, DE and its variants have one or a few parameters and thus tuning them is straight forward.

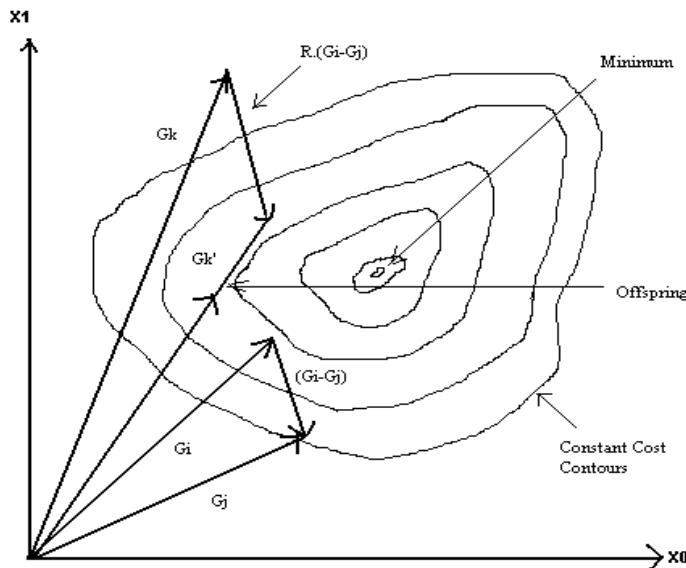
### B.2.1 Outline of the Algorithm

DE searches for a global optimum point in an N-dimensional hyperspace. It begins with a randomly initiated population of N dimensional real valued parameter vectors. Each vector, also known as ‘genome’, forms a candidate solution to the multi dimensional optimization problem. Unlike conventional GA, however the reproduction scheme in DE goes as follows. For each individual vector  $G_k^D$  belonging to generation D, randomly sample three other individuals  $G_i^D$ ,  $G_j^D$  and  $G_m^D$  from the same generation (for distinct  $i, j, k$  and  $m$ ), calculate the difference of the components (chromosomes) of  $G_i^D$  and  $G_j^D$ , scale it by a random scalar  $R \in [0,1]$  and create an offspring for the next generation by adding the result to the chromosomes of  $G_k^D$ .

$$\begin{aligned} G_{k,n}^{D+1} &= G_{m,n}^D + R.(G_{i,n}^D - G_{j,n}^D) && \text{if } rand_n(0, 1) < CR \\ &= G_{k,n}^D, && \text{otherwise.} \end{aligned} \quad (\text{B.6})$$

for the n-th component of each parameter vector.

The so called cross-over constant  $CR \in [0, 1]$  iteratively copies consecutive components of the difference vector to the offspring until the random number  $rand_n(0, 1)$  becomes greater than or equal to CR for some n. Parameters R and CR govern the convergence speed and robustness of DE. The process is illustrated in Fig. B.2. Closed curves in Fig. B.2 denote constant cost contours, i.e., for a given cost function f, a contour corresponds to  $f(x_0, x_1) = \text{constant}$ .



**Fig. B.2:** Illustrating DE in 2-D parameter space: When the difference between two parameter vectors  $G_i$  and  $G_j$  is scaled by a random scalar  $R$  and then added to another vector  $G_k$ , the offspring so produced i.e.,  $G_{k'}$  points to a lower cost contour and hence is better than the parent  $G_k$

The DE algorithm is outlined below:

#### Procedure Differential-evolution

**Begin**

    Initialize population;

    Evaluate fitness;

**For**  $i=0$  to max-iteration **do**

**Begin**

            Create Difference-Offspring;

            Evaluate fitness;

**If** an offspring is better than its parent

**Then** replace the parent by offspring in the next generation;

**End If;**

**End For;**

**End.**

In the above algorithm, population is at first initialized to random values and fitness of each vector is judged according to some predefined cost function. The algorithm is then continued to generate population by invoking differential

evolution and replacing parents by more fit offspring. The algorithm terminates when the fitness of the best genome is greater than a predefined value or maximum number of iterations has been attained.

### B.2.2 Application Domain of DE and Recent Research

Differential evolution (DE), first introduced in [18], is successfully applied to many artificial and real optimization problems and applications [13], such as aerodynamic shape optimization [16], automated mirror design [11], optimization of radial active magnetic bearings [19], optimization of fermentation by using a high ethanol -tolerance yeast [20].

A differential evolution based neural-network training algorithm was first introduced in [15]. In [21] the method's characteristics as a global optimizer were compared to other neural network training methods. Krink et al. have compared the performance of DE with other common optimization algorithms like PSO, GA etc. in context to the partitional clustering problem and concluded in their study that DE rather than GAs should receive primary attention in such partitional cluster algorithms. Recently Zhang and Xie have proposed an efficient version of hybrid Particle Swarm Optimizer equipped with differential evolution operator [22] to provide the bell-shaped mutations with consensus on the population diversity along with the evolution. It also preserves the self organized particle swarm dynamics all along.

## References

- [1] Bonabeau, E., Dorigo, M. and Theraulaz, G., *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, NY, 1999.
- [2] Bullnheimer, B., Hartl, R. F. and Strauss, C. “An improved ant system algorithm for the vehicle routing problem,” *Ann. Oper. Res.*, vol. 89, pp. 319–328, June 1999.
- [3] Costa, D. and Hertz, A. “Ants can color graphs,” *J. Oper. Res. Soc.*, vol. 48, no. 3, pp. 295–305, Mar. 1997.
- [4] Deneubourg, J. L., “Application de l'ordre par fluctuations à la description de certaines étapes de la construction d'un nid chez les termites,” *Insect Sociaux*, vol. 24, pp. 117–130, 1977.

- 
- [5] Di Caro, G., Dorigo, M., "AntNet: Distributed Stigmergetic Control for Communications Networks," *Journal of Artificial Intelligence Research*, 9, pp 317-365, 1998.
  - [6] Dorigo, M., *Optimization, Learning, and Natural Algorithms*, Ph.D. dissertation (in Italian), Dipartimento di Elettronica, Politecnico di Milano, Milano, Italy, 1992.
  - [7] Dorigo, M. and Gambardella, L. M., "A Study of Some Properties of Ant Q", *Proc. of 4<sup>th</sup> Int. Conf. Parallel Problem Solving From Nature (PPSN-IV)*, Berlin, Germany: Springer-Verlag, pp. 656-665, 1996.
  - [8] Dorigo, M. and Gambardella, L. M., "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Trans. Evol. Comput.*, vol. 1, pp. 53–66, Apr. 1997.
  - [9] Dorigo, M., Maniezzo, V. and Colorni, A., "The ant system: Optimization by a colony of cooperating agents," *IEEE Trans. on System, Man and Cybernetics- B*, vol. 26, Feb. 1996.
  - [10] Dorigo, M., Di Caro, G. and Gambardella, L. M., "Ant algorithms for discrete optimization," *Artif. Life*, vol. 5, no. 2, pp. 137–172, 1999.
  - [11] Doyle, S., Corcoran, D. and Connell, J., "Automated mirror design using an evolution strategy," *Optical Engineering*, vol. 38, no. 2, pp. 323–333, 1999.
  - [12] Gambardella, L. M., Taillard, E.D. and Agazzi, G., MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows, In *New Ideas in Optimization*, Corne, D., Dorigo, M. and Glover, F. (Eds.), McGraw-Hill, London, pp. 63–76, 1996.
  - [13] Lampinen, J., A Bibliography of Differential Evolution Algorithm. <http://www.lut.fi/~jlampine/debiblio.htm>, 2001.
  - [14] Maniezzo, V. and Colorni, A., "The ant system applied to the quadratic assignment problem," *IEEE Trans. on Knowledge and Data Eng.*, vol. 11, pp. 769–778, Sept./Oct., 1999.
  - [15] Masters, T. and Land, W., "A new training algorithm for the general regression neural network," *Proc. of Computational Cybernetics and*

- Simulation*, Organized by IEEE Systems, Man, and Cybernetics Society, no. 3, pp.1990–1994, 1997.
- [16] Rogalsky, T., Kocabiyik, S. and Derksen, R., “Differential evolution in aerodynamic optimization,” *Canadian Aeronautics and Space Journal*, vol. 46, no. 4, pp. 183–190, 2000.
  - [17] Schoonderwoerd, R., Holland, O., Bruten, J. and Rothkrantz, L., “Ant-based load balancing in telecommunication networks,” *Adaptive Behavior*, Vol. 5, no. 2, 1997.
  - [18] Storn, R. and Price, K., *Differential Evolution – A Simple and Efficient Adaptive Scheme for Global Optimization Over Continuous Spaces*, Technical Report TR-95-012, Berkeley, CA, USA, 1995.
  - [19] Stumberger, G., Dolinar, D., Pahner, U. and Hameyer, K., “Optimization of radial active magnetic bearings using the finite element technique and differential evolution algorithm,” *IEEE Transactions on Magnetics*, vol. 36, no. 4, pp.1009–1013, 2000.
  - [20] Wang, F.-S. and Sheu, J.-W., “Multiobjective parameter estimation problems of fermentation processes using a high ethanol tolerance yeast,” *Chemical Engineering Science*, vol. 55, no. 18, pp. 3685–3695, 2000.
  - [21] Zelinka, I. and Lampinen, J., “An evolutionary learning algorithms for neural networks,” *Proc. of 5th International Conference on Soft Computing (MENDEL'99)*, pp.410–414, 1999.
  - [22] Zhang, W. and Xie, X., “DEPSO: Hybrid Particle Swarm with Differential Evolution Operator,” *Proc. of IEEE Int. Conf. on Systems, Man & Cybernetics (SMCC)*, Washington D C, USA, pp. 3816-3821, 2003.

# Index

## A

ACO algorithm 693  
Action estimation 315  
Action group estimation 316  
Activation function, 10  
ADALINE neuron 210  
Adaptive bi-directional associative memory theorem 182  
Adaptive channel equalization 659  
Adaptive FCMs 504  
Adaptive resonance theory 256  
Alpha-cut 51  
AND-clauses 69  
Ant colony systems 691  
Approximate reasoning 83  
Approximately correct 24  
Artificial immune systems 612, 620  
Artificial Intelligence 1  
Artificial life 612, 613  
Artificial neural networks 169  
Available bit rate (ABR) queue length 559  
Axelord's model 498

## B

Back-propagation algorithm 217  
Back-propagation-RBF Synergism 660  
Bandwidth 564  
Basic probability assignment 373  
Bayes' law 357  
Bayesian belief network 361  
Behavioral synergism 453  
Belief calculus 353  
Belief function 23, 373  
Belief networks 20  
Belief vector  
Benchmark analysis 584

Bi-directional linked list 645  
Binarization process 578  
Bioinformatics 666  
Biological neural networks 167

## C

Call admission control 549  
Causal tree 361  
Certainty factor vector  
Chaos theory 7, 612  
Classifier design 632  
Cognition 497  
Cohen-Grossberg theorem 180  
Cohen-Grossberg-Kosko theorem 182  
Communication engineering 655  
Competitive learning 16, 175, 269  
Computational Intelligence 4  
Computational learning theory 7, 23  
Computationally intelligent 5  
Conditional probability 356  
Conflict resolution 69  
Congestion control 549  
Conjunction of fuzzy propositions 156  
Continuous Hopfield neural net 241  
Contrast intensification 62  
Conventional sets 37  
Crossover 18, 330  
Crossover point 18  
Cylindrical extension 57  
Cylindrical extension of fuzzy relations 153

## D

Data Fusion 4

Decomposition 227  
 Defining length of a schema 332  
 Defuzzification 119  
 Degree of fulfillment 115  
 Delay 564  
 Delta learning rule 478  
 Dempster-Shafer theory 372  
 Design of fuzzy relational databases 162  
 Diagnostic tree 363  
 Dienes-Rescher implication 72  
 Differential evolution 696  
 Discounting factor 297  
 Discrete Hopfield network 239  
 Dominant arc 406  
 Drug classification 664

## E

Edge 425  
 Encoding 15  
 Equality as a fuzzy relation 158  
 Equalized pulses 659  
 Equivalent relation 627  
 Equilibrium time 406  
 Evolutionary computing 6  
 Evolutionary computing algorithm 324  
 Expert networks 228  
 Extension principle of fuzzy sets 52

## F

f.g generalization 10  
 Face recognition 284  
 Fitness function 327  
 Focal elements 373  
 Frame of discernment 373  
 Functional dependency preservation 144  
 Fuzzy abductive reasoning 90  
 Fuzzy ADALINE 477  
 Fuzzy associative memory  
 Fuzzy bandwidth estimator 566  
 Fuzzy beliefs 395  
 Fuzzy chaos theory 624

Fuzzy C-means clustering 649  
 Fuzzy C-means clustering algorithm 127  
 Fuzzy cognitive maps 497  
 Fuzzy complement 50  
 Fuzzy congestion controller 555  
 Fuzzy control system 101  
 Fuzzy data mining 652  
 Fuzzy functional dependency 158  
 Fuzzy implication relations 71  
 Fuzzy implication rules 70  
 Fuzzy integrity constraints 155  
 Fuzzy linguistic hedges 60  
 Fuzzy logic 10, 73  
 Fuzzy lossless join 161  
 Fuzzy moment descriptors 429  
 Fuzzy pattern recognition 126  
 Fuzzy Petri nets 393, 396  
 Fuzzy relations 54  
 Fuzzy sets 38, 39, 40  
 Fuzzy shade moment 430  
 Fuzzy S-Norm 49  
 Fuzzy state equations 650  
 Fuzzy state variables 650  
 Fuzzy T-Norm 48  
 Fuzzy truth token 396  
 Fuzzy-GA synergism 26, 459

## G

GA-Belief network synergism 27,461  
 Gama-function 43  
 Gaussian membership function 47  
 Generalized hypothetical syllogism 75  
 Generalized modus ponens 74  
 Generalized modus tollens 75  
 Genetic algorithm 17, 323  
 Goal angle 574  
 Godel implication 73  
 Granular computing 612, 630

## H

Hebbian learning 269, 276

Heuristic function 645  
Heuristic transitive closure 506  
Hypothetical syllogism 74

## I

Image distance 431  
Image matching 423  
Image segmentation 132, 441  
Immediately reachable 398  
Inference engine 68  
Instar connectivity 273  
Intelligent traffic control 565  
Interleaved Search 435  
Internal critic 16  
Intersection operation 38  
Inverse Modeling 658

## J

Jerne's idiotypic network hypothesis 621  
Jitter 564

## K

Kalman filter 595, 596  
Klopff's model 178  
Knowledge refinement 535  
Kosko's extended model 503  
Kosko's model 500

## L

Lamda and Pi messages 368  
Learning 170  
L-function 45  
Limitcycles 406  
Linguistic variables 117  
Local stability 651  
Logic of fuzzy sets 9  
Loss rate 564  
Lossy fuzzy joins 155  
Lukasiewicz implication 72  
Lyapunov energy function 109

## M

Macro cell placement 662  
Mamdani implication 72  
Mamdani type fuzzy control 101  
Markov process 308  
Max-min composition 58  
Max-min fuzzy inverse 643  
Max-product composition 58  
McCulloch-Pitts model 198  
Mean path deviation 582  
Membership function 40  
Mixed range 425  
Mobile robotics 571  
Modifier rule 157  
Modular neural nets 225  
Modus Ponens 9, 74  
Modus Tollens 74  
Monoamine oxidase 664  
Mood analysis 442  
Multi-Agent reinforcement learning 315  
Multilayered perceptron 207  
Multivalued logic 9  
Mutation 18, 331

## N

Natural join of fuzzy relations 154  
Negative Positive Neutral logic 506  
Network paralysis 222  
Neural reinforcement learning 311  
Neuro-Belief network synergism 26  
Neuro-dynamic programming 295  
Neuro-Fuzzy synergism 26  
Neuro-Fuzzy-GA synergism 27  
Neuro-GA synergism 26, 460  
No-linear prediction problem 657  
Non-deterministic rewards 308  
Nonmonotonic reasoning 414  
Non-uniform sampling 42  
Normal 51  
NP-complete problem 573  
Nuclear reactor 114

**O**

Object localization 589  
 Oja's correction 278  
 Order of a schema 332  
 Orthogonal summation of belief functions 375

**P**

Pal and Konar's model 507  
 Partial matching 3  
 Partially observable states 298  
 Particle swarm optimization 612, 617  
 Path planning of a robot 573  
 Path-genetic operator 549  
 Pearl's belief propagation scheme 369  
 Perception-to-Action transformation 464  
 Perceptron learning model 201  
 Performance index 584  
 Permanently dominant arc 406  
 Pi-function 47  
 Place to transition connectivity 407  
 Pointer to array 645  
 Polytree 367, 369  
 Possibility distribution 146  
 Power control 114  
 Power fraction 115  
 Power spectral density 563  
 Predicted goal position 594  
 Pre-processing layer 272  
 Principal component analysis 281  
 Probably approximately correct 24  
 Process control 97  
 Production rules 68  
 Production Systems 67  
 Projection of fuzzy relations 56, 152  
 Projection operator 143  
 Proper subset 38  
 P-th percentile delay 564

**Q**

Q-learning 302  
 Quality of service 561  
 Quantizer 659  
 Queue growth rate 559

**R**

Radial basis function 223  
 Random variable 355  
 Reachable 398  
 Reactor period 115  
 Reasoning path 397  
 Recall 15  
 Recency 69  
 Recognition-act cycle 69  
 Recurrent neural nets 238  
 Refractoriness 69  
 Reinforcement learning 16, 177, 298  
 Relational model 140  
 Rod height 115  
 Rough inclusion and indiscernibility 632  
 Rough set theory 7  
 Rough sets 612  
 Rough-Fuzzy sets 629  
 Routing 548  
 Routing in VLSI design 662, 663

**S**

Schema 331  
 Schema theorem 333  
 Schur matrix 111  
 Sejnowski's model 178  
 Selection 18  
 Selection operator 143  
 Self-Organizing feature map 282  
 s-function 44  
 Shade 425  
 Signal 115  
 Signal processing 655  
 Soft computing 8  
 Specificity 69

Spoken word reconstruction 649  
State of deadlock 403  
Stationary assumption 24  
Strong alpha-cut 51  
Strongly coupled 25  
Structural conflict 394  
Supervised learning 15, 171  
Synergism in soft computing 25  
System modeling 656

## T

Target tracking 592  
Tele-Communication networks 547  
Template matching 435  
Temporal difference learning 310  
Tightly coupled neuro-fuzzy systems 456  
Transition to place connectivity 407  
Transitive relation 506  
Trapping at local minima 222  
Travelling salesman problem 246  
Traversal distance 582  
Traversal time 582  
Triangular membership 46  
T-S Fuzzy control 106

## U

Uniform sampling 42  
Union operation 38  
Universal set 38  
Universe of discourse 42  
Unsupervised classification/learning 126  
Unsupervised learning 16, 172

## W

Weakly coupled 25  
Weakly coupled neuro-fuzzy systems 455  
Working memory 68

## Z

Zadeh implication 73  
Zhang, Chen and Bezdek's model 505

## About the Author

Amit Konar is presently a Reader in the Department of Electronics and Tele-Communication Engineering, Faculty of Engineering and Technology, Jadavpur University, Calcutta, India, and Joint Coordinator, Center for Cognitive Science, Jadavpur University. He has been teaching and carrying out research work at this University for the past 18 years. Jadavpur University is one of the top three universities in India. The faculty of Engineering and Technology, with more than 100 externally funded research projects, earned recognition as a “Center of excellence” from the Government of India, and the university got the “Five Star” rating. The doctoral, master’s, and bachelor’s programs offered by the university are acknowledged as one of the very best in the country. Books by professors of the university are used as graduate-level texts in Asian, European and American universities. The university runs collaborative programs with European and Canadian universities. Amartya Sen, the 1998 Nobel Laureate in Economics, taught at Jadavpur University for some time.

Dr. Konar’s research areas include the study of computational intelligence algorithms and their applications to the entire domain of Electrical Engineering and Computer Science. Specifically, he worked on fuzzy sets and logic, neuro-computing, genetic algorithms, Dempster-Shafer theory and Kalman filtering, and applied the principles of computational intelligence in image understanding, control engineering, VLSI CAD, mobile robotics, bio-informatics and mobile communication systems.

Dr. Konar has published over seventy research papers and several books & invited book chapters on various aspects of computer science and control engineering. His books/ book chapters have been (are in the process of being) published from top publishing houses such as Springer-Verlag, CRC Press, Physica-Verlag, Academic Press, Kluwer Academic Press and Prentice-Hall of India. He regularly provides peer review for journals in his field (e.g., journals from IEEE, Kluwer and Elsevier), and has frequently been invited to review books published by Springer-Verlag and McGraw-Hill. In recognition of his teaching and research, he has been given the AICTE Career Award (1997-2000), the highest honor offered to young talented academicians by the All India council of Technical Education, Government of India.

Dr. Konar is a Principal Investigator or Co-Principal Investigator of four external projects funded by University Grants Commission (the UGC is one of the main federal funding agencies in India) and two projects funded by the All India Council of Technical Education, Government of India. The research areas of these projects include decision support system for criminal investigation, navigational planning for mobile robots, AI and image processing, neural net based dynamic channel allocation, human mood detection from facial expressions and DNA-string matching algorithms. Under his supervision, seven graduate students have already earned the degree of Ph.D., and three Ph.D. dissertations are in progress. Currently, he serves on the editorial board of the

International Journal of Hybrid Intelligent Systems. Dr. Konar is the coordinator of the image processing part of the “Second Hooghly Bridge Project”, one of the major projects of West Bengal Government.

Dr. Konar served as a member of Program Committee of several International Conferences and workshops, such as Intl. Conf. on Hybrid Intelligent Systems (HIS 2003), held in Adelaide, Australia and Int. Workshop on Distributed Computing (IWDC 2002), held in Calcutta.